Perception: RoboSub 2021 Online[1]

You Only Look Once (YOLO) is a real time object detection system that runs with darknet, a neural network framework. Training it requires five general steps: install and compile Darknet, make the dataset, data augmentation, labelling, and training. Before you start be sure to meet the following requirements, if not, then check out the VantTec/Educación/Manuales google drive directory[2].

Neural network **training and testing** require[3]:

- Opencv 3.3.0 +
- Cuda 10.0
- Darknet and YOLO
- Ubuntu linux

I. Install and Compile Darknet.

- **Setup**

```
cd
git clone https://github.com/AlexeyAB/darknet.git
cd darknet
```

Edit the Makefile and set the next parameters**:

```
GPU=1
CUDNN=1
CUDNN_HALF=0
OPENCV=1
```

```
make clean
make
```

**In case that you miss one or more parameters set to 0 instead of 1, otherwise you´ll have errors. However you might require OpenCv.

[1] Modified from the original guide shown by Roberto Mendivil on RoboBoat2019 drive: https://drive.google.com/drive/folders/18DM4OMHfxZX1kTEFDmbo4e3ZBjo2qded?usp=sharing

[2] https://drive.google.com/drive/folders/18ZerJy0lI0UPwMnxK0R5R-YBP7Q9V5ZU?usp=sharing

[3] You may test without gpu and cuda, but training without them is not really not an option.

- **Download YOLO Tiny 3 pre trained weights.**

```
cd
cd darknet
wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

- **Testing Darknet.**

```
./darknet detector test ./cfg/coco.data ./cfg/yolov3-tiny.cfg yolov3-tiny.weights data/giraffe.jpg
```

The output is shown below:

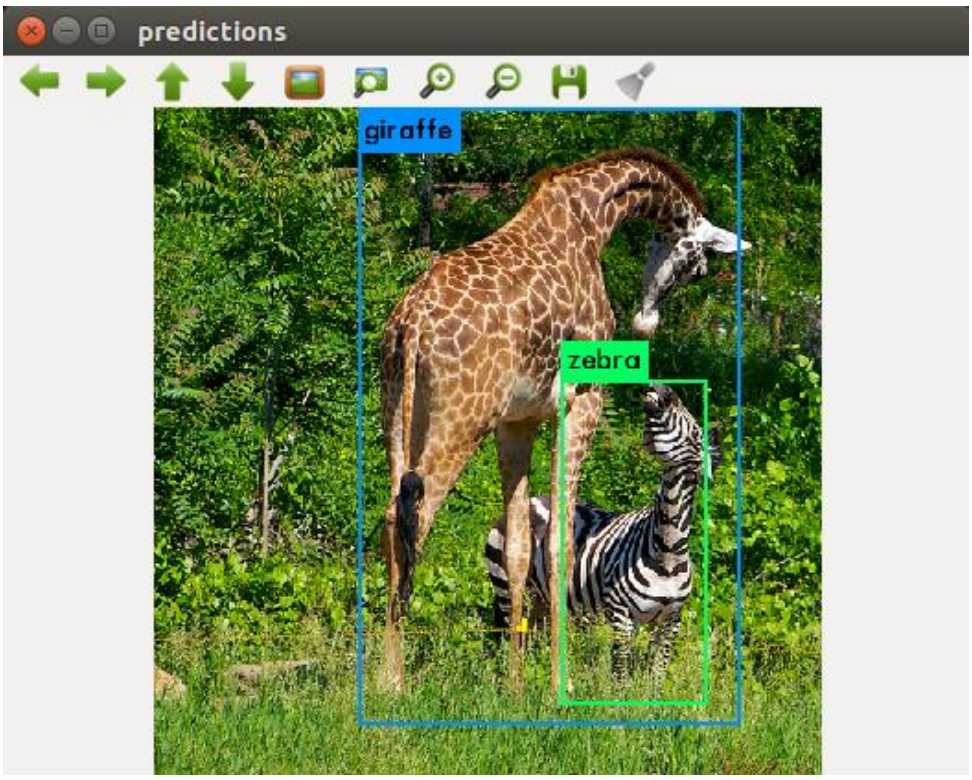


Figure 1: Expected output

- **Download Robosub2021_sim directory.**

Download and extract Robosub2021_sim directory on your Darknet directory from: https://drive.google.com/drive/folders/1j1Yz0uXWgtZsewnat-fUnY6f5FQB4tGx?usp=sharing

The tree directory must look like this:

- darknet
    - Robosub2021_sim

This directory has everything needed for Darknet and YOLO, in fact Robosub2021_sim is the final product of the entire project.

- **Testing Darknet on Robosub2021_sim.**

Edit darknet/Robosub2021_sim/cgf/config.cgf

[net]
# Testing
batch=1 **<- Uncomment this line**
subdivisions=1 **<- Uncomment this line**
# Training
#batch=64 **<- Comment this line**
#subdivisions=8 **<- Comment this line**

cd
cd darknet

./darknet detector test Robosub2021_sim/cfg/obj.data Robosub2021_sim/cfg/config.cfg Robosub2021_sim/weights/config_best.weights Robosub2021_sim/dataset/val/gun_98.jpg

Expected Output

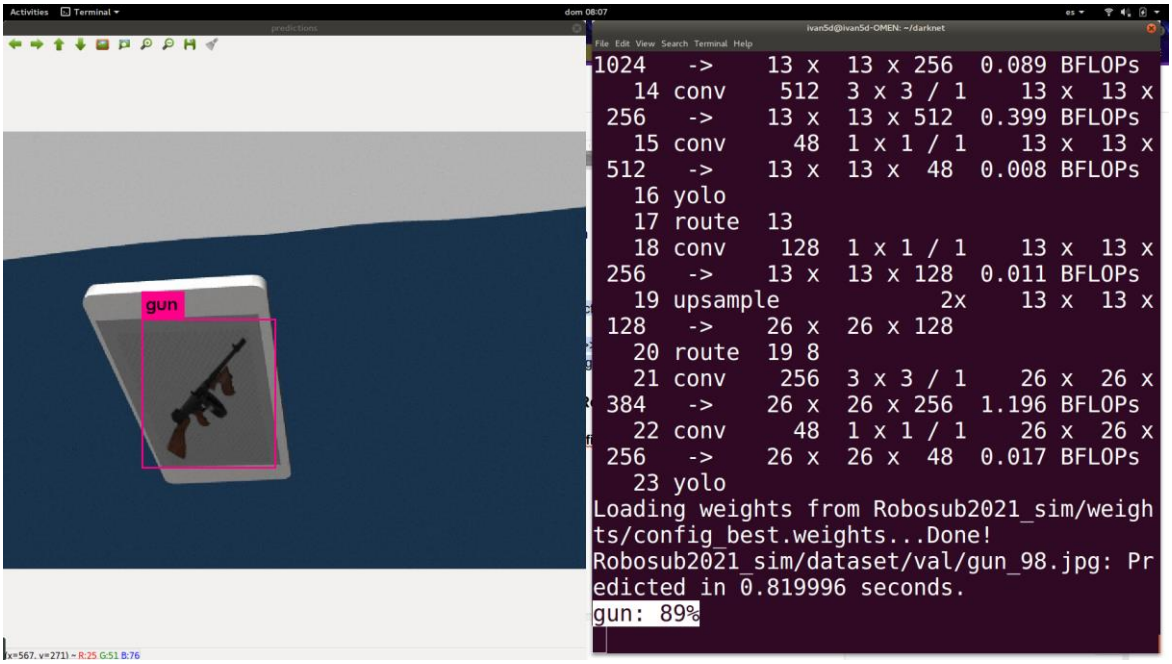


Figure 2: Expected Output with Robosub2021_sim

For testing with other weight or image:

./darknet detector test Robosub2021_sim/cfg/obj.data Robosub2021_sim/cfg/config.cfg Robosub2021_sim/weights/<<file.weights>> Robosub2021_sim/dataset/val/<<image.jpg or png>>

II. Dataset.

YOLO is used for object detection, so an image dataset of the objects is needed to tell the neural network how the objects look like. For the Robosub2021 online edition the dataset was gathered from gazebo, the simulator; thus, a teleop script for handling the virtual VantTec Submarine and taking the pictures with the keyboard was made[4]. Please notice that there is a difference between the classes names and the images taken.

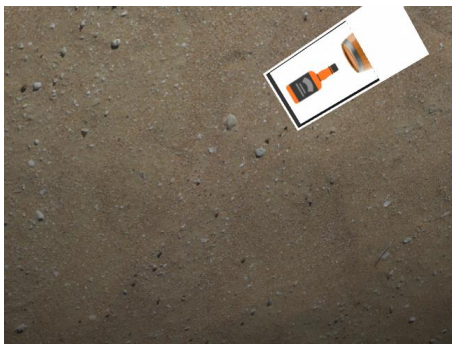
[4] This teleop node (script) is found on the /feature/marker branch of the vanttec/vanttec_uv_sim workspace: https://github.com/vanttec/vanttec_uv_sim.git.

```
police
gangster
gate
gun
badge
marker
marker_2
bin1
bin2
dolar
axe
bottle
octagon
gman
bootlegger
```

Robosub 2021 classes

We identified 15 object classes needed for the competition, but, as you may see, there are not 15 different group types of images, just 10 shown below.

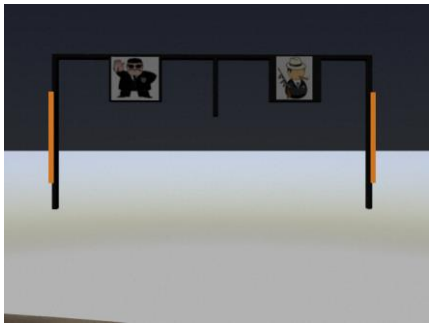
Badge



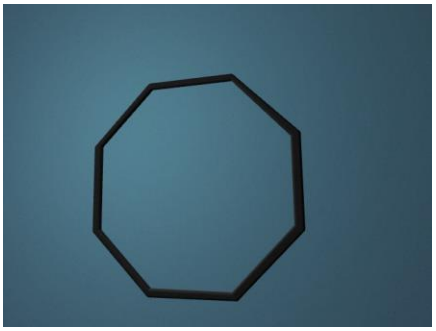
Bin 1

Gun

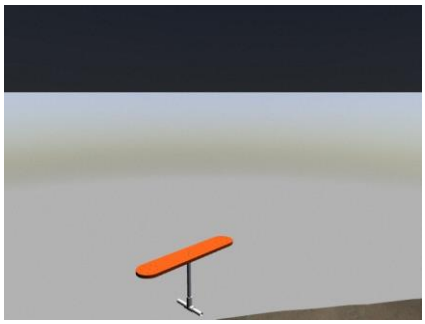
Bin 2



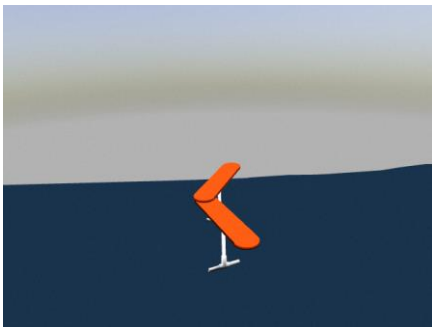
Gate



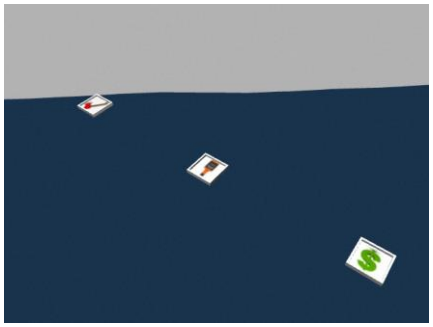
Octagon



Marker



Marker 2



Table

Torpedos

So, what is the number of classes (objects) that should be shown per group image? The answer depends on the Robosub challenge, let's use the gate example. At the very first beginning the submarine must identify the position of the *gate* class in order to reach it. Then, it must identify the *gangster* and the *police* classes to decide whether to go through one or the other, depending on the role given by Robosub. It is clear with this example that the three classes will always appear in the same image, so it would not make sense to take one image per class. The same occurs at the Table image, where 3 classes are separated from each other, but the challenge will always show them in that order and position, so no problem will be shown.

For this the dataset, **100 pictures were taken per image group (or 1000 group images in total).**

III. Data Augmentation.

Data augmentation is used to increase the amount of images taken, however in this edition we took a different path as previous years. This data augmentation version master is slightly different than the original master in https://github.com/vanttec/vanttec_classic_vision, because the scale transformations were eliminated and it was modified to duplicate also the .txt labelling files, so the **augmentation could be performed after labelling and not before.** Normally it is all the way around, due to the fact that the coordinate position of the object of interest is modified, because it is scaled, turned, among others. It was decided to leave the report in this order, even though labelling was made before, because it is the normal path to follow.

Follow the next steps to do the required data augmentation.

1. Clone https://github.com/vanttec/vanttec_classic_vision/tree/afterLabel
2. Setup:
    - 2.2 cd vanttec_classic_vision
    - 2.3 mkdir build
    - 2.4 cd build
    - 2.4 cmake ..
    - 2.5 make clean
    - 2.6 make
3. From the file src / data_augmentation.cpp the code **lines 74, 76 and 78** are the only ones that you have to change, **every time you change them repeat the instruction 2.6 “make”**

The order of your files should look like this:

data_augmentation
... dest
... ... ax
... ... etc
... source
... ... ax
... ... etc
... vanttec_classic_vision
... ... src
... ... data_augmentation.cpp

Your flow may be like this (note that the file we run is different from the MAIN, it is not the one with the test termination)

```
C:/.../vanttec_classic_vision>
        make
        cd build/bin
        ./data_augmentation
        cd ../../
<You make changes to choose another folder>
make
<repeat the beginning>
```

IV. Labelling.

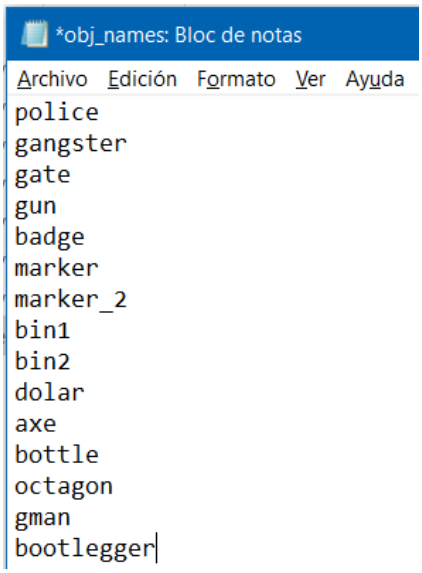
Labelling is used to tell YOLO the exact position of every image inside the dataset taken. It is made commonly by selecting a bounding box that encircles the class, the output of this section is a file.txt with the class number and the four points of the bounding box (w,h, w+dw, h+dh). The output is a file.txt with the same name as the image just labelled. On this Robosub edition two software options were used at the same time and no problem was noticed.

I. YoloLabel[5]:

YoloLabel works with Windows and the setup is very friendly, so it is a good option to follow.

It needs the dataset directory and a obj_names.txt, where all the classes are shown:

[5] https://github.com/developer0hye/Yolo_Label.git



Just select the class and create the bounding box around the object, this procedure must be repeated until every image of the dataset is labelled.

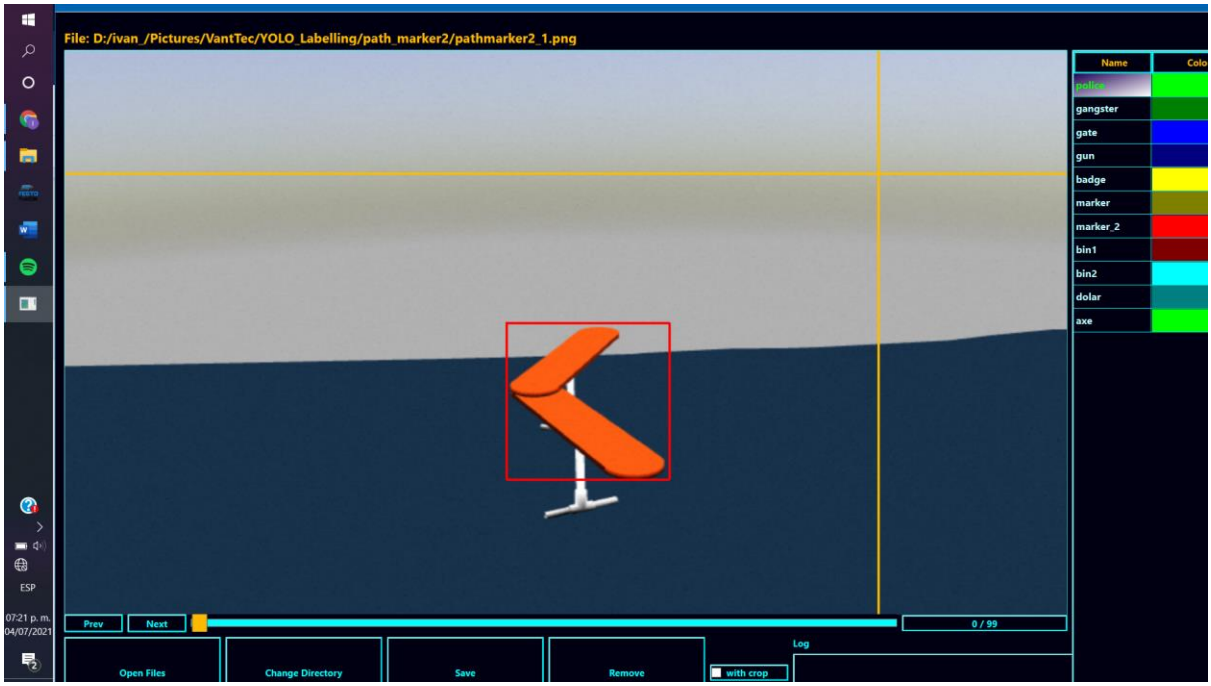


Figure: YoloLabel

II. vanttec/OpenLabelling[6]

OpenLabelling works exactly the same, but the .txt with classes is named here as class_list.txt and Ubuntu is used, for more information follow the vanttec/OpenLabelling/README.md Just be sure that your computer has the following requirements: **python3**, **OpenCV >= 3.0** and **numpy** (tqdm and lxml are

[6] https://github.com/vanttec/OpenLabeling.git

also mentioned). There are also some things to notice specially with ROS, for this reason please look also at the *OpenLabelling_Setup*[7] document. I would highly recommend the first option (YoloLabel) as almost no setup is needed and works just fine.

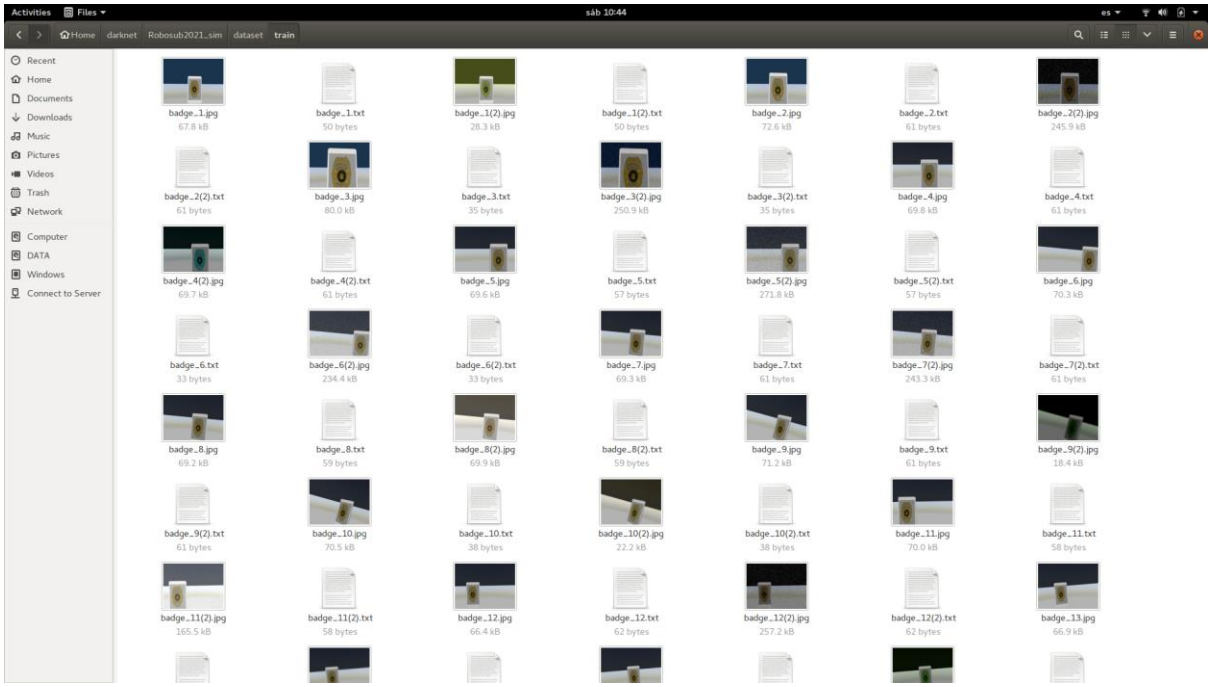
V. Training

- **Making the directories.**

There are some things to consider before training, the first step is to place the Robosub2021_sim directory inside darknet and create the 5 directory shown below.

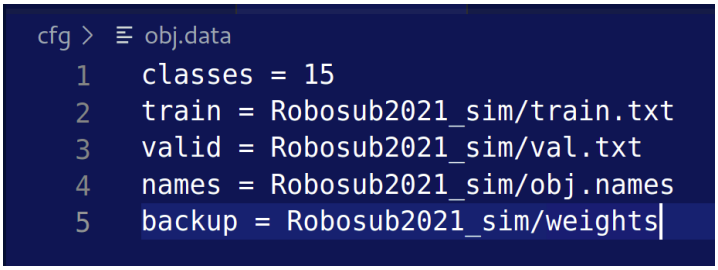
```
ivan5d@ivan5d-OMEN:~/darknet/Robosub2021_sim$ tree -d
.
├── cfg
├── dataset
│   ├── train
│   └── val
└── weights

5 directories
```

For this edition the 70/30 ratio was used, so the whole dataset was be divided 70% for training and 30% for validation. For this reason the images and their files.txt were divided and the 70% of them were introduced into /train and 30% into /val.



7 https://docs.google.com/document/d/1nADOQwWTilGP7BCgS3nfhwNpWMkUF_cXxYqCwqoXoZs/edit?usp=sharing

Now let's look at the /cfg, inside of it there are two important files. The first one is obj.data which contains the paths of the files used by darknet, it is clear that the number of classes and the name of the directory will change depending on the competition.



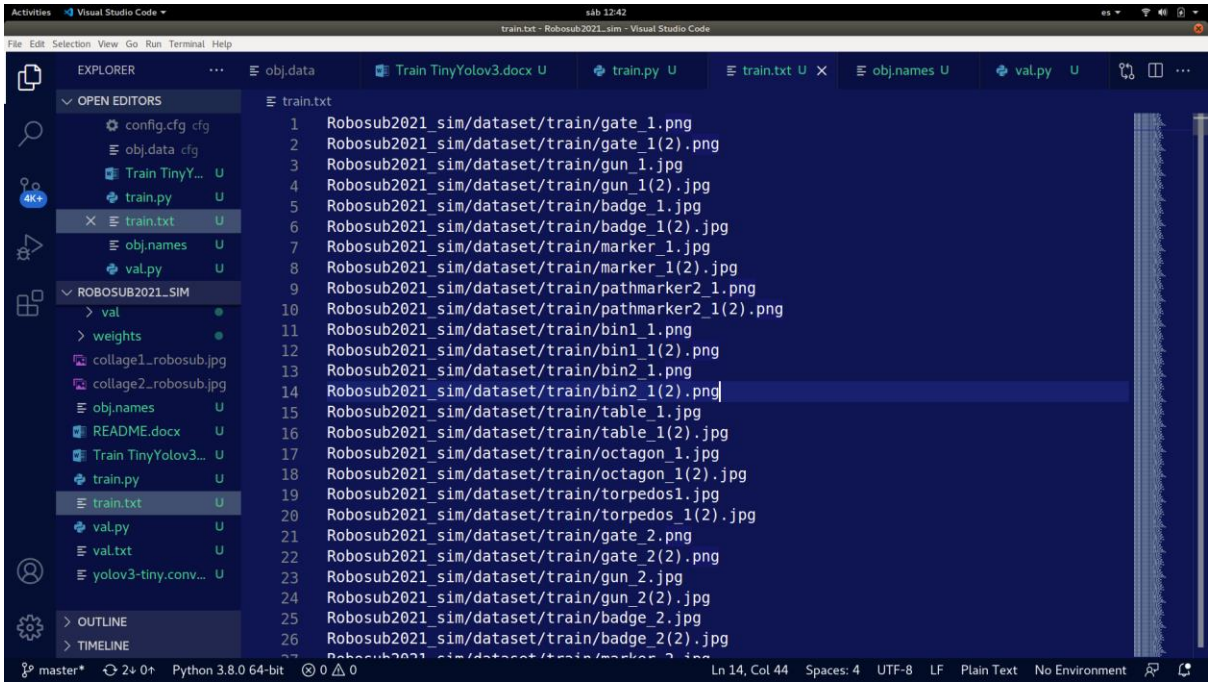
Then the config.cfg is shown, this file has the information for YOLO to work, just take the one already made and modify the next parameters.

- Lines 3 and 4: comment
- Lines 6 and 7: uncomment
- Lines 127 and 171: use the formula of filters = (classes + 5)*3
    - Example:
        - filters = 60, with our 15 classes because (15 + 5)*3 = 60
- Lines 135 and 177: modify the number of classes

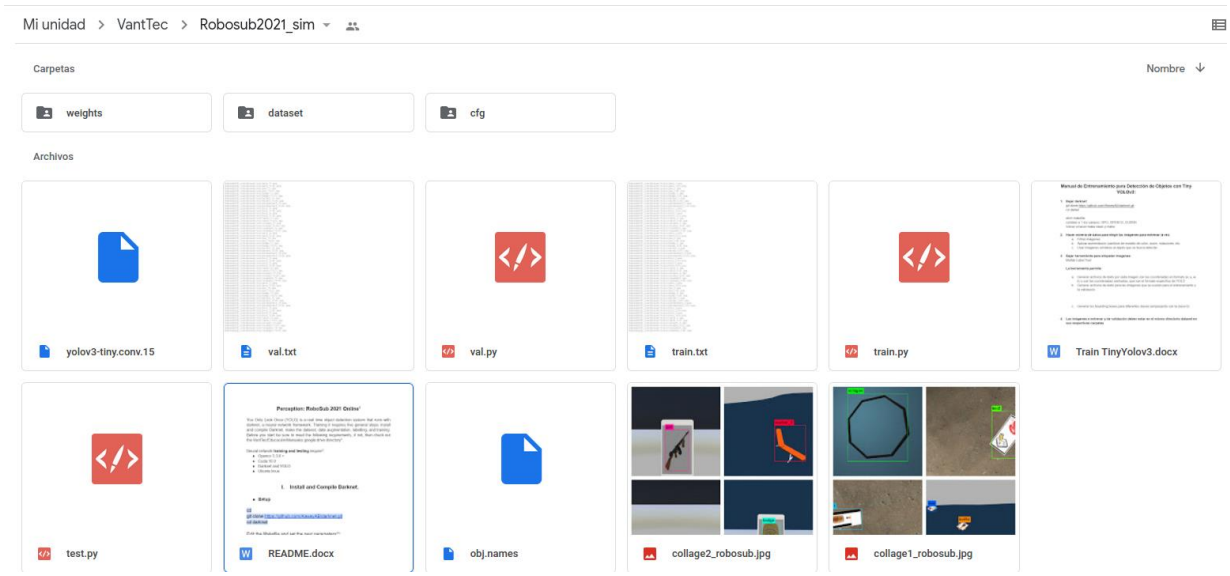
Now let's make the train.txt and the val.txt

```
cd
cd darknet/Robosub2021_sim
chmod +x train.py
chmod +x val.py
./val.py
./train.py
```

After running these two scripts, two files should be created with some paths. The train.txt gives the paths for the images that are going to be used for training, and the val.txt are the ones used for testing.



Every path is fixed with the darknet directory, for this reason, each one starts with the name of the directory inside darknet, in this case *Robosub2021_sim*. The /weights is now empty, but during training some files.weights will be generated and stored there. The final product is shown below.



- **Training**

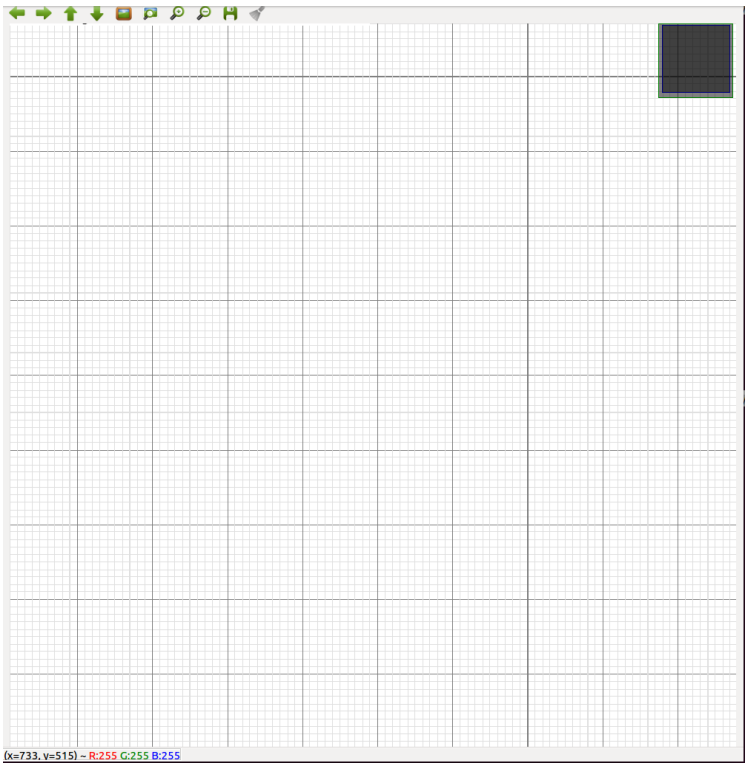
Edit darknet/Robosub2021_sim/cgf/config.cgf

[net]

Testing
#batch=1 **<- Comment this line**
#subdivisions=1 **<- Comment this line**
# Training
batch=64 **<- Uncomment this line**
subdivisions=8 **<- Uncomment this line**

./darknet detector train Robosub2021_sim/cfg/obj.data Robosub2021_sim/cfg/config.cfg Robosub2021_sim/yolov3-tiny.conv.15 -map

Expected Output:



Just leave the computer there until training is finished, normally until the current average loss is below 0.05. This may occur approximately in 4 hours, but it depends on the computer used. If you're training you may still use the computer for other purposes, but be sure to run things light enough for not crashing.

For retrain from other weight:
./darknet detector train Roboboat2019/cfg/obj.data Roboboat2019/cfg/config.cfg Roboboat2019/weights/..FILE_NAME..

Output example for training at 10 000 iterations:

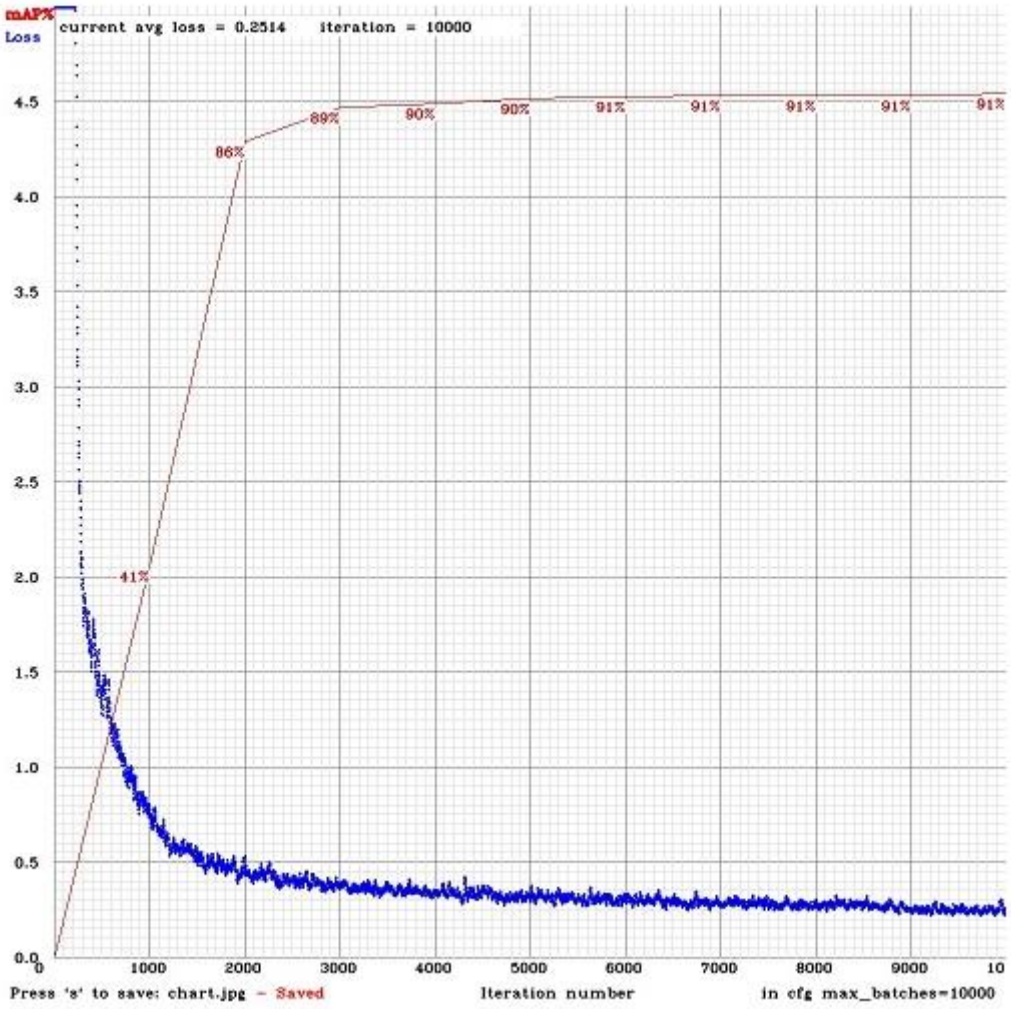


Figure 4: Training graph after 10 000 iterations

VI. Testing Weights Precision (mAP)

The last step to follow is testing the results.

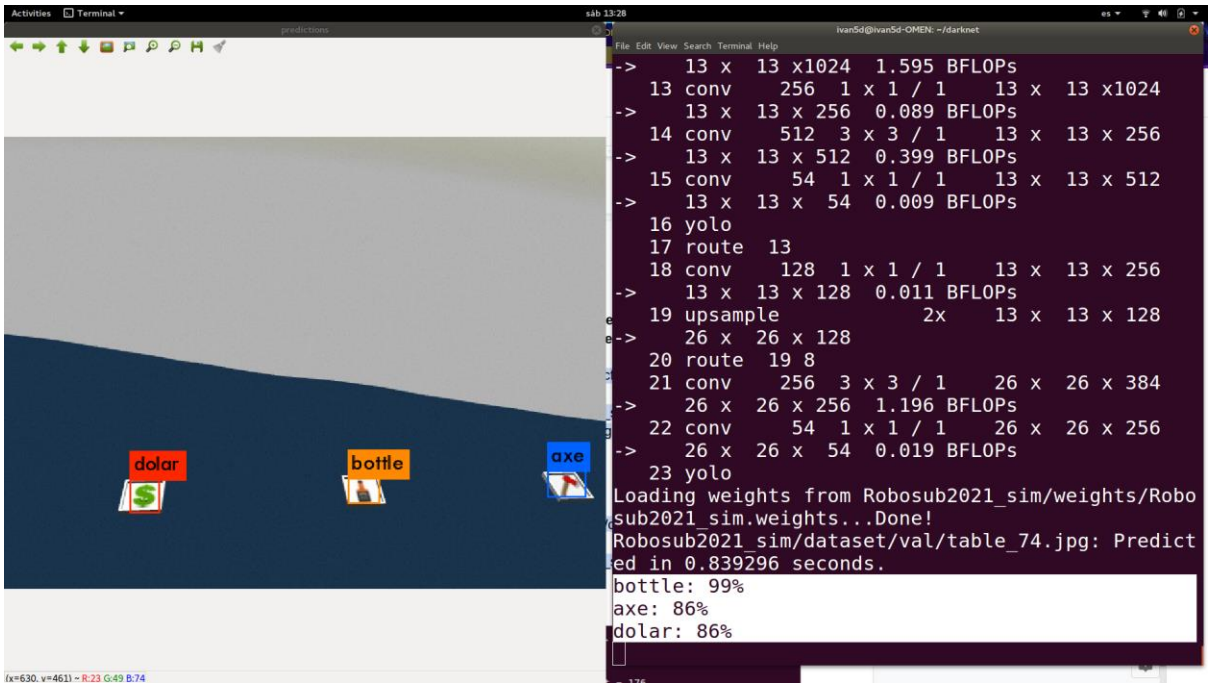
Edit darknet/Robosub2021_sim/cgf/config.cgf

[net]
# Testing
batch=1 **<- Uncomment this line**
subdivisions=1 **<- Uncomment this line**
# Training

#batch=64 **<- Comment this line**
#subdivisions=8 **<- Comment this line**

./darknet detector test Robosub2021_sim/cfg/obj.data Robosub2021_sim/cfg/config.cfg Robosub2021_sim/weights/Robosub2021_sim.weights Robosub2021_sim/dataset/val/table_74.jpg

Expected outcome:



VII. Final Comments

1. Everything was made for 15 classes including the dataset, however the last two classes were not trained due to lack of time. Thus, if you look inside the Robosub2021_sim directory you may see some things that don't match with the information given. For example, both files inside /cfg, the obj.names, the val.py and val.txt, and the train.py and train.txt were modified for only 13 classes. However, don't worry, you can just download the /Robosub2021_sim, test it and it will work just fine.

2. Whenever you have doubts sent me a message (Ivan Díaz) or to Saúl, and please don't share this document outside the VantTec team.