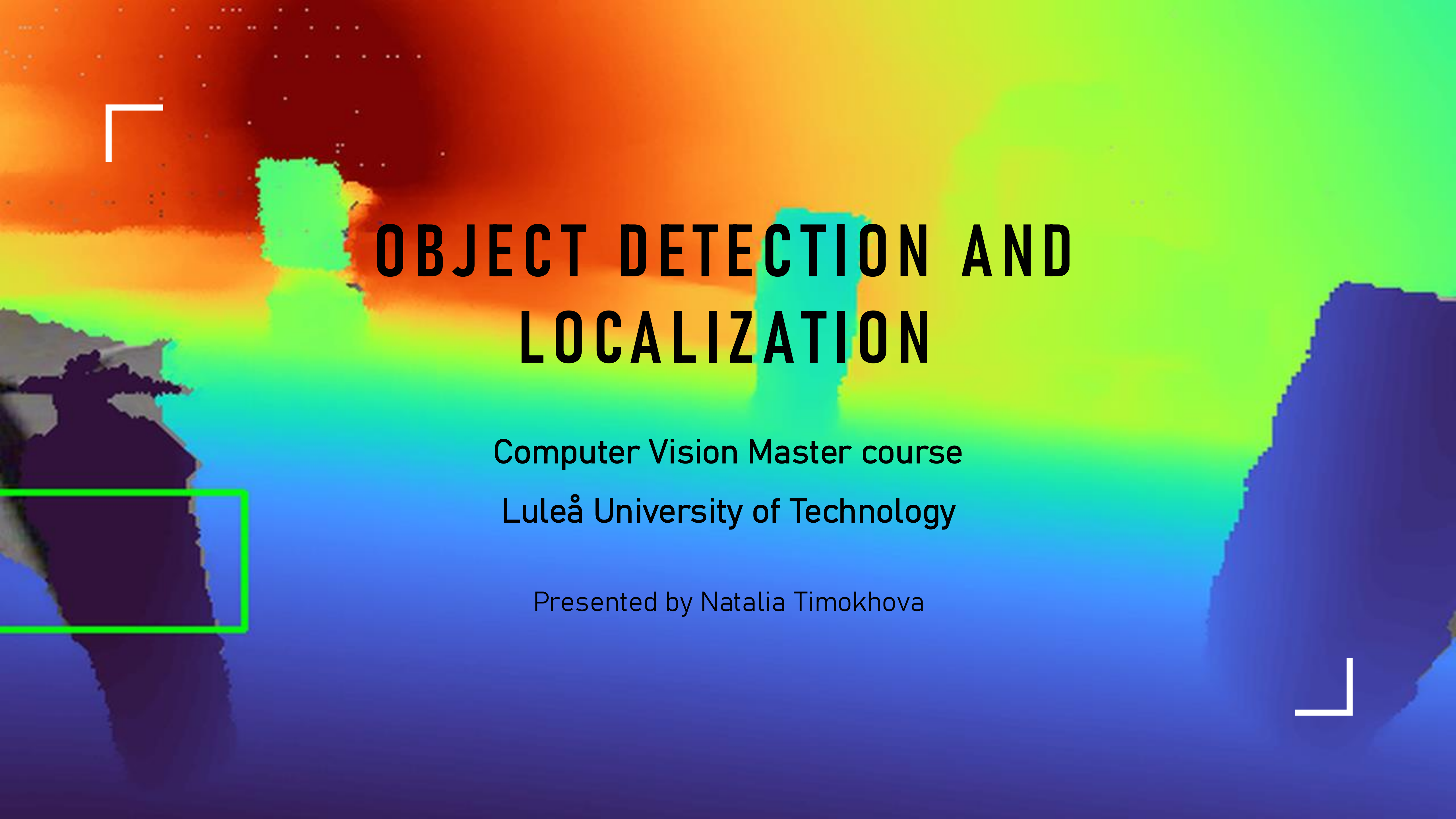


OBJECT DETECTION AND LOCALIZATION

Computer Vision Master course

Luleå University of Technology

Presented by Natalia Timokhova

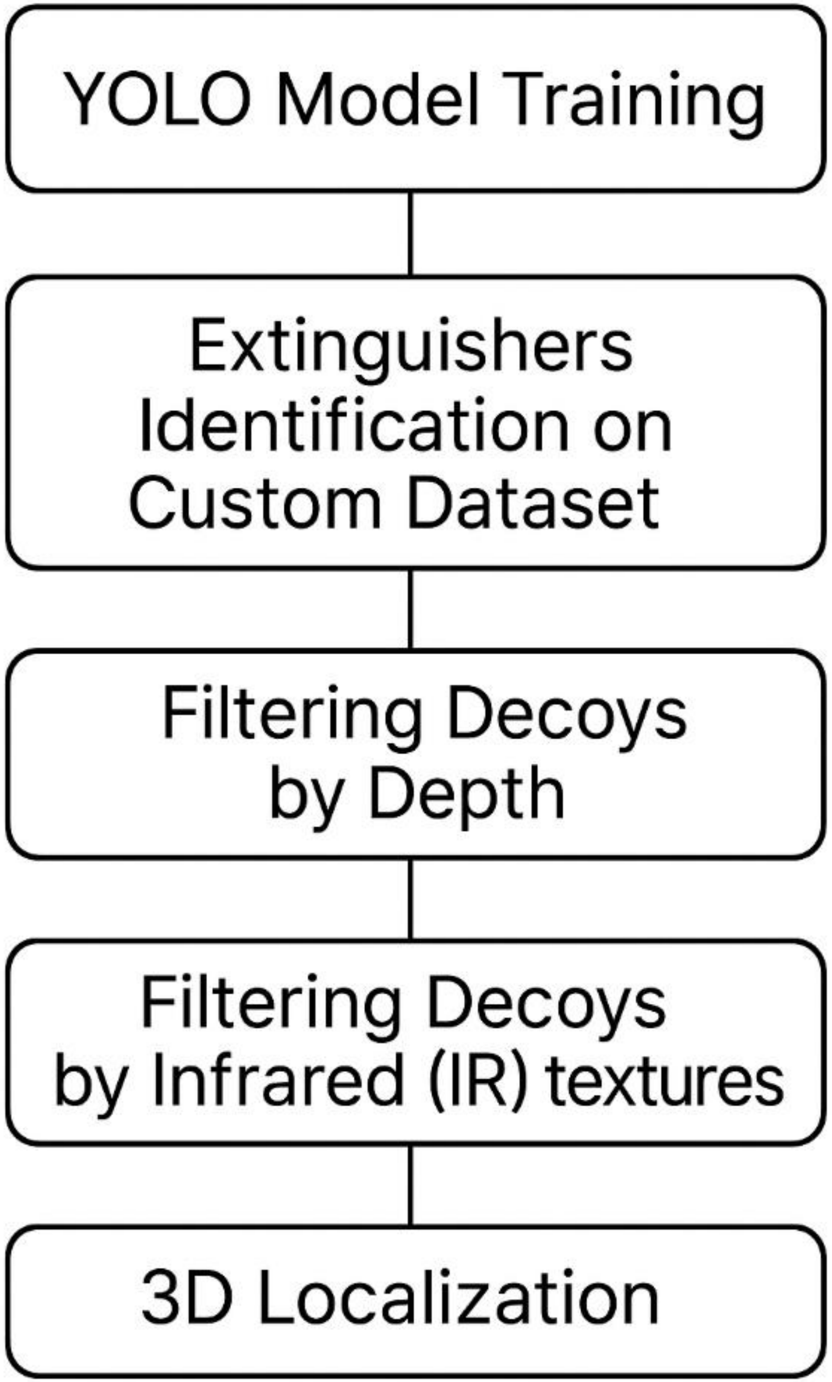
PROJECT

Objective:

Build a system that detects specific objects (fire extinguishers) in real-time, while ignoring printed replicas (decoys). The system should also estimate the 3D position of the real objects.

Requirements:

- Implement an object detection model (YOLO), trained on the provided dataset.
- Design a filtering mechanism to exclude decoy objects.
- Estimate the 3D coordinates of detected objects using stereo or depth sensors.



YOLO Model Training

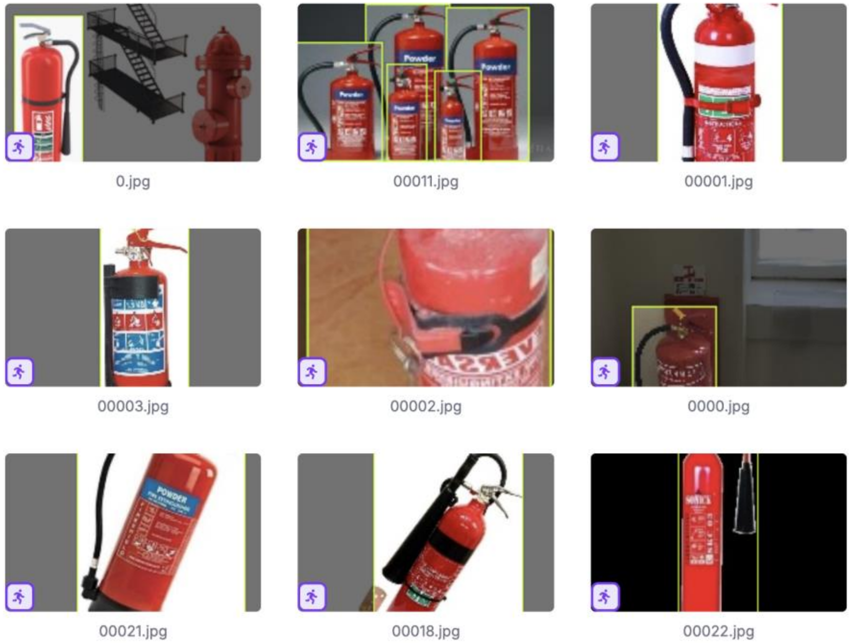
Extinguishers Identification on Custom Dataset

Filtering Decoys by Depth

Filtering Decoys by Infrared (IR) textures

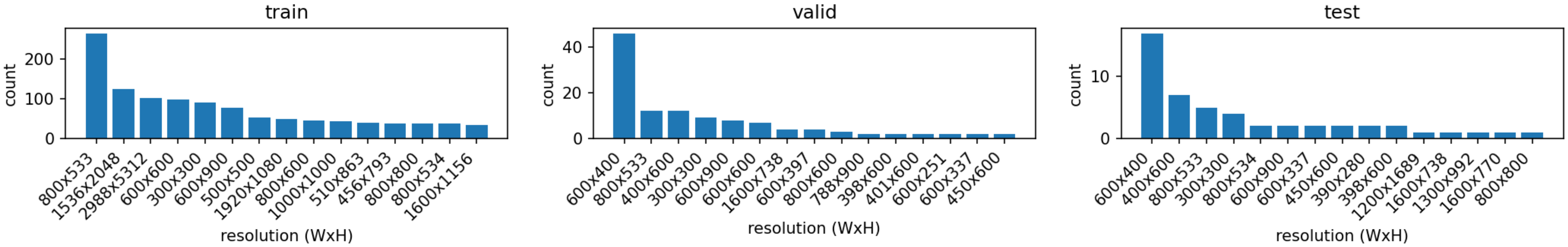
3D Localization

ROBOFLOW DATASET



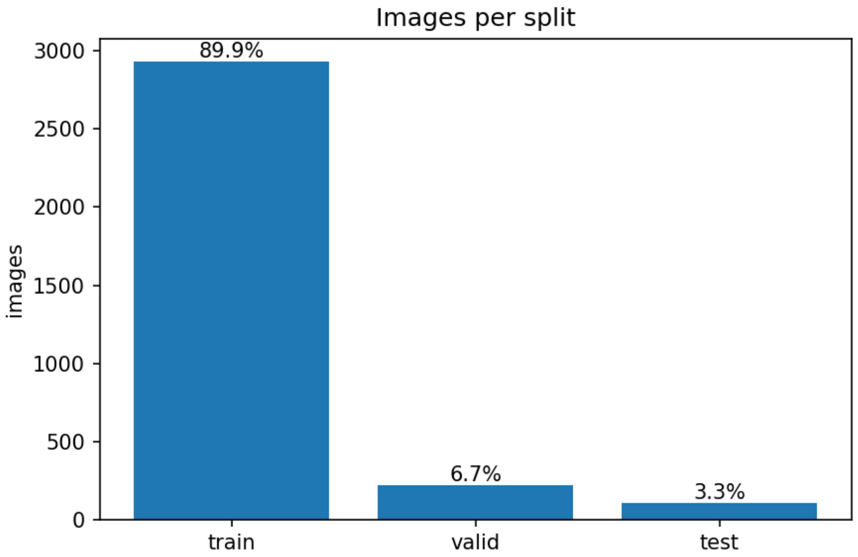
The dataset contains 3,262 images and includes a single class: **“Fire extinguisher.”** Images may contain either a single object or multiple objects, and they vary in resolution.

Image sizes – Top 15 per split



11/19/2025

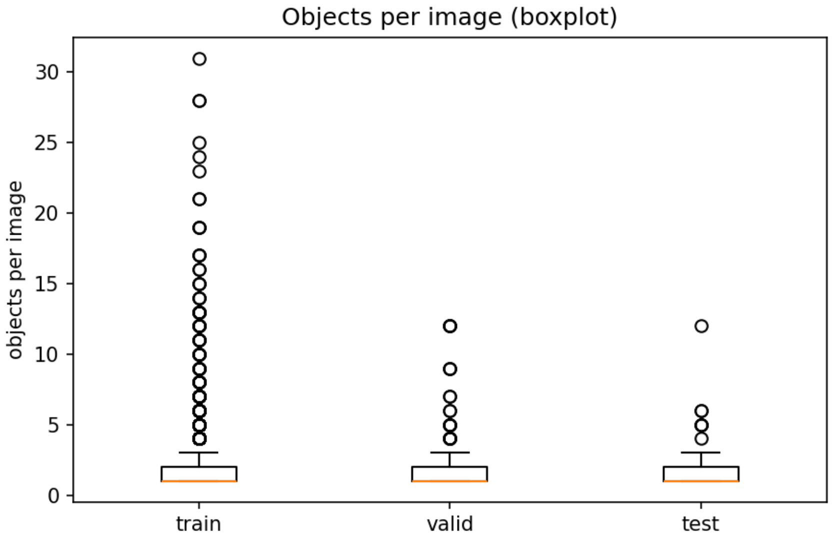
HTTPS://UNIVERSE.ROBOFLOW.COM/FIRE-EXTINGUISHER/FIREEXTINGUISHER-Z5ATR/BROWSE?QUERYTEXT=&PAGESIZE=50&STARTINGINDEX=0&BROWSEQUERY=TRUE

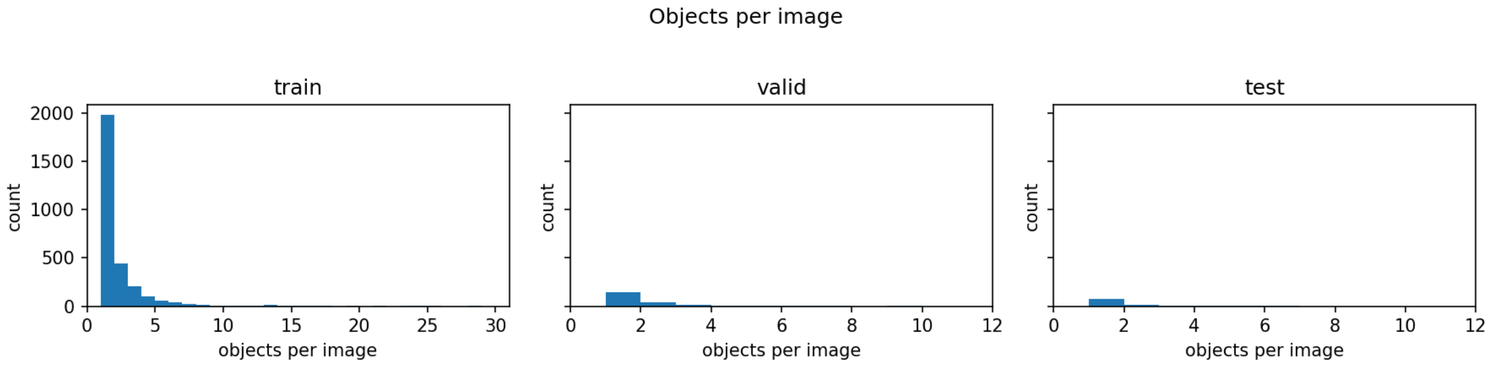


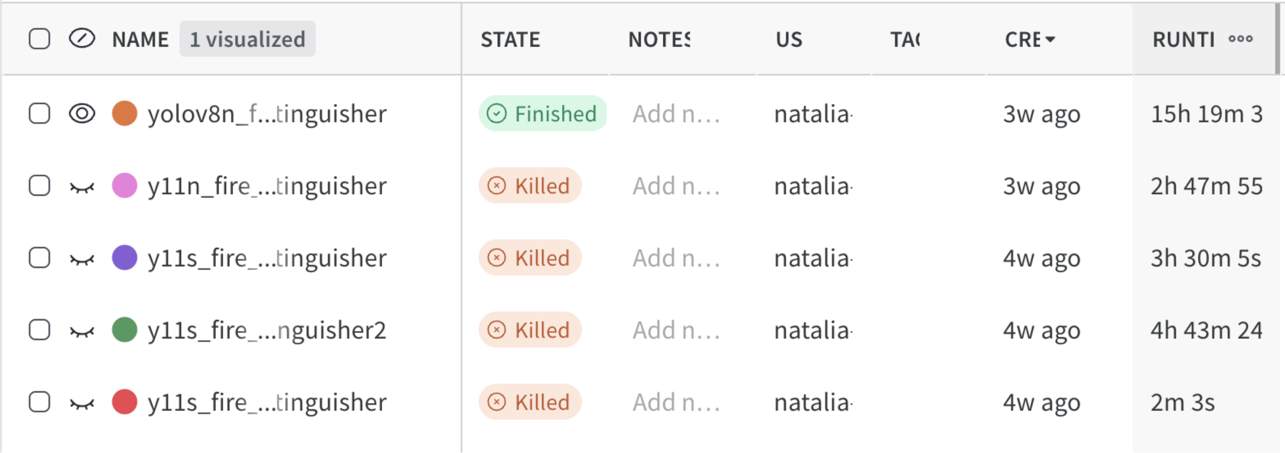
EXPLORATORY DATA ANALYSIS. ROBOFLOW DATASET

Initially, the dataset was split into train (90%) and validation (10%) subsets. However, to evaluate accuracy metrics and compare them with my custom dataset, I further restructured the data into train (90%), validation (6.7%), and test (3.3%) sets. This approach allowed me to keep 90% of the data for training, ensuring that the model had as many examples as possible.

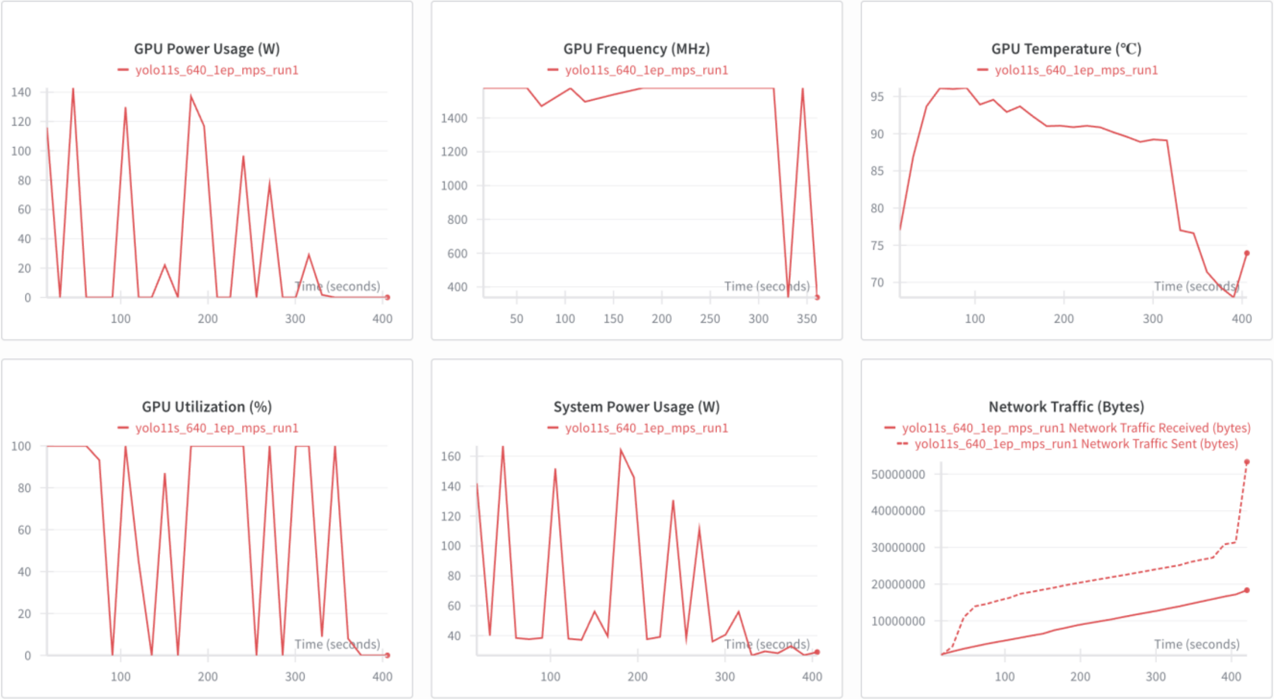
The average number of objects per image remains consistent across all subsets. However, because the training set is significantly larger, it naturally contains more outliers compared to the validation and test subsets.







□ ⊘ NAME 1 visualized	STATE	NOTES	US	TA(	CRE ▾	RUNTI ∘∘∘
□ yolov8n_f...tinguisher	Finished	Add n...	natalia		3w ago	15h 19m 3
□ y11n_fire_...tinguisher	Killed	Add n...	natalia		3w ago	2h 47m 55
□ y11s_fire_...tinguisher	Killed	Add n...	natalia		4w ago	3h 30m 5s
□ y11s_fire_...nguisher2	Killed	Add n...	natalia		4w ago	4h 43m 24
□ y11s_fire_...tinguisher	Killed	Add n...	natalia		4w ago	2m 3s



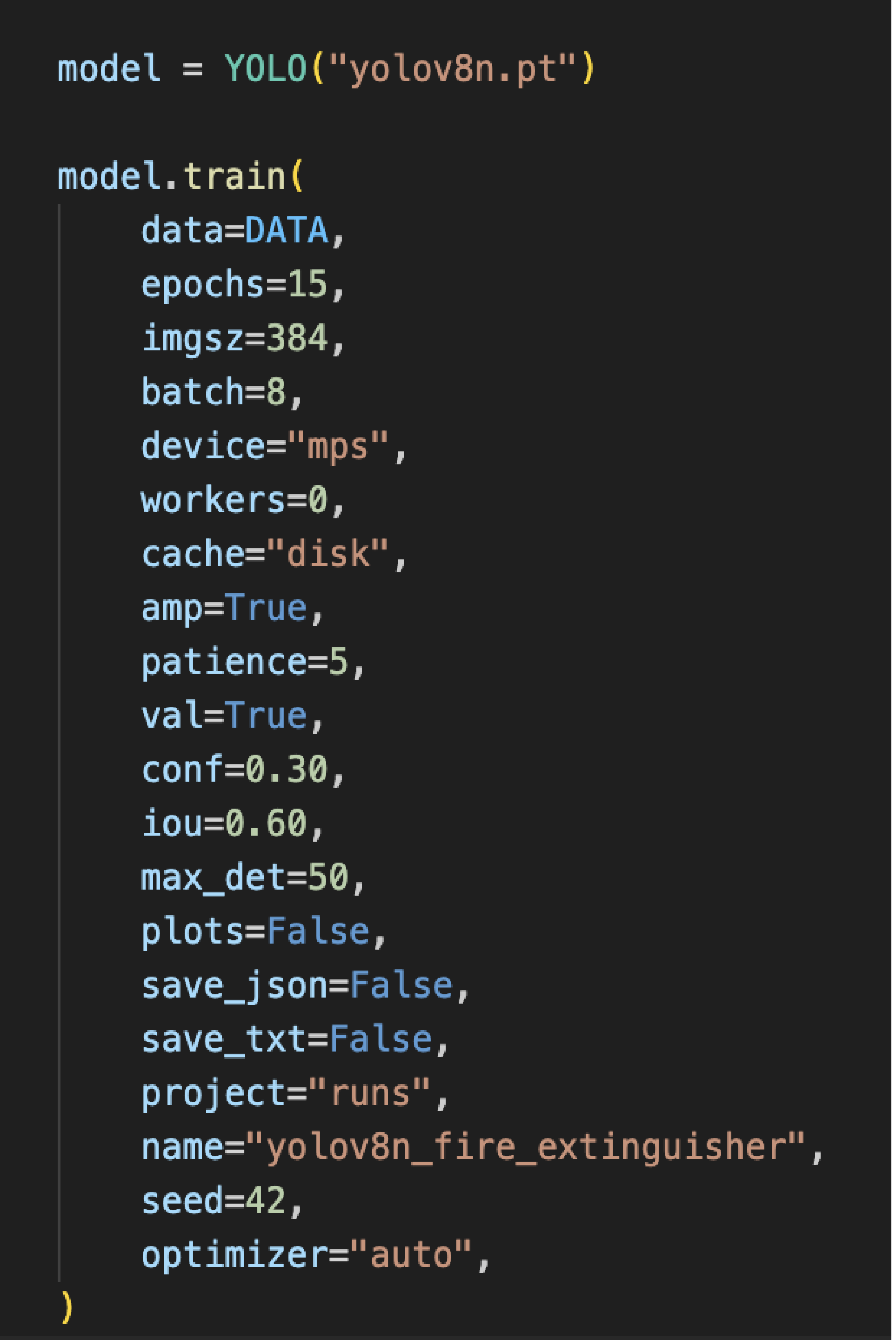
CHOICE OF THE OBJECT DETECTION MODEL

For training, the YOLO model was selected. Based on my hardware configuration (Apple M4, 16 GB memory, macOS 15.6.1 (24G90)), I first attempted to train YOLOv11 in the Small configuration. However, the training process crashed after a few hours. The same issue occurred with YOLOv11 in the Nano configuration.

As the next option, I switched to YOLOv8 in the Nano configuration, which completed successfully. The training ran without crashes and took 15.5 hours.

The main problem appeared during the validation stage. While the training itself was executed on the GPU, the validation was performed on the CPU, which created a significant performance bottleneck and increased the overall training time.

The charts show GPU usage and temperature during the YOLOv11-S training attempt, illustrating the hardware behaviour before the crash.



```
model = YOLO("yolov8n.pt")

model.train(
    data=DATA,
    epochs=15,
    imgsz=384,
    batch=8,
    device="mps",
    workers=0,
    cache="disk",
    amp=True,
    patience=5,
    val=True,
    conf=0.30,
    iou=0.60,
    max_det=50,
    plots=False,
    save_json=False,
    save_txt=False,
    project="runs",
    name="yolov8n_fire_extinguisher",
    seed=42,
    optimizer="auto",
)
```

TRAINING PIPELINE

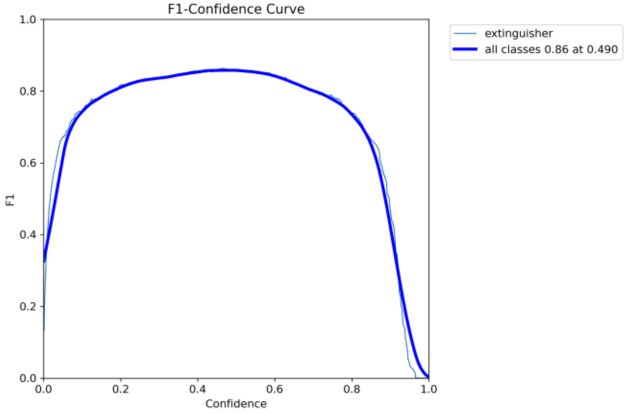
The **YOLOv8-nano** model was downloaded from Ultralytics. I integrated the Weights & Biases (wandb) dashboard to track GPU usage. The model parameters were selected to match the limited GPU/CPU capacity of the hardware.

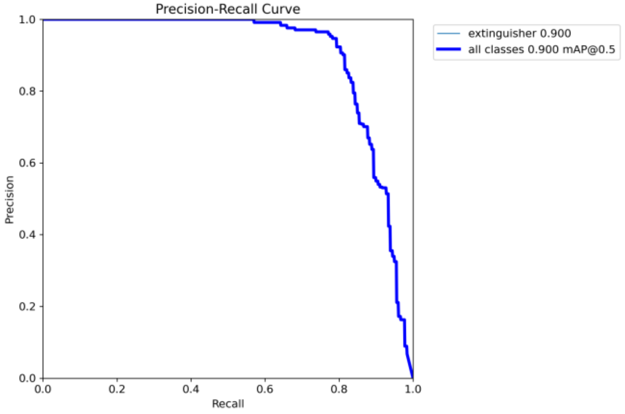
The model was trained for **15 epochs**, which is sufficient for a small, single-class dataset. All images were resized to **384×384**, a resolution that balances speed, memory use, and accuracy. A **batch size of 8** was selected to fit within the 16 GB memory of the Apple M4 GPU and ensure stable performance.

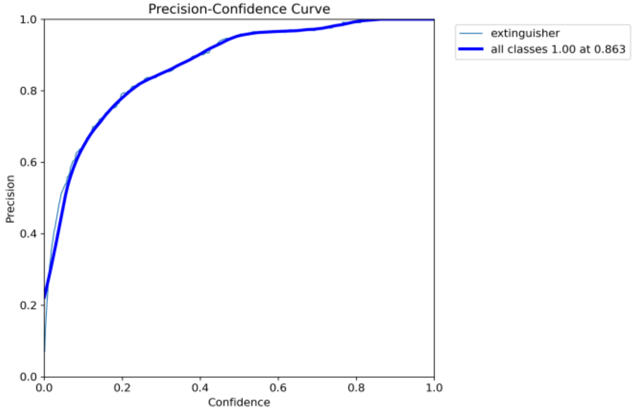
Training was executed on the **MPS backend** (Apple's GPU), with **0 dataloader workers** to avoid macOS/MPS multiprocessing issues. Using **cache="disk"** allowed faster data loading across epochs. **Automatic Mixed Precision (amp=True)** reduced memory usage and improved speed.

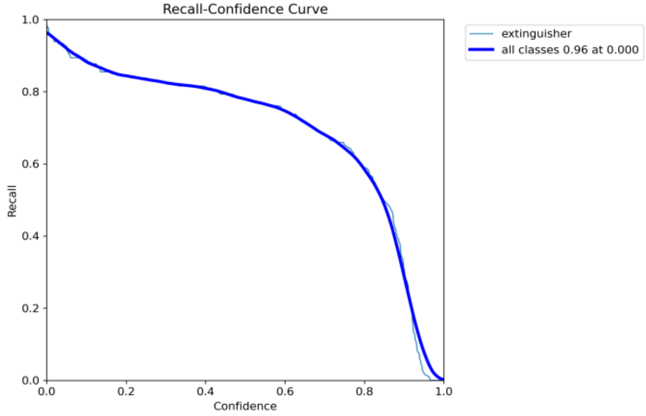
Early stopping was enabled with **patience=5**, meaning training stops if validation metrics do not improve for 5 epochs. **val=True** ensures validation is run after every epoch to compute metrics and support early stopping.
During evaluation, a **confidence threshold of 0.30** and **IoU threshold of 0.60** were used for NMS and mAP calculation (two boxes match if IoU ≥ 60%). The **max_det=50** parameter limits detections per image during validation. The **optimizer="auto"** setting lets Ultralytics automatically choose between AdamW and SGD.

EVALUATION METRICS









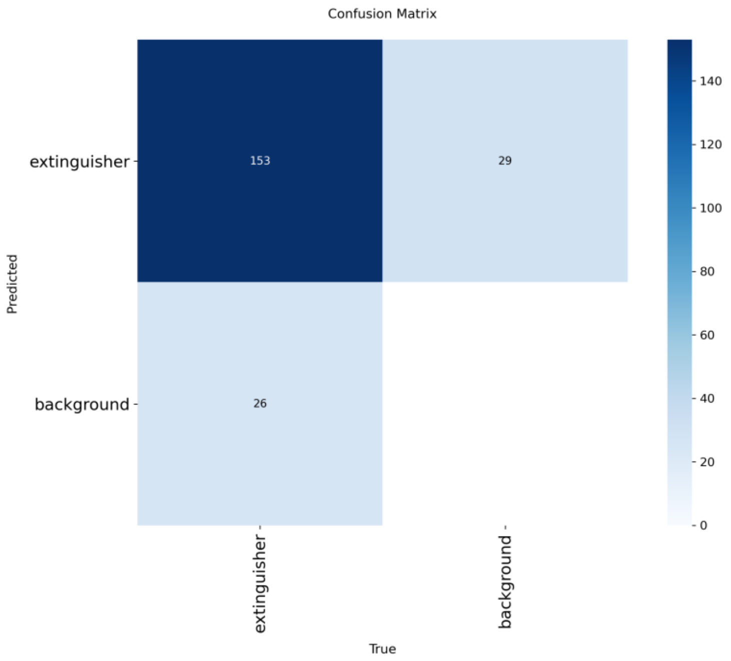
The model output includes the predicted class (0 in our case) and the bounding box parameters: the center coordinates (x, y), along with the width and height.

The **F1–Confidence curve** shows how the F1 score changes across different confidence thresholds. It reaches its highest value at a confidence of about 0.49, with an F1 score of approximately 0.86.

The **Precision–Recall curve** demonstrates high precision at medium levels of recall, with precision gradually decreasing as recall approaches 1.0. The curve results in **mAP@0.5 of about 0.90**, which directly reflects the model's overall detection quality since there is only one class.

The **Recall–Confidence curve** starts with a high recall value (about 0.96 at confidence 0). As the confidence threshold increases, recall drops because the model becomes more conservative and starts missing more objects.

Recall is especially important when the goal is to detect all extinguishers, including small or low-quality ones. Using a lower confidence threshold (between 0.3 and 0.5) helps maintain higher recall.



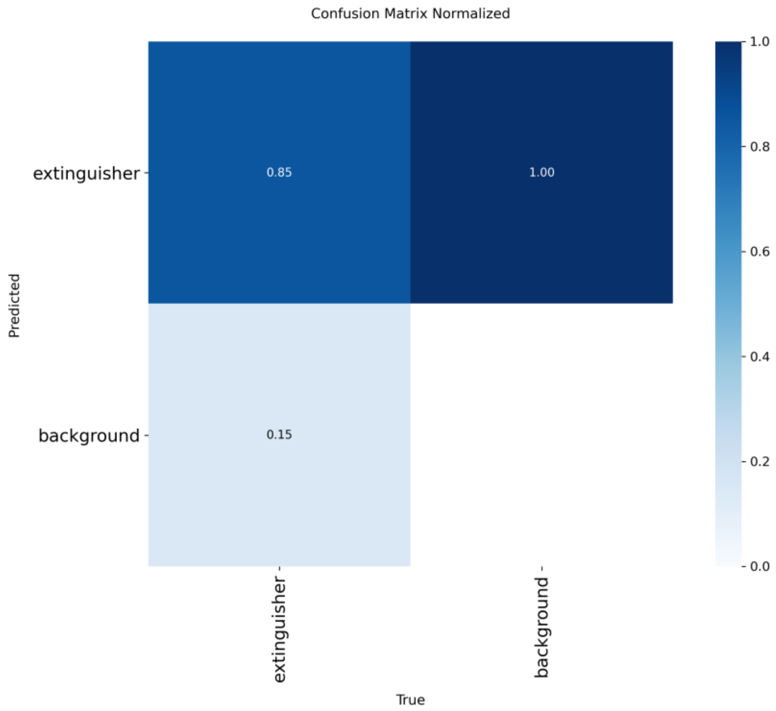
CONFUSION MATRIX

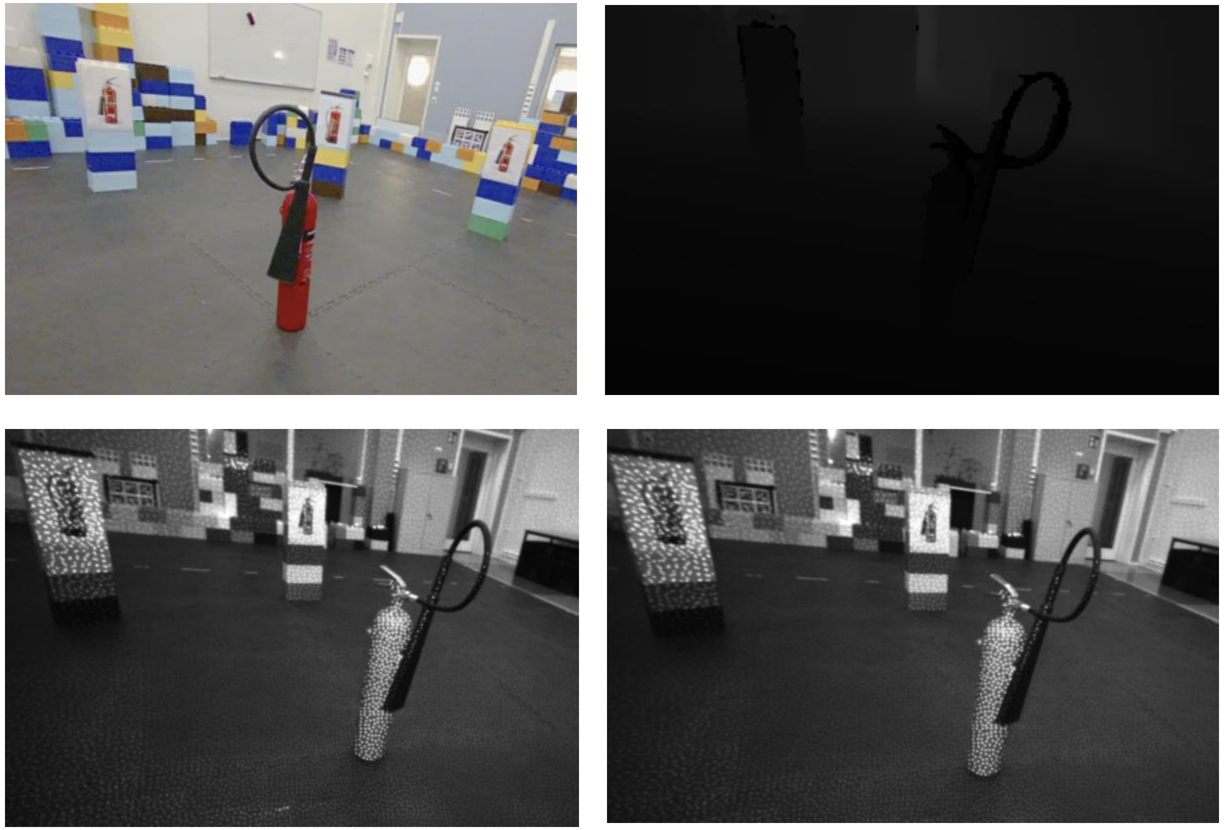
Strengths

High number of true positives (153 objects or 0.85%): the model reliably detects extinguishers.

Weaknesses

- 29 false positives: model sometimes confuses background / decoys with extinguishers.
- 26 false negatives: small or partially occluded extinguishers may be missed.





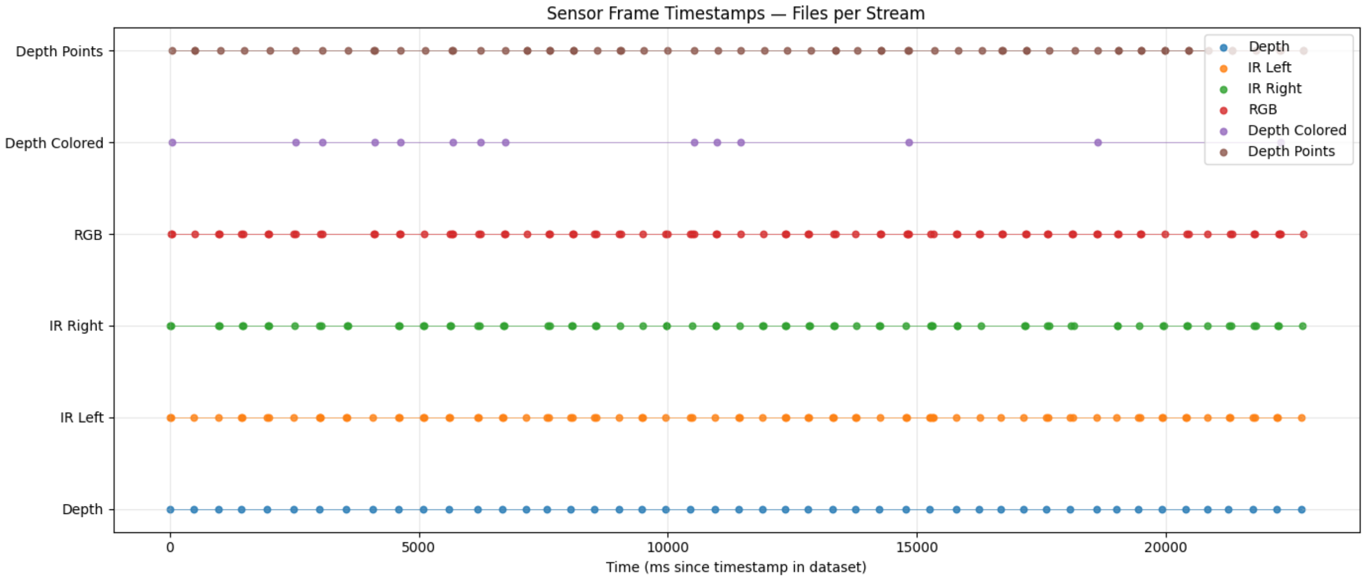
CUSTOM DATASET

The **custom dataset** represents a fully extracted dataset already separated by sensor type. The original data came from a **.bag** file, which is a raw recording from a real camera.

A .bag file stores all sensor streams internally as ROS topics. Our dataset is simply the unpacked version of this .bag file, where each stream has been exported into standard file formats such as PNG, TXT, and PCD.

Folder	File Count	Interpretation
camera_color_*	88	RGB camera, main image stream
camera_depth_*	48	Depth sensor recorded less frequently
camera_depth_points	67	Point clouds were not generated for every frame
camera_left_ir_image_raw	81	Left infrared camera recorded at its own frequency
camera_right_ir_image_raw	76	Right infrared camera recorded at its own frequency
depth_registered_colored_pointclouds	14	Very heavy 3D point cloud files, typically saved only occasionally

CUSTOM DATASET. SYNCHRONIZATION

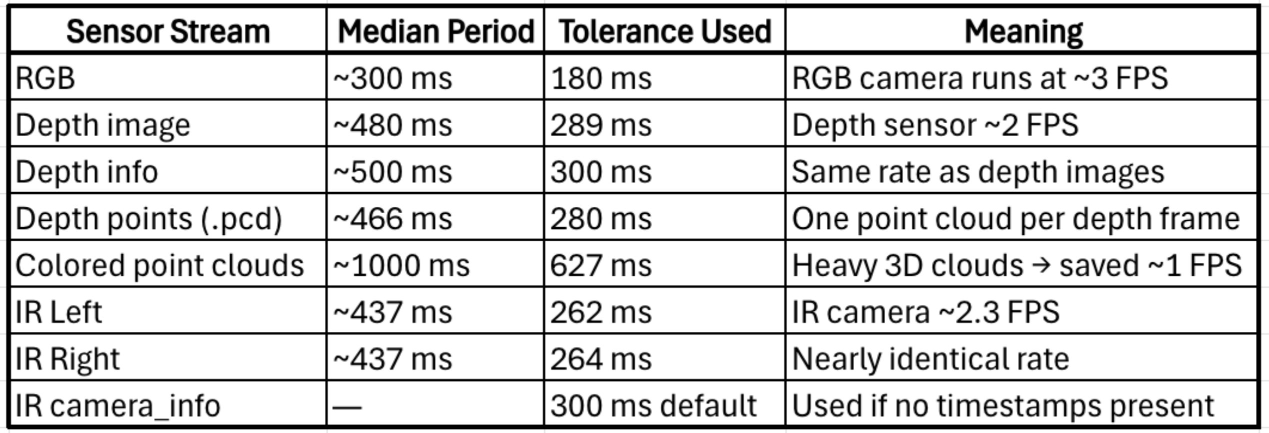


Multi-sensor setup records data from several cameras (RGB, depth, IR, point clouds), each running at a different frame rate. To match frames from different sensors to the same physical moment, the pipeline uses the **timestamp embedded in the filename**.

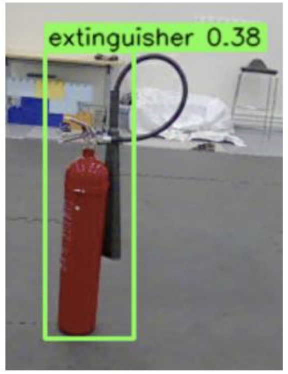
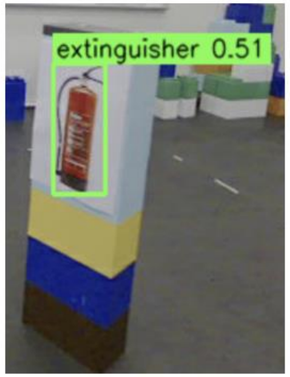
```
Example:
camera_color_info_1727164479160357265.txt
```

This long number is a **UNIX timestamp in nanoseconds from** 1 January 1970, 00:00:00 UTC. It tells exactly when the frame was captured. My pipeline converts these timestamps to milliseconds and computes time differences between frames from different sensors. If the time difference is below a sensor-specific tolerance, the frames are considered **synchronized**.

The script also measures the **median interval** between consecutive files in each folder. This reveals the effective frame rate of the sensor and helps set the tolerance window for matching frames. The paired images are stored in **rgb_sensor_pairs.csv**.



Sensor Stream	Median Period	Tolerance Used	Meaning
RGB	~300 ms	180 ms	RGB camera runs at ~3 FPS
Depth image	~480 ms	289 ms	Depth sensor ~2 FPS
Depth info	~500 ms	300 ms	Same rate as depth images
Depth points (.pcd)	~466 ms	280 ms	One point cloud per depth frame
Colored point clouds	~1000 ms	627 ms	Heavy 3D clouds → saved ~1 FPS
IR Left	~437 ms	262 ms	IR camera ~2.3 FPS
IR Right	~437 ms	264 ms	Nearly identical rate
IR camera_info	—	300 ms default	Used if no timestamps present

True Positive			False Positive
	46	26	
	9	7	
False Negative			True Negative

CUSTOM DATASET. PERFORMANCE METRICS

The model trained on the Roboflow dataset is now evaluated on the custom dataset. According to the project requirements, the performance metrics must satisfy: **True Positive Rate (Real Objects) ≥ 0.75** and **False Positive Rate (Decoys) ≤ 0.15**.

Because the custom dataset is not labelled, the performance metrics were calculated manually. The confusion matrix shows how the model detected both real extinguishers and decoy objects.

The **True Positive Rate (Real Objects)** is **0.84** (Roboflow data set achieved 0.85), which meets the project requirement. However, the **False Positive Rate (Decoys)** is **0.79**, which is far above the required threshold. This indicates the need for additional filtering steps.

$$\mathrm{TPR} = \frac{TP}{TP + FN} = \frac{46}{46 + 9} = 0.84$$

$$\mathrm{FPR} = \frac{FP}{FP + TN} = \frac{26}{26 + 7} = 0.79$$

DEPTH-BASED FILTERING

Depth images are hard to interpret visually because they are stored as **16-bit depth maps**, where each pixel represents the actual distance from the camera in millimetres. When viewed as a normal 8-bit image, this range is compressed to 0–255, making the depth map appear as a flat grey picture even though it contains detailed distance information. For each YOLO bounding box, the corresponding region of the depth image is extracted.

From all valid depth pixels inside the bbox we compute:
Median depth — typical distance of the object
Depth standard deviation — variation of depth within the box

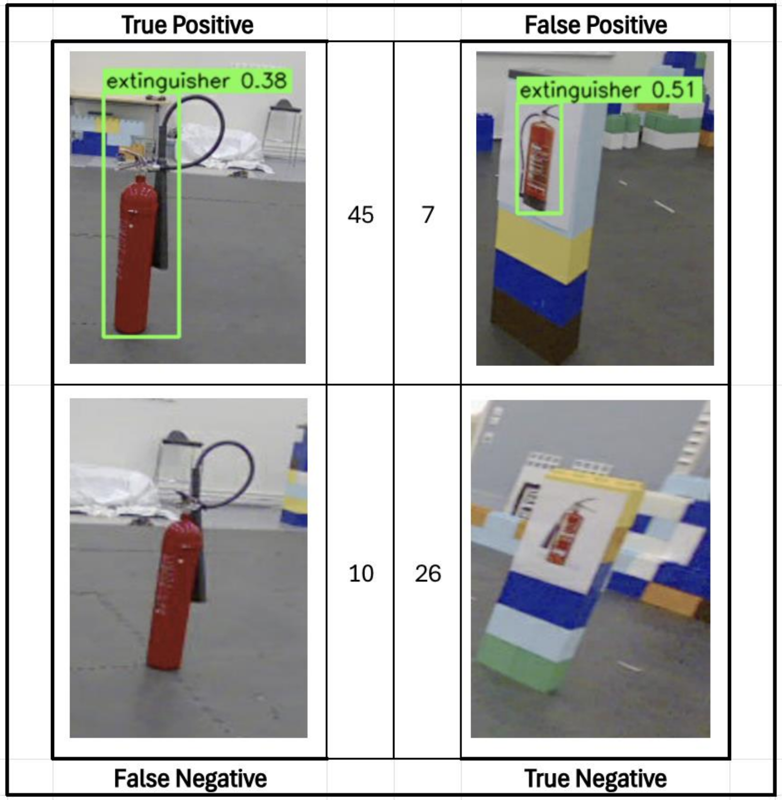
During analysis, depth standard deviation showed a consistent pattern distinguishing real extinguishers from decoys. Real 3D objects had high depth variation, while flat decoys had very low variation.

A threshold was therefore selected manually by inspecting these metrics across all samples. The final threshold was:

DEPTH_SD_THRESHOLD = 476.0
Depth std ≥ 476 → real object (keep)
Depth std < 476 → decoy (reject)

Filtered results are saved both as annotated RGB images and as a CSV file containing the computed depth statistics and the keep/reject decision for each detection.

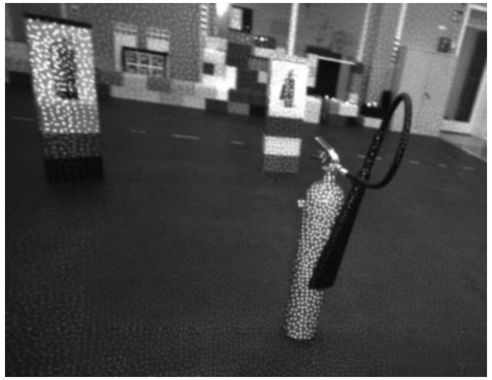
DEPTH FILTER result:
TRR = 0.81 (a slight decrease)
FPR = 0.21 (a significant improvement from 0.79, but still above the project requirement of 0.15)



True Positive			False Positive
	45	7	
	10	26	
False Negative			True Negative

extinguisher 0.38
extinguisher 0.51

INFRARED (IR) -BASED FILTERING



After the depth filter removes clearly flat decoys, the remaining bounding boxes are further checked using infrared (IR) images. For each RGB frame, a synchronized IR image is selected (left or right IR sensor, whichever is closest in time). IR-filter does not use stereo disparity, baseline distance, camera calibration, or any triangulation formulas. This stage is purely **texture-based**, not geometry-based.

The IR frame is aligned to the RGB resolution, and a slightly reduced bounding box is used to crop the IR patch, avoiding noisy borders. Before analysis, the IR patch is pre-processed: its contrast is normalized (1–99 percentile clipping), the characteristic IR dot pattern is smoothed with a Gaussian blur, and the patch is downscaled to reduce noise and speed up computation.

The core decision metric is **FFT-Entropy**, which measures the complexity of the frequency spectrum of the patch:

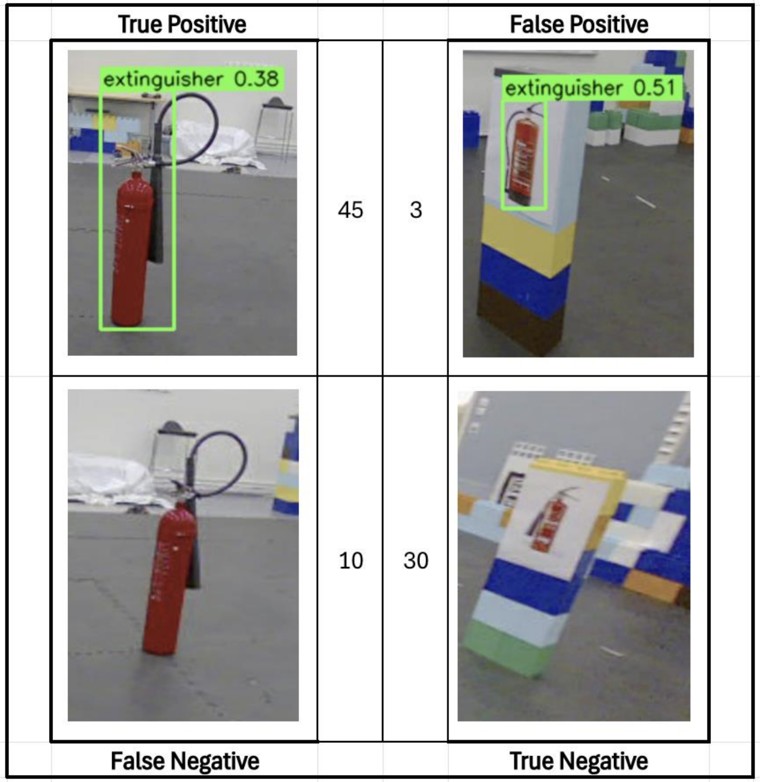
Real objects have curves, reflections, textures, shadows → many frequencies → high entropy
Flat decoys have uniform IR response → few frequencies → low entropy

A manually tuned threshold (T_FFT = 6.5) separates real objects from flat decoys:

Entropy > 6.5 → real object (kept)
Entropy ≤ 6.5 → flat decoy (rejected)

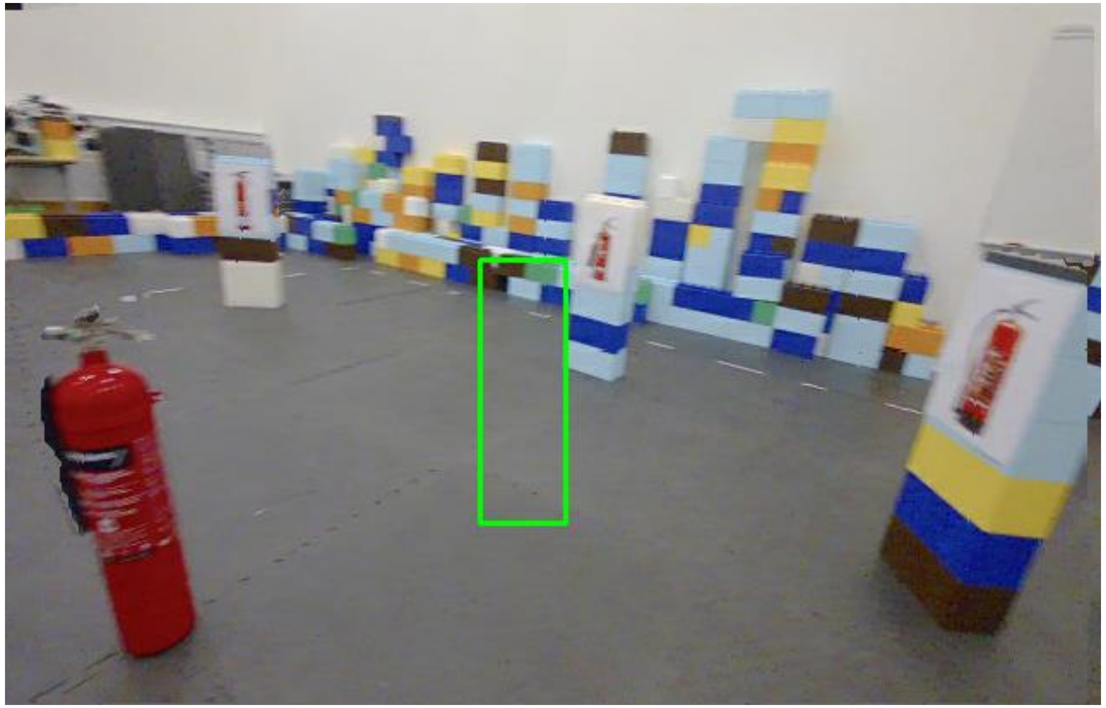
The script saves visual overlays on RGB images (green for real, red for decoy) and writes all decisions, entropy values, and IR paths into a summary CSV file.

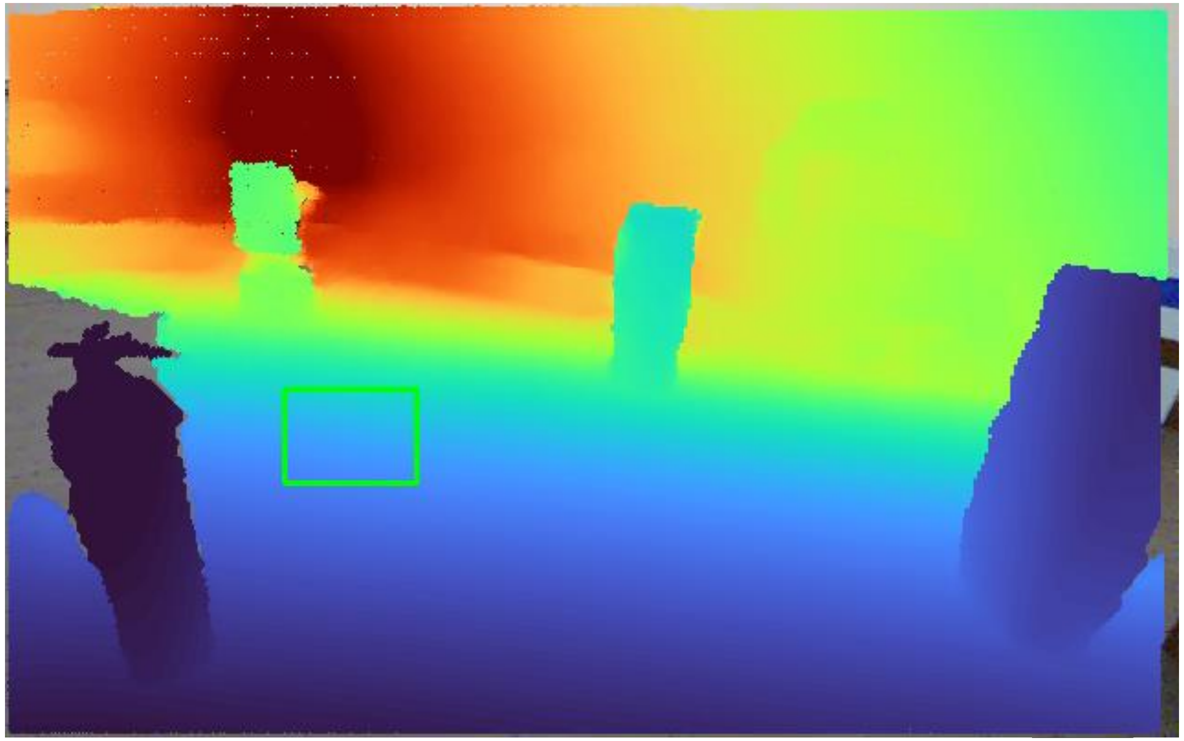
INFRARED (IR) FILTER result:
TRR = 0.81 (no difference)
FPR = 0.09 (achieved the project requirement of ≤ 0.15)



DATASET LIMITATIONS

As explained earlier, the custom dataset is limited: for 88 RGB images we only have 67 files with depth point clouds (image below) and just 14 files with colored point clouds (the left image). The first image on the left is not an original RGB frame - it is a visualization of the 3D colored points.





Because of this mismatch in available data, synchronizing point clouds with RGB images (and extracting 3D distances inside the filtered bounding boxes) leads to the results shown in the images above and on the right. That is why the localisation of some objects will have bias.

3D LOCALIZATION

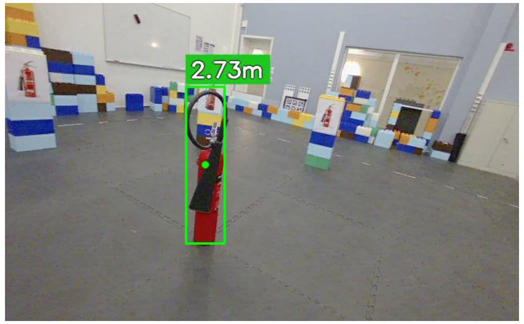
The final step is to estimate how far each object is from the camera. For every valid RGB frame, the script retrieves the synchronized **depth point cloud (.pcd)** and the camera intrinsics (fx, fy, cx, cy). All 3D points (X, Y, Z) are projected into the RGB image using these parameters, and only the points that fall within the image area are considered.

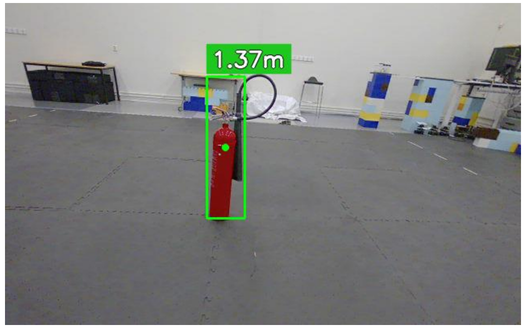
Each bounding box is **reduced by 5 pixels** to avoid borders and background. From the projected points inside this region, up to **five 3D points closest to the box center** are selected. This small, central subset provides a stable measurement: too few points make the estimate noisy, while too many introduce background and reduce accuracy. The distance to the object is computed as the median of their 3D ranges:

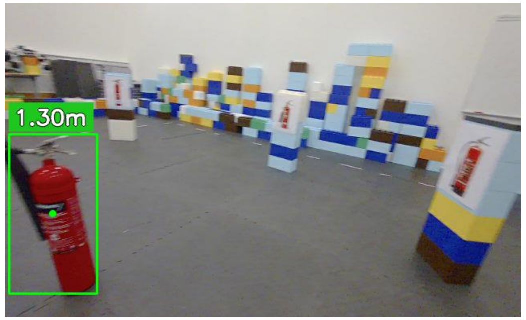
$$\text{range} = \sqrt{X^2 + Y^2 + Z^2}$$

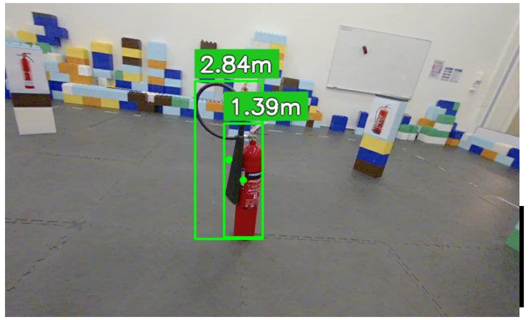
Using the median ensures robustness against noise and outliers. The final visualization overlays each RGB image with a green bounding box, a center marker, and a distance label.

We can observe that some distance estimates are biased due to the dataset limitations discussed earlier. In addition, the measured distance depends on the position of the bbox center: if the bounding box is large, its center may fall on the background rather than on the object itself, resulting in an overestimated distance.









Q&A