Implementation of a real-time detection model on a Drone

Figure 1. YOLOv8's bounding boxes surrounding the SVGS beacon using the D455 camera.

Summary

1 - Context

1.1 - Problem definition

This report provides an overview of the work I conducted as an intern at *Florida Tech*. In the context of developing a neural network for beacon recognition, the primary objective I was given was to implement an object classifier algorithm on a drone's embedded system so that it could recognize different illuminated targets in its flight environment. This document is a step-by-step tutorial assuming that **you have already installed ROS on your computer** (ROS1 noetic). If you have not already installed ROS. Follow this tutorial: https://wiki.ros.org/noetic/Installation/Ubuntu.

1.2 - Create your workspace

Run the following commands:

```
me@myComputer:~$ mkdir catkin_ws_beacon_id
```

You can name your folder differently but **catkin_ws_[project name]** is a convention.

```
me@myComputer:~$ cd catkin_ws_realsense

me@myComputer:~/catkin_ws_beacon_id$ mkdir src

me@myComputer:~/catkin_ws_beacon_id$ catkin_make
```

1.3 - Installation of the realsense-ros wrapper

In order to use the *Intel RealSense D455 Depth Camera*, you need to install the realsense-ros wrapper available here: https://github.com/IntelRealSense/realsense-ros . **Select the ros1-legacy branch** as below:

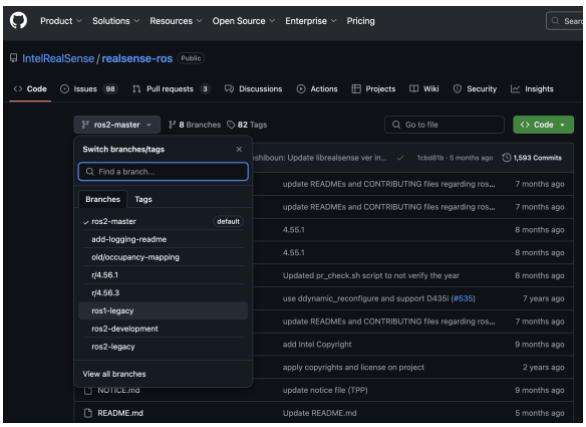


Figure 2. Select the ros1-legacy branch of the realsense-ros wrapper

Usually you are supposed to clone the repository, but as you selected another branch, download it by clicking on **Code > Download ZIP**.

With the file manager, navigate to the **/catkin_ws_beacon_id/src** folder and put the realsense-ros wrapper inside. Then, run this command:

```
me@myComputer:~ $ sudo apt-get install ros-noetic-realsense2-camera
```

You can now verify the installation by subscribing to the **camera/color/image_raw** topic in **rviz**:

```
me@myComputer:~ $  roslaunch realsense2_camera rs_camera.launch
```

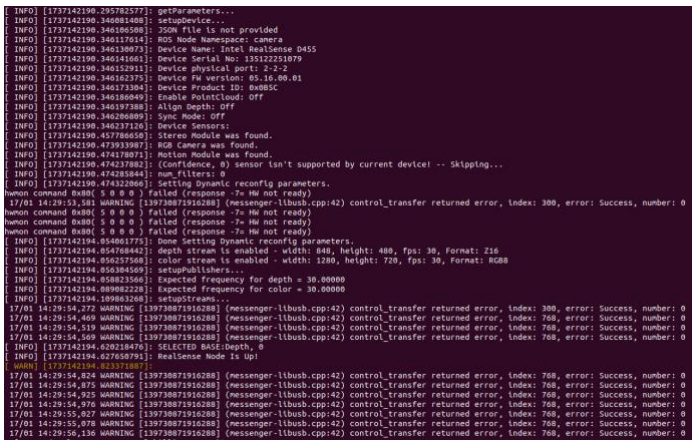


Figure 3. Normal behavior of the camera in the terminal window

You can safely ignore “control_transfer” warning messages. However, if you get an error saying “package not found”, verify that you have correctly sourced your environment during the ROS installation tutorial. If you don’t want to source automatically your environment, you can also source it for every new terminal window with this command:

```
me@myComputer:~/catkin_ws_beacon_id$ source devel/setup.bash
```

To visualize the output of the camera launch **rviz**:

```
me@myComputer:~ $ rviz
```

Click on “**add**” then “**By topic**”, finally **/camera/color/image_raw**. You should be able to see the output of the camera.

1.4 - Create a new package for beacon identification

Run this command in a terminal window:

```
me@myComputer:~/catkin_ws_beacon_id/src$ catkin_create_pkg beacon_identification cv_bridge rosbag roscpp roslib rospy sensor_msgs
```

Create all these folders:

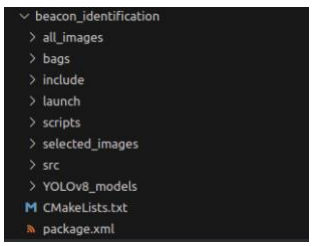


Figure 4. beacon_identification package tree

Build and source your workspace again:

```
me@myComputer:~/catkin_ws_beacon_id/src$ cd ..

me@myComputer:~/catkin_ws_beacon_id$ catkin_make

me@myComputer:~/catkin_ws_beacon_id$ source devel/setup.bash
```

2 - Data capture using ROS and the D455 camera

In order to have a streamlined process of the data capture and training of the model, I created two launch files to simplify the creation of datasets.

2.1 - Capturing images from the Intel D455 camera

You can record ROS bags at the default rate using the following launch file: /launch/**create_dataset_bag.launch**.

```
roslaunch beacon_identification create_dataset_bag.launch
```

What this does:

- It activates the *Intel D455 camera* and launches different Realsense ROS nodes publishing multiple topics including the "/camera/color/image_raw" topic, using: catkin_ws/src/realsense2_camera/launch/**rs_camera.launch**.
- It waits for 10 seconds to make sure the "/camera/color/image_raw" is indeed published and writes a ROS bag file containing its data using this python script: /scripts/**beacon_rosbag_record.py**. The result is a "**test.bag**" file saved in /bags.

2.2 - Converting ROS bags to jpeg files

You can break the ROS bag you have just created using the following launch file: /launch/**break_dataset.launch**

```
roslaunch beacon_identification break_dataset.launch
```

This simply runs the /scripts/**beacon_dataset.py** that converts ROS bags into jpeg files. The result is a set of images stored in the /all_images directory and a subset of images stored in the /selected_images directory.

The /all_images directory contains a number of frames equal to the default frame rate multiplied by the recording duration.

Note: To extract a subset of images at a 5Hz rate from a video captured at 30 frames per second, one image is selected from the full dataset every 6 frames. This parameter is used by default. You can choose a different saving directory, a different subset directory or a different step by changing the "args" parameter in the <node> tag.

Example: args="myfile.bag /path/to/saving/directory /path/to/subset/directory 30".

3 - Data capture using the VOXL2 drone

3.1 - Setting up Android Debug Bridge (ADB)

If you are using Ubuntu 20.04, copy and paste these commands in a terminal window:

```
me@myComputer:~ $ sudo apt update

me@myComputer:~ $ sudo apt install-y adb fastboot
```

3.2 - Connect the drone to your computer

This part will vary depending on the work already done on the drone (Whether ROS is installed...). Ask questions if you are not sure. Plug the drone to your computer with a USB-C cable. Then, in a terminal window type:

```
me@myComputer:~$ adb shell
```

You should see Modal AI's logo printed in the terminal.

[MODAL AI LOGO]

You should get "No current network connection". It is not necessary to enable the Wifi unless you need to install ROS.

3.3 - Verify that ROS is installed

Try to source your environment. To do that, check what type of setup file could be already installed on the drone by navigating to the **/opt/ros/[your ROS distribution (i.e. melodic, noetic, …)]**:

```
voxl2:/$ cd /opt/ros

voxl2:/opt/ros$ ls # this command will let you see if you already have a ROS distribution folder
```

If your folder is **melodic**, navigate inside to see if a setup file is installed:

```
voxl2:/opt/ros$ cd melodic
```

If there is a **setup.bash** or **setup.sh** file type:

```
voxl2:/opt/ros/melodic$ source /opt/ros/melodic/setup.bash  # or setup.sh
```

If you get the error "file not found" you need to install ROS. In this tutorial, we will be installing ROS melodic.

To install ROS (if it is not already installed) you need to enable the wifi.

3.3.1 - Enabling the Wi-fi

First, connect a router to the Ethernet switch under your desk. You must plug the cable coming from the switch to the '**Internet'** port on the router.

Then, connect the router from one of the Ethernet ports to your computer. Exit the drone and type:

```
voxl2:/$ exit

me@myComputer:~$ adb shell voxl-wifi
```

Select the '**station**' mode. Enter a custom **SSID** name and '**1234567890**' as the password. Then type:

```
me@myComputer:~$ adb reboot && adb wait-for-device
```

Wait for the drone to reboot. Now you can use the command **voxl-my-ip** to get the IP address of your device:

```
me@myComputer:~$ adb shell voxl-my-ip
```

3.4 - Installing the mpa-to-ros package

3.4.1 - Pushing the mpa-to-ros package with the terminal

Check if the **voxl-zbft-mpa-to-ros_X.X.X_arm64.deb** Debian file is already on the drone by navigating to the **/data** folder. If you find it, skip this part.

Otherwise, simply copy and paste this command in a terminal window with the correct path to locate your Debian file (This Debian file was probably sent to you via an e-mail to do this work). If you just downloaded it, it will be in your **~/Downloads** folder.

```
me@myComputer:~ $ adb push ~/Downloads/voxl-zbft-mpa-to-ros_0.1.0_arm64.deb /data/
```

3.4.1 - Pushing mpa-to-ros with VS Code (If you have Wifi)

You don't have to do this step if the previous one worked well. Otherwise, open Visual Studio Code, go to Extensions in the left panel and install the official '**Microsoft Remote – SSH**' extension. Press **Ctrl+Shift+P**. Then, search for '**Remote-SSH: Add New SSH Host…**' and type:

```
> ssh root@[your drone's IP address]
```

Select '**/home/user/.ssh/config**' as the SSH configuration file to update.

Press **Ctrl+Shift+P** again, search for '**Remote-SSH: Connect to Host…**' and click on you IP address. You should have access to the files stored inside the drone. Navigate to **/data**

You should see a **file manager panel on the left**. Drag the Debian file (**voxl-zbft-mpa-to-ros_0.1.0_arm64.deb**) into the **/data** folder.

3.4.2 - Installing mpa-to-ros

Connect to the voxl2 device:

```
me@myComputer:~ $ adb shell

voxl2:/$ apt install ./voxl-zbft-mpa-to-ros_0.1.0_arm64.deb
```

3.5 - Recording ROS bags

Datasets are often very large. For practical reasons, open 4 different terminal windows (let's call them A, B, C, D).

In **terminal window A**:

```
me@myComputer:~ $ adb shell

voxl2:/$ roslaunch voxl_mpa_to_ros_mod voxl_mpa_to_ros.launch
```

You should see the topic of interest being published: "**Found new interface: hires_small_color**".

Once you are ready to record a ROS bag, in **terminal window B** type:

```
me@myComputer:~ $ adb shell

voxl2:/$ cd /data

voxl2:/data$ rosbag record-o [name of the bag].bag /hires_small_color
```

Press Ctrl+C to stop recording. Then, in **terminal window C**:

```
me@myComputer:~$ sudo adb pull /data/[name of the bag].bag [destination path]
```

Then, open the **bagbreaker.launch** file (this launch file was probably sent to you via an e-mail to do this work). Edit the filename_format and sec_per_frame with custom values. Setting the file name to a unique identifier containing the color configuration, label of the beacon, background setup is recommended.

Examples:

YYBY_7-plain_desk.jpg YYBY_7-blue_folder_desk.jpg YYBY_7-ground.jpg

YYBB_10-reflections.jpg YYBB_10-hand.jpg

```
<arg name="filename_format" value="[destination path]/[custom name]%04d.jpg"/>

<arg name="sec_per_frame" value="0.25"/>          # value of your choice 0.25 -> 4 images per second
```

In **Terminal window D :**

```
me@myComputer:~$ roslaunch [path to file]/bagbreaker.launch bag_file:=[path to your bag file].bag
```

Your pictures will be located in your **[destination path]** folder.

4 - The dataset

4.1 - Content of the dataset

I captured several ROS bag recordings from various angles and distances relative to the beacon, adjusting to a random LED intensity in each. Additionally, I included numerous background images and photos of light reflections on objects like my desk and monitors. I also incorporated images where the beacon was off. Nearly all the bags were recorded while moving around in the Lab to create motion blur images to better reproduce flight conditions.

4.1.1 - The training set

Usually the training set contains images with lots of variations (processing, light changes, angles distances, backgrounds …). However, the laboratory I work in lacks diversity. To simulate variation in the training set, I artificially adjusted the white balance of the camera under my phone's flashlight, changed the color of the beacon's stand, and added ducks…

4.1.2 - The validation and test set

However the validation and test set are more "reasonable" and need to incorporate images with less processing, matching with the real flight conditions. The test set generally only includes images of the actual deployment environment but my limitations are preventing me from fully exposing the model to the real-world it will encounter when deployed on the drone.

Figure 5. Photos extracted from a validation batch of the green dataset

4.2 - Image labelling

Roboflow is really practical when it comes to labeling images and it is the only tool I used to label datasets. I firstly drew tight bounding boxes around the beacon LEDs but was asked to add more space around them to improve the box_loss parameter. (More of this in the Results).

4.2.1 - Setting rules for effective labelling

As you can imagine, adding space between the object of interest and the bounding boxes is an approximate process that can introduce variability in the training and decrease the precision of the model. To reduce variability in labeling, it might be helpful to establish some guidelines.

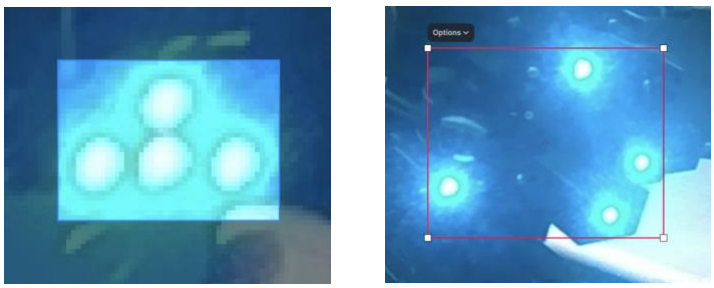


Figure 6. Considered good labeling: Saturated light spots are given some space, while the blue hue is closely enclosed.

Figure 7. Considered good labelling: We can see 4 different stripes of light. The thinner dimmer stripes are ignored.

Figure 8. Images must be excluded from the dataset. Either a LED is missing or some of them are merging

5 - Training of the model

5.1 - YOLO (You Only Look Once)

I chose YOLO due to its performance in real-time applications. It has one of the best mean Average Precision (mAP) to FPS ratios (see figure 7), and it's also very user-friendly, with guided usage and easy implementation since it's based in Python. I opted not to use the latest version but the eighth version of YOLO to ensure that sufficient documentation would be available.

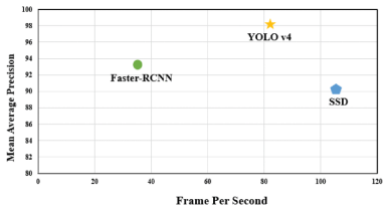


Figure 9. J. -a. Kim, J. -Y. Sung and S. -h. Park, "Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition," *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, Seoul, Korea (South), 2020, pp. 1-4, doi: 10.1109/ICCE-Asia49877.2020.9277040.

5.2 - Google Colab Notebooks

Google Colab offers great Graphics Processing Units for free so I used their Notebooks with multiple e-mail accounts to train my models. When the dataset only had green beacons, each training session took me around one hour every time I decided to change a parameter making it very time-consuming. (See Hyper-parameters for more information). Once the dataset was completed with red and blue beacons it took me 3 hours. Therefore, the **fine-tuning of the model was done using only green**.

5.2.1 - Download the dataset on a Google Drive using Colab

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="rH0MT2pv2i4qVVExOfQL")
project = rf.workspace("louis-0n46j").project("beacon-id-d455")
version = project.version(10) Dataset version: 10
dataset = version.download("yolov8")

Collecting roboflow
  Downloading roboflow-1.1.49-py3-none-any.whl.metadata (9.7 kB)
```

Figure 10. Code to download the beacon dataset on your drive

5.2.2 - Training

Create a new Notebook inside your dataset to train the model. Click on **Runtime > Change runtime type**. Select **T4 GPU**. This will enable faster trainings.

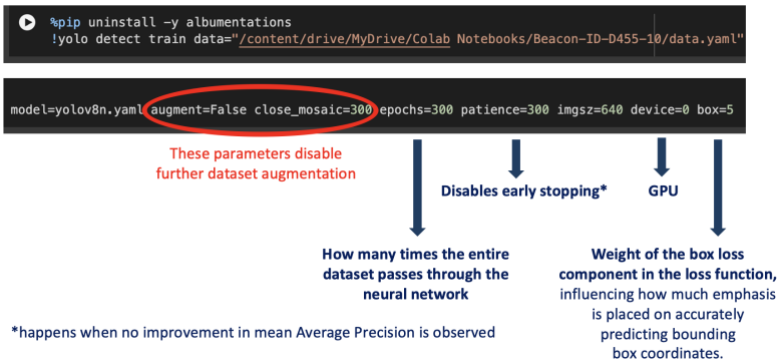


Figure 11. Training parameters that gave satisfactory results.

5.3 - Processing images with Albumentations

By default, *YOLOv8* applies image processing with the *Albumentations* module, which is pre-installed on *Colab*. Initially, I used these parameters for image processing, but I was later instructed to limit processing to rotations and zoom only. Consequently, I disabled *Albumentations* for my training sessions.

5.4 - Expectations for the model

The goal is to achieve a box loss of less than 50% and a mean Average Precision (mAP50-95) of 90%.

5.5 - Hyper-parameters

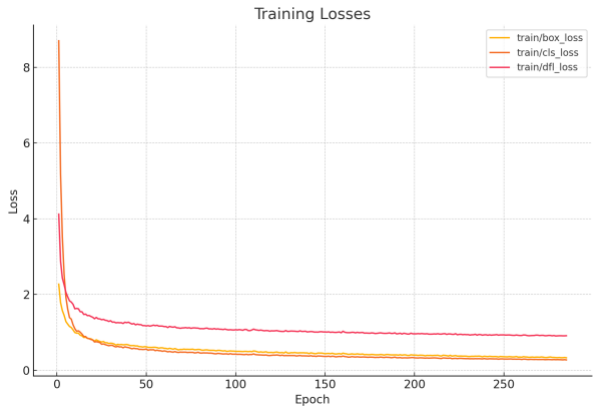
Dataset	Train	data	model	epochs	patience	batch	imgsz	close_mosaic	augment	lr0	lrf	box	box_loss0	box_lossf	status
Beacon-ID-D455-5	1	data.yaml	yolov8n.yaml	300	50	16	640	10	TRUE	0.01	0.01	12	6.3329		interrupted
Beacon-ID-D455-5	2	data.yaml	yolov8n.yaml	30	50	16	640	10	TRUE	0.01	0.01	7.5	2.0434		interrupted
Beacon-ID-D455-5	3	data.yaml	yolov8n.yaml	300	50	16	640	10	TRUE	0.01	0.001	7.5	4.0016		interrupted
Beacon-ID-D455-5	4	data.yaml	yolov8n.yaml	300	50	16	640	10	TRUE	0.01	0.01	12	6.3329		interrupted
Beacon-ID-D455-5	5	data.yaml	yolov8n.yaml	30	50	16	640	10	TRUE	0.01	0.01	7.5	4.0016		interrupted
Beacon-ID-D455-5	6	data.yaml	yolov8n.yaml	30	50	16	640	10	TRUE	0.01	0.01	10	5.3198		interrupted
Beacon-ID-D455-5	7	data.yaml	yolov8n.yaml	30	50	16	640	10	TRUE	0.01	0.01	8	4.2729		interrupted
Beacon-ID-D455-5	8	data.yaml	yolov8n.yaml	30	50	16	640	10	TRUE	0.01	0.01	7	3.7322		interrupted
Beacon-ID-D455-5	9	data.yaml	yolov8n.yaml	30	50	16	640	10	TRUE	0.01	0.01	6.5	3.7322		interrupted
Beacon-ID-D455-5	10	data.yaml	yolov8n.yaml	300	50	16	640	10	TRUE	0.01	0.01	6.5	3.466		interrupted
Beacon-ID-D455-5	11	data.yaml	yolov8n.yaml	300	50	16	640	10	TRUE	0.01	0.01	6.5	3.466	0.65	interrupted
Beacon-ID-D455-5	12	data.yaml	yolov8n.yaml	300	300	16	640	10	TRUE	0.01	0.01	5	2.6764	0.3624	finished
Beacon-ID-D455-6	1	data.yaml	yolov8n.yaml	300	300	16	640	50	TRUE	0.01	0.01	5	2.536	0.364	finished
Beacon-ID-D455-6	1	data.yaml	yolov8n.yaml	300	300	16	640	10	TRUE	0.01	0.001	5	2.536	0.376	finished
Beacon-ID-D455-6	2	data.yaml	yolov8s.yaml	300	300	16	640	10	TRUE	0.01	0.01	5	2.5786	0.31463	finished
Beacon-ID-D455-6	3	data.yaml	yolov8m.yaml	300	300	16	640	10	TRUE	0.01	0.01	5	2.569	0.2921	finished
Beacon-ID-D455-4	3	data.yaml	yolov8m.yaml	300	300	16	640	10	TRUE	0.01	0.01	5	2.6522	0.3046	finished
Beacon-ID-D455-7	1	data.yaml	yolov8n.yaml	300	300	16	640	10	TRUE	0.01	0.01	5	2.6283		interrupted
Beacon-ID-D455-7	2	data.yaml	yolov8n.yaml	300	300	16	640	10	FALSE	0.01	0.01	5	2.6283		interrupted
Beacon-ID-D455-7	3	data.yaml	yolov8n.yaml	300	300	16	640	300	FALSE	0.01	0.01	5	2.331	0.32925	finished
Beacon-ID-D455-8	1	data.yaml	yolov8n.yaml	300	300	16	640	300	FALSE	0.01	0.01	5	2.429	0.33377	finished
Beacon-ID-D455-10	1	data.yaml	yolov8n.yaml	300	300	16	640	300	FALSE	0.01	0.01	5	2.271	0.32771	finished

Figure 12. Trainings I have kept records of.

I mainly adjusted the following parameters:

- lrf (final learning rate): Final learning rate as a fraction of the initial rate = (lr0 * lrf), used in conjunction with schedulers to adjust the learning rate over time.
- box: Weight of the box loss component in the loss function, influencing how much emphasis is placed on accurately predicting bounding box coordinates.

6 - Results



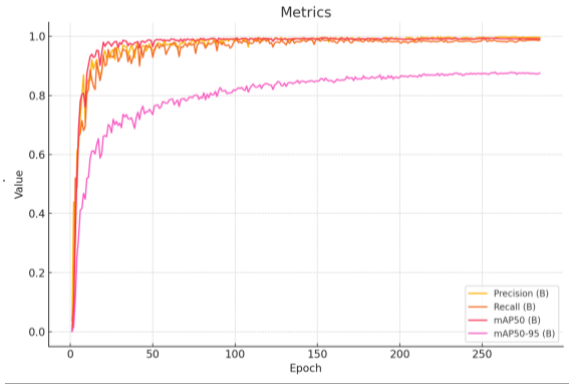


Figure 13. (left) Training with RGB beacon dataset. Box loss achieved: 0.32771

Figure 14. (right) Evolution of metrics during the training. mAP50-95 was 0.87581 while precision and recall were excellent.

6.1 - Detection performance

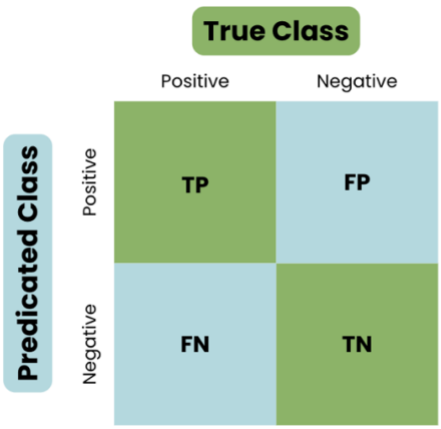
6.1.1 - Understanding the metrics

Precision quantifies the proportion of true positives among all positive predictions, assessing the model's capability to avoid false positives. On the other hand, **Recall** calculates the proportion of true positives among all actual positives, measuring the model's ability to detect all instances of a class.

Note : In our case scenario, precision might be a more significant metric than recall because it is crucial that the drone doesn't recognize a beacon when there is none (false positive).

6.1.2 - Confusion matrix

In order to better understand these metrics, we can use a confusion matrix (see figure 11). A confusion matrix is a table used to evaluate the performance of a classification model by displaying the actual versus predicted classifications. A perfect confusion matrix is an identity matrix (only True predictions).



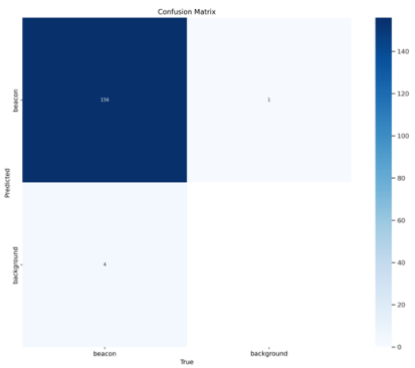


Figure 15. Explanatory infographic (left). Actual confusion matrix after training on the green dataset (right).

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Figure 12 shows that the trained model is highly accurate, with 97%* of predictions being true. Its recall was 0.98*, and its precision was 0.99* which indicates that approximately 2% of the time, the model missed detecting a beacon when one was present in the image (FN), and 1% of the time it detected a beacon when none was present (FP).

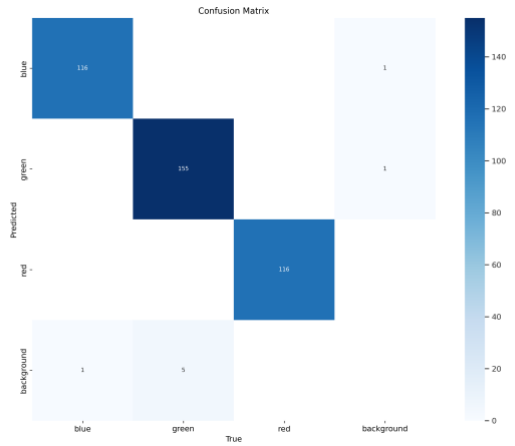


Figure 16. Confusion matrix of the model trained on the RGB dataset.

After training on the RGB dataset, our model proved to be highly accurate, with a total accuracy of 98%*. Its recall was 99%* and its precision was 98%* which indicates that approximately 1% of the time, the model missed detecting a beacon when one was present in the image (FN), and 2% of the time it detected a beacon when none was present (FP). **However, no misclassification has been observed.**

Details :

R accuracy = 100%*	R precision = 100%*	R recall = 100%*
B accuracy = 98%*	B precision = 99%*	B recall = 99%*
G accuracy = 96%*	G precision = 99%*	G recall = 97%*

*Based on the test set

6.2 - Localization performance

The previous results show that our model can detect beacons with a relatively good accuracy. However, the mean Average Precision is our most important metric as it takes into consideration the **Intersection over Union** (IoU) of the model which is essential when **precise object location** is crucial.

6.2.1 - Intersection over Union (IoU)

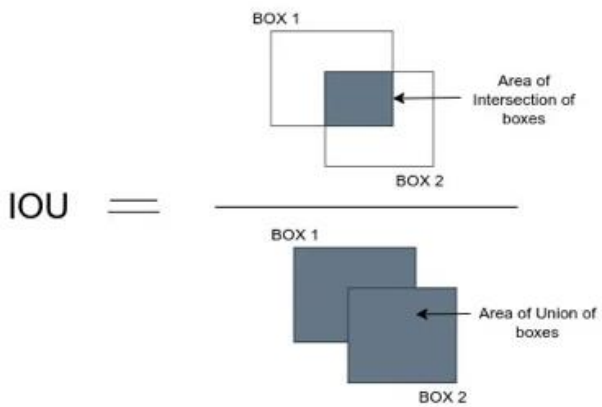


Figure 17. How to calculate Intersection over Union (IoU)

The IoU (Intersection over Union) is calculated by dividing the area of overlap between the actual (ground-truth) boxes and the predicted boxes by the area of their union. While IoU itself isn't directly used as a metric for fine-tuning the model, it's important to note that this value is computed each time a bounding box is predicted.

6.2.2 - Average Precision

Average Precision (AP) computes the area under the precision-recall curve, providing a single value that encapsulates the model's precision and recall performance. The figure 14 shows the average precision scores for different IOU thresholds. Choosing a low IOU threshold would mean we don't care about the exact amount of space the beacon occupies in the image whereas choosing a high IOU threshold ensures a better localization.

Note : mean Average Precision (mAP) extends the concept of AP by calculating the average AP values across multiple object classes. This is useful in multi-class object detection scenarios to provide a comprehensive evaluation of the model's performance. For the moment, there is only one class to detect so the notions of Average Precision and mean Average Precision are equal.

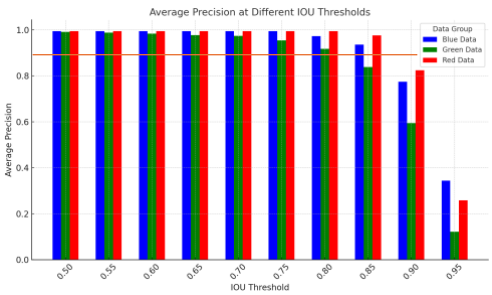


Figure 18. Average Precision scores at different IOU thresholds (0.50-0.95). Red line = mAP(50-95)

We can see that for an IOU threshold of around 0.80 we achieve the required average precision of around 90% for all the beacons. However when we calculate the average value of AP across all IOU threshold between 0.5 and 0.95 (see red line on figure 14) we get a mAP(50-95) equal to 0.87581 which is under the expectations.

6.3 - Conclusion

In this experimental context, no significant difference was observed when the final learning rate was reduced by a factor of ten. The only hyperparameter that seemed to impact the box loss metric was the "box" hyperparameter (This parameter is counterintuitive because reducing its weight, meaning the importance of minimizing box loss in model training, actually decreased the box loss). The model's **detection performance** is relatively strong, with a 98% accuracy. While this seems quite high to me, I'm not entirely sure about the expected benchmark. Concerning the model's **localization performance**, Unfortunately I did not achieve the required mAP(50-95) of 90% but only 87.6% .

6.4 - Critical analysis

My understanding of the situation is that the objects we aim to detect are light spots that vary in shape and size. Additionally, the Intel D455 camera has certain imperfections, occasionally causing astigmatism in the images (see Figure 15), which introduces additional variability to the dataset. To improve the model's ability to localize the beacon in the image, we need to clearly define what constitutes the beacon. For instance, if we include these stripes of light, should the bounding box encompass them entirely? (I think it should not). If we add space between the box edges and the light spots, it may be beneficial to quantify this spacing to reduce variability in labeling.

Figure 19. Astigmatism with Intel's D455 camera

The Ultraltyics Documentation gives us tools to fine tune their models. Ultralytics uses genetic algorithms, primarily mutation, to explore the hyperparameter space, aiming to improve metrics such as accuracy, precision, and recall. Unfortunately these processes are extremely time-consuming and can take up to hundreds or even thousands of hours with a "regular" computer.

We also could consider using larger models or different versions of YOLO as it could increase the performance of the mean Average Precision metric (see figure 16).

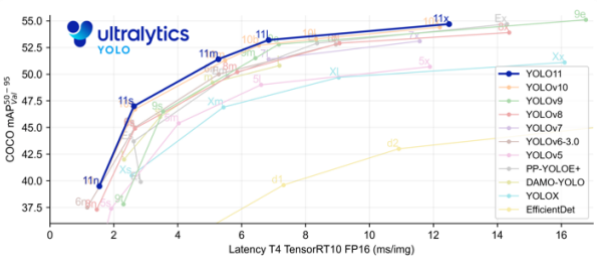


Figure 20. Comparison of different model sizes and versions of YOLO

7 - Implementation of the classifier with ROS

7.1 - Signaler ROS node

You can stream the real-time results of the model's predictions by executing the following launch file: /launch/**yolo_stream_rgb.launch**

Make sure you wait for at least 10 seconds before adding the topic in rviz. Wait for Python version warning (You can safely ignore this message)

```
roslaunch beacon_identification yolo_stream_rgb.launch
```

This launch file does 3 things:

- Running the /src/**ultralytics_rgb.cpp** node
- Launching the Intel D455 camera
- Starting Rviz

The /src/**ultralytics_rgb.cpp** ROS node basically converts an OpenCV image into a python compatible frame (or table), does the object detection with a default model (beacon_id-v10-vmbeaconid5-train1-1552.pt) located in the **YOLOv8_models** folder and returns an annotated image with the bounding boxes (I used pybind11 as a C++ python-wrapper to call the beacon-trained YOLOv8 model). This nodes depends on this python script: /scripts/**beacon_detection_rgb.py** . Inside the script, the "image_callback" function publishes the annotated image to a ROS topic that can be visualized using rviz.

Note: You can use your own model by adding new weights in the /YOLOv8_models folder and by changing the "weights" argument in the **yolo_stream_rgb.launch** file.

7.2 - Beacon detection on Rviz

Once you have waited around 10 seconds (it can be less when launching for the second time), you can add the /ultralytics/detection/image topic in Rviz. You should obtain figure 17 and 18.

Figure 21.Resulting image window in Rviz when the "/ultralytics/detection/image" is added.

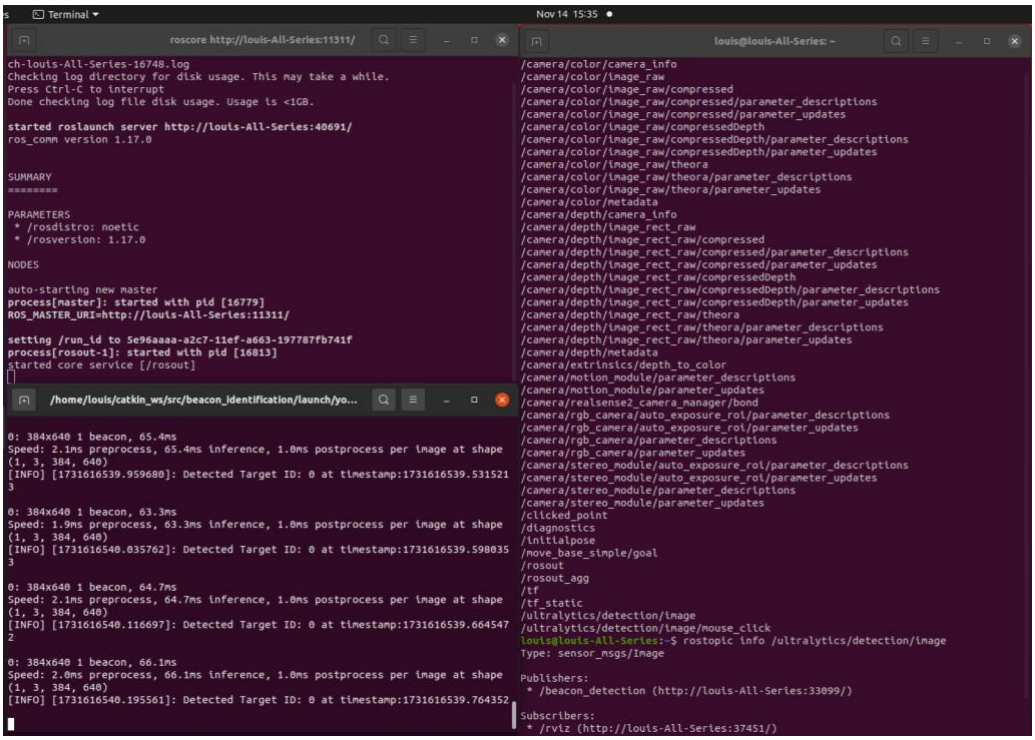


Figure 22. "Detected target" message in the terminal (bottom left corner).

7.3 - Handling color encoding and array size issues when dealing with multiple colors

When the dataset consisted solely of green beacons, the training, validation, and ROS implementation of the model appeared to work perfectly. However, after introducing blue and red beacons, the model's performance significantly declined and frequently misclassified the beacons. Therefore, because I suspected that the ROS processes of publishing and subscribing might have been interfering with the model's detections, I checked if the model worked without ROS. I used a python script that I found online to capture real-time detection with the D455 camera and the detection was good … I finally found out the error came from the color encoding that was somehow different when using YOLO on ROS although I used the official Ultralytics Documentation to design my code.

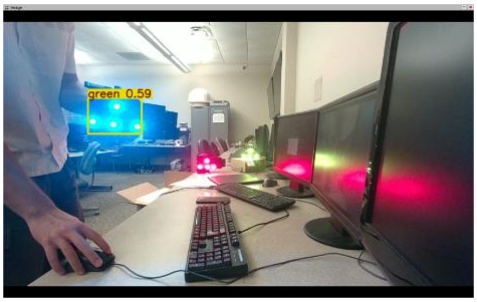
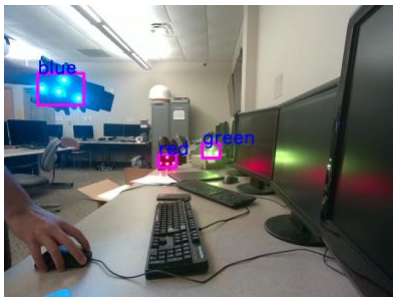


Figure 23. My node with ROS (left).



Python script found online used without ROS (right).

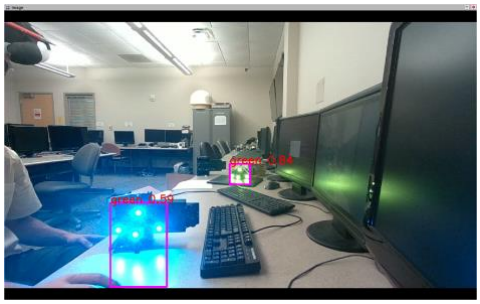


Figure 24. Python script found online with ROS (left).

My node with ROS and different encoding (right).

I must admit that I don't know exactly at what point did the encoding change but to address this issue, I had to examine the numpy frames the model was processing and I concluded that two key factors had to be considered to ensure the model performs consistently with its validation phase behavior: the **color encoding** but also **image resolution** provided to YOLO must match those of the images used in the dataset.

Roboflow uses 640p images by default so the data coming from the realsense D455 camera must be converted to this format. I used the **cv2.resize()** function.

To check the encoding type, you can use the command **rostopic echo /camera/color/image_raw/encoding** while the camera is running. The expected output is "rgb8". There are various methods to convert an image from rgb8 to bgr8, but you can easily change the encoding with a simple NumPy array manipulation:

```
array_bgr = array_rgb[:, :, ::-1]
```

8 - Upscaling the model to 14 beacons

Upscaling the model to a larger number of classes required using a real methodology to record the ROS bags. Around 10 different backgrounds were chosen (beacon on the desk, on the ground …) and reproduced for each beacon color configuration.

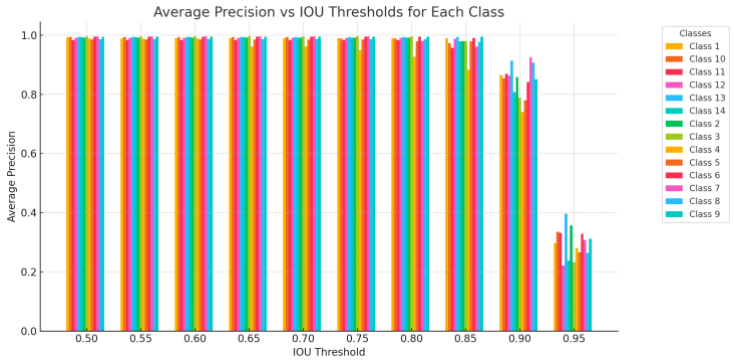


Figure 25. Average Precision at different IOU thresholds

The results of the model seem relatively good as we achieved an IOU of 80% with all the classes having an Average Precision above or equal to 90%.

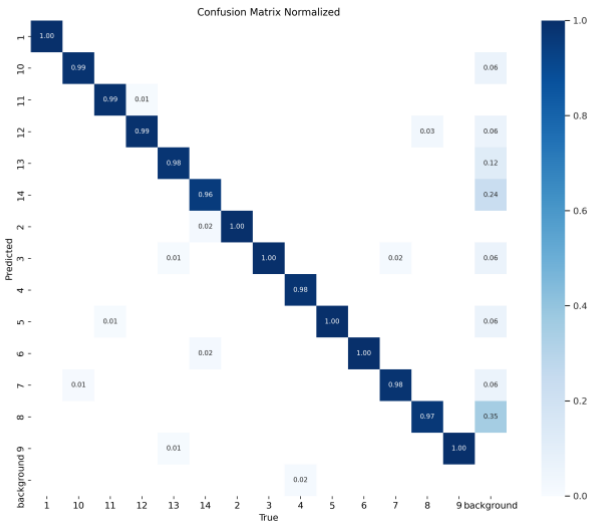


Figure 26. Normalized confusion matrix

The confusion Matrix is close to a perfect diagonal matrix apart from the last column which implies a few False Positives (FP). This means that the model detected beacons when there were in fact none in the image. This phenomenon can be observed especially for beacon 8 and 14 which had poorer datasets.r