Parking Slot Detection for Autonomous Vehicles from Surround Camera Images

Technische Universität Berlin

Hella Aglaia Mobile Vision GmbH

Project Members:

Lei Kang

Furkan Kilicaslan

Rudhishna Narayanan Nair

Jiaqiao Peng

Abstract

In the Winter semester of 2018/19, our group members applied ourselves to this project, aimed at the autonomous parking task, supervised by Professor Olaf Hellwich from Technische Universität Berlin and tutors from Hella Aglaia Mobile Vision GmbH. Our group is the camera group, our task is to detect the free parking slots based on the camera sensors, then translate our results to the fusion group.

Keywords: *Parking slot, Camera, Object detection, Yolo, Coco, Cassandra*

1. Introduction

With the development of technology, autonomous driving has become an increasingly popular research field. Autonomous parking in a parking lot is a very important part of the autonomous driving experience. With the cooperation and fusion of different types of sensors such as camera sensors, radar sensors and laser sensors in a vehicle, autonomous parking now is a feasible approach.

Using the camera sensors in a car, a camera model can be built to recognise the free parking slots. The main task of the camera model is object detection. Recently many algorithms and architectures are developed for object detection. R-CNN (1) algorithm proposed by Ross Girshick et al uses selective search to extract just 2000 regions from the image which is called the region proposals instead of trying to classify a huge number of regions. Then the region proposals are fed to R-CNN.

After R-CNN, Ross Girshick et al developed the Fast R-CNN (2) algorithm by solving some drawbacks of R-CNN. Fast R-CNN algorithm is similar to R-CNN. The difference is that in Fast R-CNN the input image is fed to the CNN to generate a convolutional feature map. R-CNN & Fast R-CNN use selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. Based on Fast R-CNN algorithm, Shaoqing Ren et al develops the Faster R-CNN [3] . Similar to Fast R-CNN, Faster R-CNN algorithm uses the images as inputs to a convolutional network to generate a convolutional feature map. Instead of using a selective search algorithm on the feature map to identify the region proposals, Faster R-CNN uses a separate network to predict the region proposals.

Our team used a faster algorithm to detect the objects which have some tradeoffs. This algorithm is YOLO(You Only Look Once) version 3, to use this algorithm, OpenCV 3.4.2 and higher version are required.

2. Methodology

2.1 Algorithm: Yolo V3

Before YOLO all the object detection methods can be classified as two-stage algorithms. They had to perform some type of detection and then classification on the Region of Interest (ROI). The mainstream series of R-CNN algorithms are; R-CNN, Fast R-CNN and Faster R-CNN. Firstly they use region proposals methods such as selective search or region proposal network to generate potential bounding boxes in an input image and secondly run a classifier on these bounding boxes and finally refine these bounding boxes, eliminating duplicate boxes and rescore these boxes after classification. Region proposal is a slow and source-consuming process and even the best one of them, Faster R-CNN, is infeasible to be used real-time in this project.

Though they achieve high precision, they are not suitable for us where we need the algorithm working on a source-limited situation for real-time video detection.

Different from the series of R-CNN algorithms, YOLO (4) algorithm treats object detection as a regression problem, straight from image pixels to bounding box coordinates and class probabilities. Therefore this algorithm is named as YOLO: *You Only Look Once* at one image to predict what kinds of objects are presented in the image and where the objects are in the image. It predicts all bounding boxes across all classes for an image with a single network.

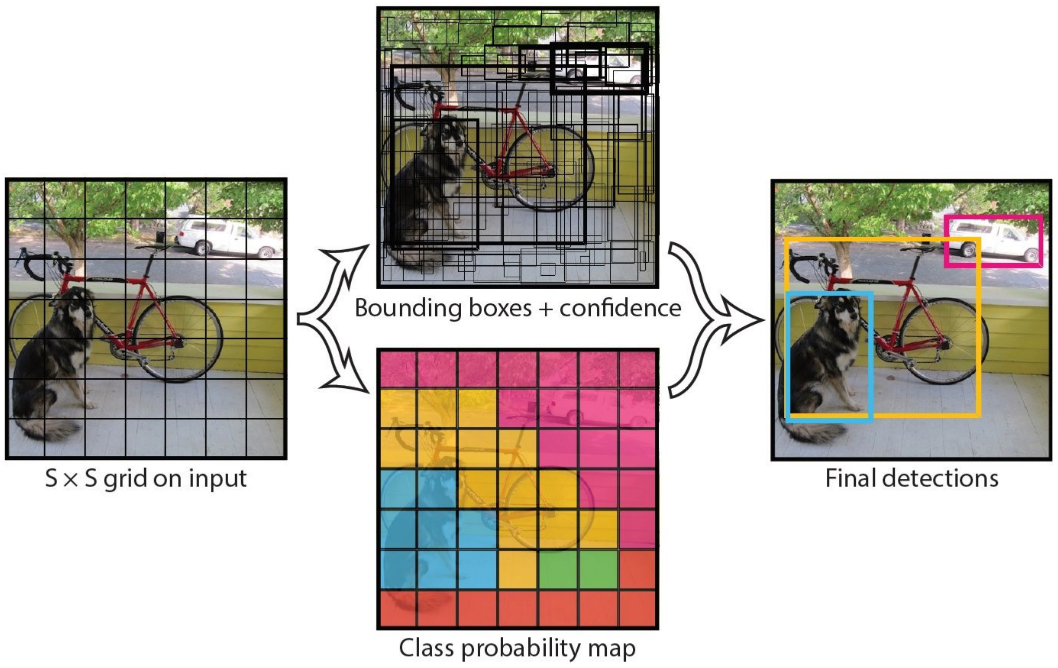

Figure 1: the model of YOLO algorithm (5)

Figure 1 shows a model of YOLO algorithm. YOLO algorithm divides the input image into an S × S grid. The grid cell will detect the object if the centre of this object falls into the grid cell. Each grid cell predicts bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the YOLO model is, whether the box contains an object. In order to get the confidence score for each bounding box, all the bounding boxes will be scanned, then only the ones with high confidence scores are kept and the bounding box will be assigned the class label with the highest score for the box.

In this process, the size of the image needs to be set. The width and height are predicted relative to the whole image. For each of the bounding box, the network outputs a class probability and offset values for the bounding box. To get the class probability map, the confidence value should be given a threshold, when the confidence score is bigger than the threshold, then the bounding box contains an object, otherwise, the bounding box does not contain an object. The next step is to remove the bounding box with low confidence using non-maximum suppression. A threshold for the non-maximum suppression should be set.

YOLO is orders of magnitude faster than traditional region proposal based algorithm. It is not as precise as the best one of other algorithms but it achieved a trade-off between precision and speed. Besides, by using features from the entire image to predict each bounding box, YOLO algorithm makes a prediction by the global context in the image.

Our model is based on YOLO algorithm version 3 (6) . We set the confidence threshold as 0.5, and the non-maximum suppression threshold as 0.3. We set the size of the network's input image as 320×320.

2.2 Database: COCO

We pre-train our model on COCO public dataset by Darknet, a framework provided by the inventor of YOLO (7) . COCO dataset (8) contains photos of 91 objects types that would be easily recognised and a total of 2.5 million labelled instances in 328k images. In our model, we only use coco database to recognise the car. We can easily use opencv_dnn module to load YOLOv3 pre-trained models in our cpp code.

3. Implementation

3.1 Implementation to Cassandra

Cassandra is an integrated workspace built by Hella Aglaia Mobile Vision GmbH, in which all the works from the radar group, lidar group and camera group can be integrated and processed by fusion group.

Our model is built in visual studio with OpenCV. Then we implement our model into Cassandra. The type of the input is cImg, we need to convert it into the type Mat, in order to realise this process, firstly we convert the type cImg into QImage, then from QImage into Mat.

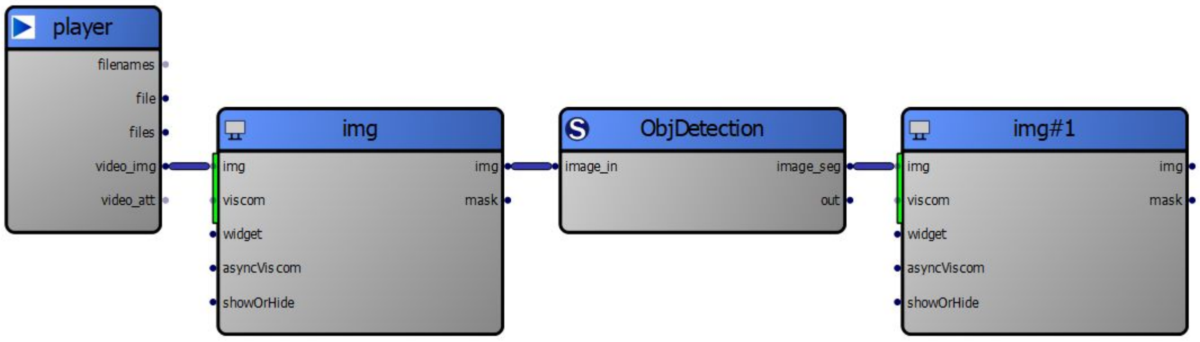

Figure 2: The overall system pipeline

Figure 2 shows the overall system pipeline in Cassandra. We build one station for object detection, for that we have programmed the code in visual studio. The first two stations are for viewing the frames, which are automatically created when we input the video in Cassandra. The last station is the image viewing station, which is inbuilt in Cassandra. We only choose it from their options and connect it to the output of the ObjDetection station. The C++ codes have been plugged in and we have linked OpenCV library within the project.

3.2 Result

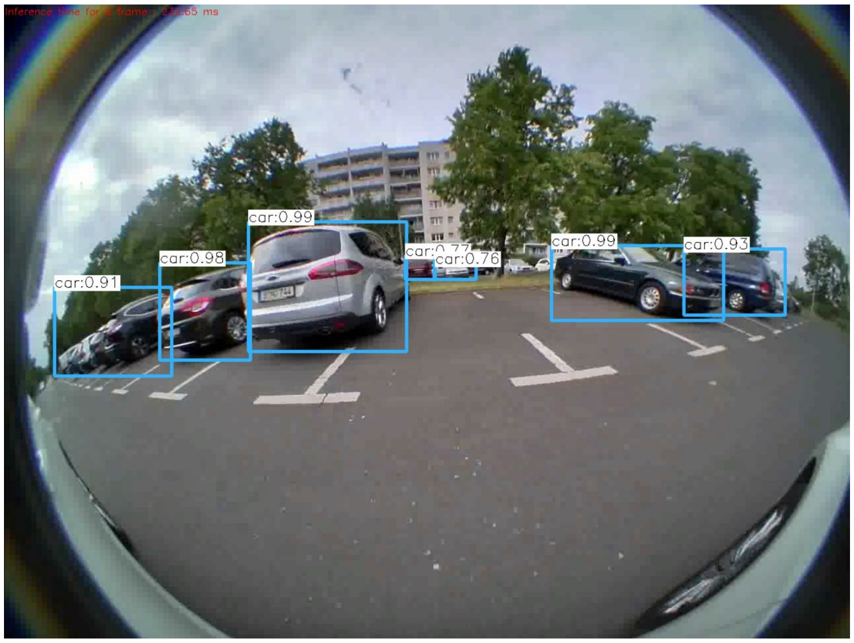

Figure 3: The screenshot of the video output of our model in Cassandra

Figure 3 shows the screenshot of the output video of our model in Cassandra. We successfully detect all the cars in the input video.

But our results were not successfully translated to the fusion group. For converting the coordinates to the vehicle coordinate system, we have tried the given functions. But unfortunately, we could not produce any results successfully.

4. Conclusion

As shown in the result of implementation, Our model can detect all the cars in the output videos, which can satisfy the accuracy requirement of autonomous parking in object detection. Meanwhile, real-time detection is another very important attribute of an autonomous parking system. Our model cannot detect the objects as fast as it is required in a real-life scenario. In future research, we can make improvements in this area. Another valuable improvement is that we can strive to achieve our first approach (Details please see the appendix 1).

5. Acknowledgement

We would like to express our special thanks to Professor Olaf Hellwich from Technische Universität Berlin as well as our tutors from Hella Aglaia Mobile Vision GmbH, who gave us the golden opportunity to participate in this wonderful project on the topic (Parking Slot Detection for Autonomous Vehicles from Surround Camera Images), which also pushed us in doing intensive research and we came to learn about many new concepts in autonomous driving and computer vision. We are really thankful to them.

Secondly, we would also like to acknowledge the hard work and dedication of our team members: Jiaqiao Peng and Rudhishna Narayanan Nair, who still persisted through the project in hard times when we immersed ourselves in a depressed atmosphere after the failure of our first approach. We could not have produced any results without their dedication.

References

(1): https://arxiv.org/pdf/1311.2524.pdf

(2): https://arxiv.org/pdf/1504.08083.pdf

(3): https://arxiv.org/pdf/1506.01497.pdf

(4): https://arxiv.org/pdf/1506.02640v5.pdf

(5): J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517–6525. IEEE, 2017.

(6): J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

(7): J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

(8): https://arxiv.org/pdf/1405.0312.pdf

Appendix 1

Simple introduction of our first approach

The original camera images that we got were the fisheye view images with a high field of view which creates distortion. We can not use these images directly without preprocessing them in our model training. Our idea was to rectify these images and then translate the fisheye view into a ground plane projection. Then to label the cars in the rectified images for the preprocessing and finally to train the model on this dataset. This model should contain the segmentation function and object detection function. For the segmentation, we can classify the entire parking space into parking slots and driving way. For object detection, we can classify the parking slots into free parking slots and occupied parking slots. All the above processes are operated in Python, Visual Studio and OpenCV. Then we update our model to Cassandra, and then test our model in Cassandra, after the successful test of our model, we translate our result to the fusion group. Unfortunately, the image rectification process caused a bottleneck in this approach and we had to pivot later to object detection through a larger pre-trained model.

Appendix 2

Contribution of our group member

Group Members	1. First approach	2. Deep learning model	3.Integration to cassandra	4. Presentation	5. Report
Lei Kang	x	x		x	xxx
Furkan Kilicaslan	xxx			x	x
Rudhishna Narayanan Nair	x	x	xxx	xx	x
Jiaqiao Peng	xx	xxx	x	xx	xx

x: take part in

xx: work hard

xxx: be mainly responsible