### Máster Universitario

### en Ingeniería Industrial

### (MUII)

### Universidad de Alcalá

### 202005 Sistemas de Percepción

### PRÁCTICA FINAL

### *OBJECT DETECTION AND 3D ESTIMATION USING CAMERA*



### 1. INTRODUCTION

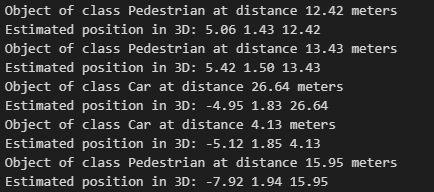
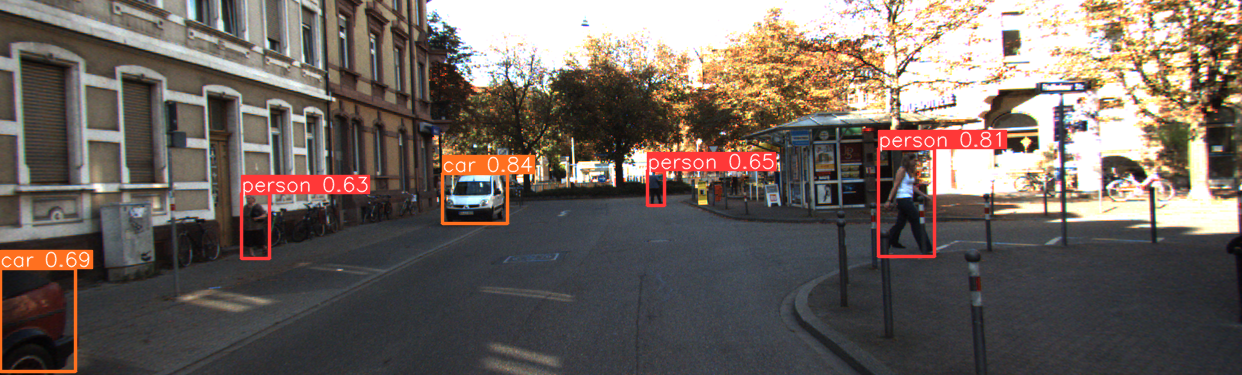
The main objective of this work is the implementation of a camera object detection pipeline in Python in order to generate estimated 3D positions of objects of interest in the scene of an autonomous driving scenario.

To begin with, all objects of interest in the input image are located using a pre-trained Yolo (You Only Look Once) object detector. This information is used to perform a geometric projection into the 3D world using the camera parameters, estimating the distance to the camera and generating a bounding box for each object. In addition, it is proposed to implement an evaluator that produces metrics to measure the performance of the Yolo 2D image detector.

The different images that will be used are from Kitti autonomous driving dataset, in particular from the 3D object detection benchmark (<https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d>).

### 2. GENERAL DESCRIPTION

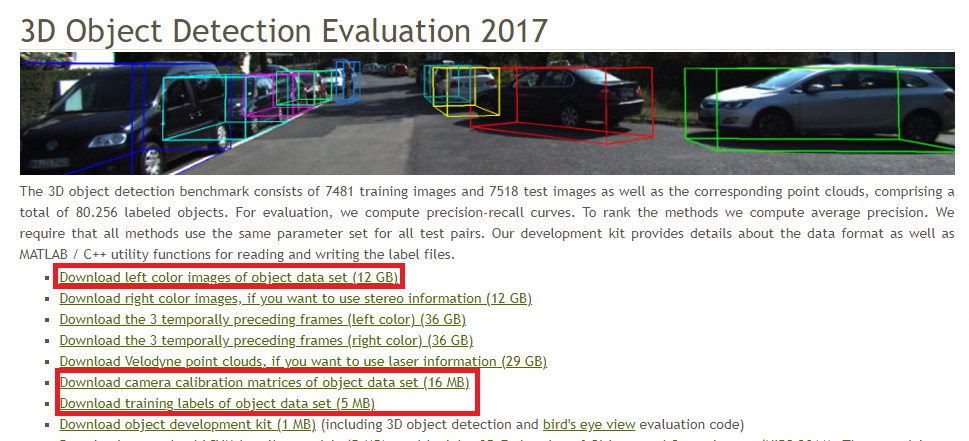
This work relies on external code and resources, therefore it is recommended to follow the steps described in this section inside an empty folder to avoid dependency or file issues. A sequence of the steps is despicted in Fig. 1.



**Figure 1:** Different steps of the 3D object detection algorithm.

1) Download and extract the different images, labels and camera data from the data subset in: URL o BLACKBOARD.

(Optional) It is possible to download all the 7481 **training** images, in order to work with the whole dataset, from the official website of Kitti (<https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d>).



**Figure 2:** Data used from Kitti.

2) Download the official Yolov5 repository inside the project folder: <https://github.com/ultralytics/yolov5>.

You can either download the zip directly or clone it with:

git clone https://github.com/ultralytics/yolov5.git

The file structure containing all external data should be the following:

Practica/

yolov5/

Kitti/

image\_2/

000000.png

...

label\_2/

000000.txt

...

calib/

000000.txt

...

3) Install the necessary Python dependencies to process the images and perform the inference with Yolov5 models. It is recommended to install Anaconda on Windows or Conda on Linux.

Linux (shell commands):

conda create -n practica\_percepcion python=3.10

conda activate practica\_percepcion

cd yolov5

pip install -r requirements.txt

Windows:

Open Anaconda Navigator

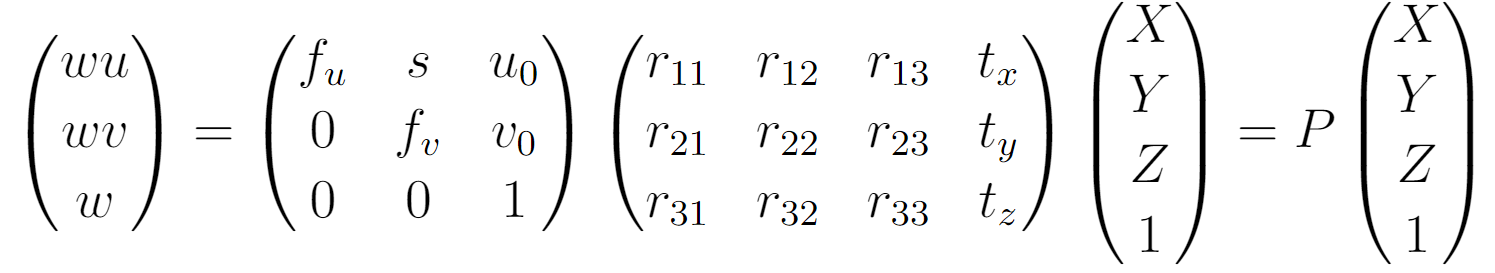
Go to environments 🡪 Import

Select *requirements.txt* from local drive inside the yolov5 folder (Pip requirements file).

Once all the prerequisites are installed, before running your Python code, you should activate the environment with *conda activate practica\_percepcion* on Linux or from Anaconda Navigator on Windows.

4) Using one of the Yolov5 models, detect all objects of interest in all Kitti images. It is recommended to read the information provided in <https://github.com/ultralytics/yolov5/issues/36> about using Yolov5 from PyTorch Hub and loading a pretrained model from a local repo. Save the detections in .txt or .csv files in another folder as they will be needed later.

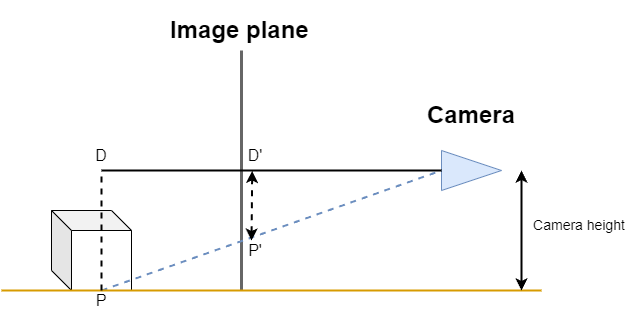
5) Project each detected object (files saved in the previous step) into the 3D world using the camera coordinate system. The rectified projection matrix can be found in each calibration file (see development readme for more information) and the camera position can be found in the official website (<https://www.cvlibs.net/datasets/kitti/setup.php>). As shown in equation 1 the camera loses all 3D information, therefore to estimate the position of the objects it can be assumed that the lower part of the 2D bounding box is at ground level.



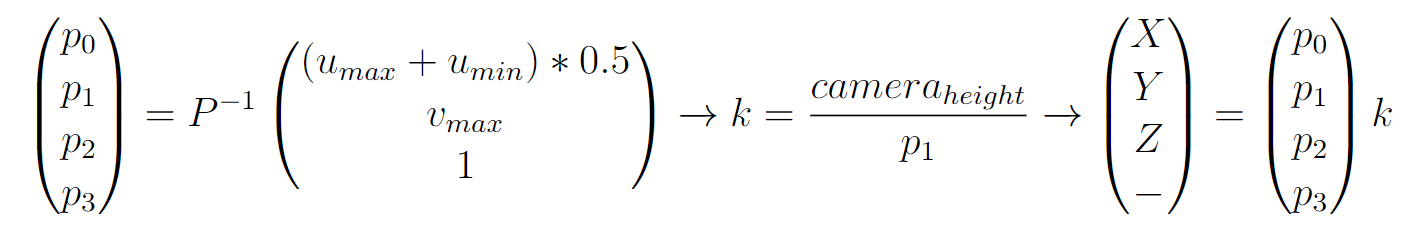
**Equation 1:** Projection of a 3D point into pixel coordinates.

With this approximation and the pin-hole model of the camera, the distance and approximate coordinates of the object can be calculated (Eq.2) using the triangle similarity theorem between the two triangles shown in Fig.3.

Plot the distance, class and the 2D bounding boxes from the camera to each object in the same image using **cv2.putText**() and **cv2.rectangle**().



**Figure 3:** Relation between camera plane and 3D projection.



**Equation 2:** Estimation of 3D coordinates from pixel.

6) Assuming that all cars have the same 3D size (mean of all cars in Kitti), generate and plot on the original image (Fig.1) the different 3D bounding boxes for objects that belong to the ‘Car’ class. It can be assumed that all cars have the same orientation as the camera (90 degrees of rotation respect to the x camera axis). The projection of the points can be done with Eq.1.

7) Implement a simple evaluator that generates the Precision and Recall metrics by classifying all previous 2D yolo detections into TP (true positive), FP (false positive) or FN (false negative). Each of the files in the label\_2 folder contains the ground truth objects for each image, following the format specified in the readme file from the development kit. A true positive is considered if the IoU (Intersection over Union) between the ground truth and the detection is greater than 0,5.

P = TP/(TP + FP) R = TP/(TP+FN)

### 3. EVALUATION CRITERIA

**≤ 4 points:** Yolo detections on Kitti.

**≤ 7 points:** Estimate the 3D position of all objects (X,Y,Z).

**≤ 9 points:** Plot the 3D boxes in the images.

**≤ 10 points:** Calculate precision and recall of 2D detections.