Object Detection-Based Behaviour using YOLO on Thymio

Francesco Saverio Zuppichini[1] and Alessia Ruggeri[2]

Abstract— **The ability to classify objects in both space and domain is encoded in an enormous variety of species on the planet. Recently, deep learning approaches have been developed to make machines able to mimic the same task with interesting results. With this paper, we propose a minimal architecture to archive the same goal using a two wheels robots, the Thymio, with a mounted frontal camera in order to make it act based on the objects detected in the surrounding.**

I. INTRODUCTION

Finding objects in space by identifying their identity and their coordinates is a basic task that almost each animal is able to accomplish by feeding the inputs from their eyes to the brain. The ability to solve this problem is extremely important in robotics, where a machine needs to act accordingly based on its surrounding. With our architecture, only the raw data from the camera are used as input to an Object Detection model and the decision are based directly on the prediction, without the need of any other sensor.

The paper is structured as follows: we will first describe the architecture by showing each individual part of the topology and how they interact; then, we will discuss the challenges during the development process and finally we will show the results.

II. ARCHITECTURE OVERVIEW

We used ROS[**ROS**] as operating system to develop and deploy our code on the Thymio. Our architecture is composed by:

1) A robot with a mounted camera and an on-board computer
2) A web server running the Object Detection model
3) A ROS node running on a local machine that moves the robot

As a robot, we used the *MightyThymio* [2], which is a robot developed on top of the Thymio, a small mass-produced mobile robot usually used in primary and secondary education. The MightyThymio has a powerful Odroid C1 on board, a 1GB quad-core ARM Linux machine, with a HD camera mounted on top. The authors also implemented a ROS node to communicate with it and open-sourced the code.

MightyThymio

In our project, the MightyThymio has the same configuration proposed in [2]. Briefly, the on-board computer expose the Thymios sensors through ROS. Thus, another ROS node can publish and subscribe in order to communicate with its sensors. For example, we can subscribe to the *camera sensors* to get information from the proximity sensors to avoid hitting obstacles. Also, we can simply subscribe to the camera's topic to get both compressed and uncompressed images.

Local ROS node

We created a new ROS node to communicate with the running node on the MightyThymio in order to make it act accordingly with a specified behaviour based on the object detected by the model. The user specifies a set of target classes, for example *dog* and *cat*. Then our node, after having subscribed to the camera topic of the MightyThymio, sends the fetched image to the model trough an HTTP request and gets the bounding boxes for the detected classes. If there is no target prediction, the robot enters in `exploration` mode and starts spinning around. When one or more of the target classes are found, then our robot will start going forward while keeping the bounding box of the target in the centre of the camera in order to follow it. If we detect more classes, then the closest, which is represented by the biggest bounding box area, will be followed. In addiction, we have an obstacle-avoiding algorithm in order to properly stop the Thymio before it hits the target.

Object Detection REST-API Web Server

We used pre-trained YOLO[4], the state-of-the-art real-time object detection model, to predict the bounding boxes for each class. The model was trained on the COCO(Common Object in Context)[1] Dataset, that contains 80 classes.

On top of that, we built a REST-API Web Server using Python3 and Flask, a micro-framework to create Web Servers, to make the model accessible from the Web.

We decided to decouple the logic behind the robot behaviour and the model in order to make our architecture more scalable. For instance, if needed, we can deploy our model in the cloud while keeping the running ROS code intact.

III.

A. YOLO: Real-Time Object Detection

YOLO [4] is a trained model for real-time object detection. While prior work on object detection repurposes classifiers to perform detection, YOLO handle the object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. YOLO avoids resorting to complex pipelines that are slow and hard to optimize; instead, it uses a single convolutional neural network that simultaneously predicts multiple bounding boxes and class probabilities for those boxes. Using YOLO, You Only Look Once at an image to predict what objects are present and where they are.

YOLO neural network uses features from the entire image to predict each bounding box simultaneously across all classes. The convolutional neural network contains 24 convolutional layers, that extract features from the image, followed by 2 fully connected layers, whose aim is to predict the output probabilities and coordinates. The system divides the input image into an $S \times S$ grid; if the centre of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. There is also a fast version of YOLO, designed with a neural network with fewer convolutional layers and fewer filters in those layers.

YOLO is extremely fast, it reason globally about the image when making predictions, it learns generalizable representations of objects and it is open source. YOLO still lags behind other detection systems in accuracy; while it can quickly identify object in images, it struggles to precisely localize some objects, especially small ones. However, it is still the state-of-the-art model for object detection in real-time.

For our project, we used the third version of YOLO, or YOLOv3 [3], which has been improved by making it a little bigger than before, but more accurate.

IV. Code

V. Server

VI. Results

VII. Conclusion

In this paper we presented a light and scalable architecture that uses a real-time Object Detection model to correctly find objects in the surrounding environment and to perform actions based on that. Since we only require a camera, our work can be used into similar robots running ROS with the correct hardware in order to extend their functionalities. Furthermore, since we used YOLO to perform the classification task, the model can be easily re-trained with other classes to expand the context domain. This symbiosis between Machine Learning and Robotic highlights the power of both areas by creating new interesting user cases and scenarios in which both disciplines can shine.

References

[1] Tsung-Yi Lin Michael Maire Serge Belongie Lubomir Bourdev Ross Girshick James Hays Pietro Perona Deva Ramanan C. Lawrence Zitnick Piotr Dollar. “Microsoft COCO: Common Objects in Contex”. In: 2015.

[2] Jerome Guzzi et al. “Mighty Thymio for Higher-Level Robotics Education”. In: *AAAI*. 2018.

[3] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: 2018.

[4] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: 2016.