UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Rasmus Vahtra

Parking Space Monitoring and ID Based Car Tracking

Master's Thesis (30 ECTS)

Supervisor: Egils Avots, MSc
Prof. Gholamreza Anbarjafari

Tartu 2019

Parklakohtade monitoorimine ja autode ID-põhine jälgimine

Lühikokkuvõte:

Parklate jälgimissüsteeme on aegamisi arendatud ning realiseeritud, kuid vähe on süsteeme, kus kasutataks tehisnärvivõrke. Tehisnärvivõrkude arenedes on võimalik neid ka parklajälgimissüsteemides rakendada. Käesolevas magistritöös uuritakse YOLO närvivõrku, mis suudab pakkuda reaalajas kiirusi. YOLO eelmistest versioonidest ja eelistest teiste närvivõrkude ees tuuakse kokkuvõte. Siin magistritöös uuritakse ka tuvastatud objektide jälgimist unikaalsete ID-ga, kus viimaseid saadakse Kalmani filtrist. Seletatakse Kalmani filtri tööpõhimõtet. Ühtlasi seletatakse lahti käesolevas magistritöös kasutatud tarkvara disain ning pakutakse välja uus jälgimisüsteem parklakoha jaoks, mille kasutamist pole autori teada varem dokumenteeritud. Lõpus uuritakse tulemusi ning pakutakse välja lahendused teatud probleemidele. Ühtlasi kirjeldatakse võimalikke täiendusi, mida antud tööle teha saaks.

Võtmesõnad:

Tehisnärvivõrgud, YOLO, Darknet, Kalmani filter, parklatejälgimine, ID-põhine jälgimine

CERCS: Pilditehnika (T111)

Parking Space Monitoring and ID Based Car Tracking

Abstract:

Parking space monitoring research has been going on for some time already, but the field is still being widely researched. As neural networks are becoming better, they can be utilised in more areas, including parking space monitoring. This thesis takes an approach with YOLO that is capable of performing detections in real-time. A short summary of YOLO's previous versions and its advantages over other neural networks is given. In addition, this thesis also takes a look at unique ID based detected object tracking by using Kalman filter. The workflow of Kalman filter is explained. The overall design of the software used in the thesis is explained and a method for parking space monitoring that, to the author's knowledge, has not been documented before, is offered. Results of the thesis are analysed and

solutions are provided for some of the encountered issues. Some of the future possible improvements are explained.

Keywords:

Neural networks, YOLO, Darknet, Kalman filter, parking space monitoring, tracking IDs

CERCS: Imaging, image processing (T111)

Contents

Abbreviations, constants, terms 7

1 Introduction 9

1.1 Purpose of the Thesis 9

1.2 Technical Challenges 10

1.2.1 State of Cameras 10

1.2.2 State of Hardware 10

2 Related work in Parking Lot Monitoring 11

2.1 ParkingDetection 11

2.2 Parking Spaces with Mask R-CNN and Python 12

2.3 Intelligent Parking Tracker 13

3 General Overview of Used Solutions 14

3.1 YOLO - You Only Look Once 14

3.1.1 YOLOv1 14

3.1.2 YOLOv2 18

3.1.3 YOLOv3 22

3.2 Kalman Filter 23

3.2.1 Basics of a Kalman filter 23

3.2.2 Performing Predictions 23

3.2.3 Combining Predictions with Measured Data 26

4 Software Solution **27**
4.1 Darknet 27
4.1.1 Implementation of a Kalman Filter 27
4.2 Module 27
4.2.1 Pipe Reader 27
4.2.2 Displaying Detected Objects on Video Stream 28
4.2.3 Displaying Data in a Table 28
4.2.4 Parking Space Handler 28
4.2.5 Parking Space Monitor 29

5 Results **34**
5.1 Efficiency of Parking Space Solution 34
5.1.1 Adding Parking Spaces 34
5.1.2 Parking Space Monitoring 35
5.2 Problems Encountered and Attempted Solutions 36
5.2.1 Parking Space is Falsely Observed to be Unoccupied 36
5.2.2 New Tracking ID for Already Detected Objects 37
5.2.3 Object is Falsely Presented 40
5.2.4 Detecting Reflections of Objects 40
5.2.5 Detected Objects Exchange Tracking IDs 41
5.2.6 Hijacking Tracking ID 41
5.2.7 Occluded Parking Spaces 42
5.2.8 Parking violation 42
5.3 Others Problems Not Encountered 43
5.3.1 Weather Conditions 43
5.3.2 Unstable Video 43
5.3.3 Other Objects in Parking Slots 44

6 Further Possible Improvements **45**

6.1 Implementing a Custom Kalman Filter 45
6.2 Specialised Training of Darknet 45
6.3 More Cameras 45
6.4 Additional Functionality 46
6.5 Capsules Instead of Convolutional Neural Networks 46

Conclusion **46**

Acknowledgements **47**

References **48**

License **51**

Abbreviations, constants, terms

4K - display resolution, 3840 x 2160 with 16:9 ratio

software - the entire thesis containing Darknet and python module

GPU - graphics processing unit

CPU - central processing unit

YOLO - You Only Live Once, neural network, (there are 3 different versions, but in this paper YOLO represents YOLOv3 unless stated otherwise and in YOLO chapter)

DPM - deformable parts models

SVM - support vector machine

Fast R-CNN - fast region-based convolutional neural network

real-time speed - 20+ frames per second (developers of YOLO have set the threshold to 30+ frames per second, but in this thesis it is lower, because the camera used does not provide anything higher than 25)

AP - average precision, can also appear as mAP, which stands for mean average precision

IoU - intersection over union, a method where the percentage of unified area is found

parking lot - area containing parking spaces

parking space - single space slot that can be occupied by one vehicle if in standard size

tracking ID - a unique ID given to detected objects by Darknet

Darknet - implementation of YOLO

Kalman - prediction filter

Gaussian distribution - also known as normal distribution, it describes a data distribution where most values reside in the middle range and the rest on either side of the middle, forming a bell curve [1]

quadrilateral - polygon (shape) with four edges and 4 vertices

pentagon - polygon (shape) with five edges and 5 vertices

1. Introduction

Parking lots are important parts of any city that has a lot of cars. They can be located somewhere in the open, but they can also be internal parking lots, for example in special parking buildings or in big shopping centers. Some parking lots are unsupervised (especially open area ones), where cars come and go without supervision. There are also parking lots that give a ticket upon entering, because parking there costs money. This is usually used in internal parking lots. Some internal parking lots also use cameras that take a picture of the car's number plate upon entry. This feature can be used to make sure that people can not easily cheat their way out of the parking lot without paying, but it also adds additional convenience for users, for example, upon exiting a second camera checks the number plate again and if the car is allowed to leave, it will open the barrier without requiring the driver to roll down their window and insert the ticket. In most cases this is where the information gathering about the cars ends. Some internal parking lots also feature a light above each parking space that indicates whether the space is unoccupied with a green light or whether it is occupied with a red light. This is usually, however, entirely local and is not connected with the other systems of the parking lot (it is not known which car exactly is under it).

1.1 Purpose of the Thesis

The purpose of this thesis is to figure out the current challenges and possible solutions for developing a real-life deployable parking lot monitor and a car tracker by using a camera overviewing the entire parking lot or a part of it. More specifically, the general idea is to detect whether parking spaces are unoccupied or occupied, perform ID based car tracking and apply additional functionality, such as time measurement for parking space occupation and statistics generation. The resulting solution should handle these tasks in parallel and with real-time speeds. It should also be adaptable to different requirements.

1.2 Technical Challenges

1.2.1 State of Cameras

In general surveillance cameras are not designed for high resolution videos, but rather for the mere ability to observe certain places. As technology evolves, so do the surveillance cameras and nowadays some can even produce 4K resolution video at 30 frames per second, but this is not always ideal, especially in cases where the video from the cameras is recorded and kept for an extended period of time. It is also not ideal when this video is being used to detect cars because high resolution means longer scanning times, which in turn will have an impact on performance. So lower resolution videos are desired, since they have less impact on performance but on the other hand, the resolution should not be too small because then it might get too difficult to detect cars.

In addition to resolution, frame rate is another important factor. Even if a video stream is acquired at the desired resolution, it will not be enough if the frame rate is only at 1 frame per second. Generally, anything below 20 frames per second can already hinder performance of car tracking. Although the software used in this thesis could still perform some easier tasks, such as detecting whether a parking spot is occupied or unoccupied, having 1 frame per second would simply prevent some other desired features from working properly, such as ID based tracking.

1.2.2 State of Hardware

Real-time performance speed is desired for this thesis. This requires good hardware, which can run the software in real-time. The most important hardware piece is a GPU, because it is capable of parallel computing. The software could also work on a regular CPU, but it would be a lot slower, due to the difference in workflow between a CPU and a GPU. This does not mean that a CPU could be low quality and slow because a CPU's performance is still important even when using a GPU for detection. This means that in addition to a good video feed, emphasis must also be put on computing hardware. Currently the software is only tested on an Intel i7-6700K CPU and a nVidia GTX 1080 GPU, which provide satisfactory real-time performance with a video resolution of 1920x1080 at 25 frames per second.

2. Related work in Parking Lot Monitoring

There are other similar projects of parking space monitoring. Unfortunately most of them lack detailed documentation and their exact methods are not described. A few projects are developed by companies and their projects are also more sophisticated. One of the them listed here is also being sold. This also means that the source code is closed and any detailed analysis and comparison can not be made.

2.1 ParkingDetection

Figure 2.1: An example application of the ParkingDetection software. This solution also keeps track of parking spaces with charging stations. [2]

ParkingDetection, developed by RCE Systems, is a software that utilises computer vision and image analysis for parking space monitoring. This software entered the commercial market in 2018 and is probably the leading solution in the field. Their system works by taking the video feed from cameras and sends it to a cloud-based AI for analysis. Their solution uses multi-camera setups for better monitoring and information gathering (see Image 2.2). In addition to parking space monitoring,

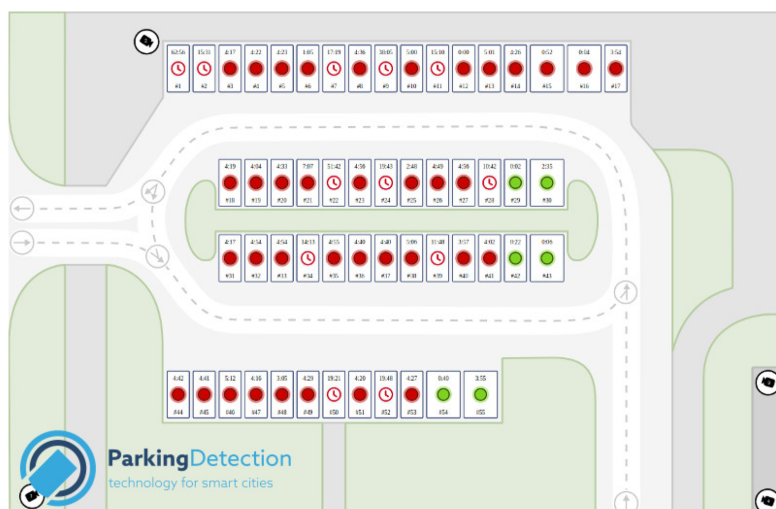

Figure 2.2: A CGI layout of one of the parking lots monitored by a ParkingDetection solution. The positions of 4 cameras can be seen. [2]

it can look out for badly parked cars, guide drivers to empty parking spaces, keep track of parking time, recognize license plates and also differentiate between special parking spaces like on image 2.1. According to their web page, it can be adapted to the requirements of every client. There is no mention of ID based tracking, but from the company's other solution DataFromSky [3], it can be estimated that they are capable of performing ID based tracking [4]. ID based tracking was performed on videos captured by drones. [2]

2.2 Parking Spaces with Mask R-CNN and Python

This solution is developed by A. Geitgey and is simpler in design than the software developed in this thesis. It also checks parking spaces and tries to determine whether they are occupied or unoccupied, but the approach used here is different. Instead of manually adding parking spaces, it learns about parking spaces by analysing detected cars - if some cars stay still for a set amount of time, it is then decided that the cars are in parking spaces. For occupation detection, the software uses the intersection over union approach in order to calculate how much of a detected car's bounding box resides in a known parking spot. The solution does not perform in real-time speed and it does not have ID based tracking. [18]

2.3 Intelligent Parking Tracker

Automatic Parking Management (also named Intelligent Parking Tracker) is a project presented by A. Khare. In a video he had uploaded to YouTube [5], he shows how his solution works. Since there is no further documentation in the video description nor in his Github project [6], it is unclear how the system determines whether a parking space is occupied or not. It is worth noting, however, that the method for adding parking spaces is very similar. Unfortunately it is not known how that data is later processed and used. In addition, based on what can be seen in the video, the performance speed is not real-time. It is unclear whether it is due to limitations of the proposed solution or whether the performance has been artificially slowed down. Also, there is no ID based tracking of detected cars.

3. General Overview of Used Solutions

The software developed in this thesis consists of two parts. One part is Darknet, specifically an enhanced Darknet build, created by AlexeyAB, for its several extra features, one of them being ID based tracking. This tracking is resolved with a Kalman filter that analyses detected objects and performs predictions on their next possible locations. A detected object will be given a unique ID, which allows tracking of the object as long as Darknet can see it.

The second part is a module programmed in Python 3. Darknet gives information about the detected objects with the output - their locations and the tracking ID. It does not do anything else, which is why a module has been added that can read this output and perform additional tasks with it, such as keeping track of detected objects. The main reason why this additional functionality was built with Python, is Python's flexibility and ease of workflow when developing and testing concepts. Without having intricate knowledge about any programming languages, one can potentially do more work in Python than in C or C++, which are the programming languages used in Darknet. Regardless of a programming language, the module can be later used to connect to more cameras or change neural networks.

3.1 YOLO - You Only Look Once

3.1.1 YOLOv1

YOLO's principle

Classic detection systems use classifiers to detect objects on frames. A classifier is taken for an object and the object is then evaluated at different scales and locations. Then during post-processing the bounding boxes are refined, duplicate detections removed and boxes rescored according to other boxes in the frame. The entire process is slow and difficult to optimize because they all need to be trained

separately. With the You Only Look Once (YOLO) system, the image is looked at only once and the object detection becomes a single regression problem, where looking at pixels is followed immediately by bounding boxes and class probabilities (see Fig. 3.1). In YOLO, a single convolutional network predicts multiple bounding boxes and class probabilities on a single frame at the same time. YOLO trains on full images, which allows direct optimization for detection. This is what makes YOLO amazingly fast. Thanks to having a regression problem, complex pipelines are no longer needed. The base network of YOLOv1 can run 45 frames per second on an nVidia Titan X GPU, which allows detection on live videos with less than 25 ms of latency. [7]

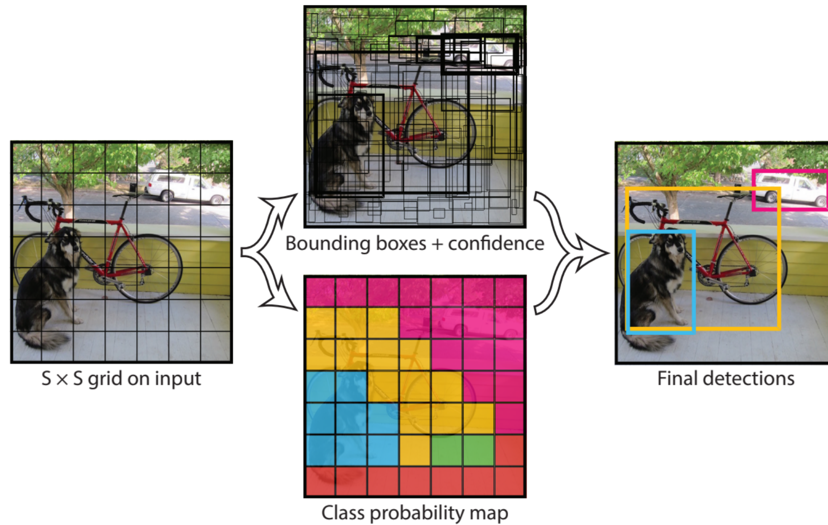

Figure 3.1: A simple visualisation of how YOLO performs object detection as a regression problem [7].

Design of YOLOv1

The model is implemented as a convolutional neural network, inspired by the GoogLeNet model. The network has 24 convolutional layers and 2 fully connected layers. The inception modules used by GoogLeNet are replaced by 1x1 reduction layers, which are followed by 3x3 convolutional layers. The final output is a 7x7x30 tensor of predictions. Separate components of the object detection are unified into a single neural network that uses features of entire images to identify each bounding box. It can do so for all the classes for each bounding box simultaneously. This design enables very fast speeds with high average precision. [7]

Limitations of YOLO

YOLO has some limitations imposed by its grid. Since each grid cell can only predict two bounding boxes and have only one class, it places strong spatial limitations on the number of nearby objects the model can predict. The model struggles with small objects that are in groups, such as small birds. It also struggles to generalise objects in unseen aspect ratios or angles, because it learns to predict bounding boxes from given data. [7]

Comparison to Other Detection Systems

Detection pipelines usually start with extracting a set of robust features from input images. Classifiers or localisers are then used to detect objects in the feature space. These are run in a sliding window method over the whole image or just on some regions on the image. One of the systems using sliding windows approach is DPM. DPM makes use of a disjoint pipeline to extract static features, classify regions, predict bounding boxes for high scoring regions and so on. YOLO replaces all of that with a single convolutional neural network. The network performs all those tasks concurrently. Features are trained in-line and optimized for the detection task. YOLO's architecture leads to a faster and more accurate model than DPM. [7]

R-CNN uses region proposals. Potential bounding boxes are generated with a selective search, a network extracts features, an SVM scores the boxes, a linear model adjusts the bounding boxes and a non-max suppression eliminates duplicate detections. Each stage must be precisely tuned independently, which results in a very slow system, taking more than 40 seconds per image. YOLO has some similarities with R-CNN - each grid proposes potential bounding boxes and scores them. Due to spatial constraints, multiple detections of the same object are eliminated early on. YOLO also suggests fewer bounding boxes per object - 98 versus Selective Search's 2000. [7]

There are some faster detectors than previously mentioned ones. Fast and Faster R-CNN speed up the R-CNN frame by using neural networks to propose regions instead of Selective Search. The improved version is still not fit for real-time performance though. DPM pipeline has also received efforts for speeding it up by using cascades and a GPU, but only 30Hz DPM runs in real-time. Since YOLO is not using a large pipeline, it's fast by design in comparsion to other solutions. [7]

Deep MultiBox uses a convolutional neural network that is trained to predict regions of interest instead of using Selective Search. It can perform single object detection by using single class prediction instead of confidence prediction, however, it can not perform general object detection. Both YOLO and MultiBox use a convolutional network, but while YOLO is a complete detection system, MultiBox

is only part of a bigger pipeline and requires further work. Another system called OverFeat also uses a convolutional neural network, but it is trained to perform a localization and to adapt the localizer to perform a detection. A sliding window detection is used rather efficiently, but it is a disjoint system. It is optimized for localization, not for detection performance. OverFeat cannot properly see the global context, which is why it requires a lot of post-processing for coherent detections. [7]

MultiGrasp is a system that uses grasp detection. YOLO's grid approach to bounding box detection is based on MultiGrasp's system. However, MultiGrasp only predicts a single graspable region for a frame that contains one object - everything else required for detection is left out. YOLO also predicts class probabilities for more than one object for more classes. [7]

Experiments

YOLO was tested against other real-time detection systems on the PASCAL VOC 2007 dataset. Since most of the research was aimed to improve the speed of pipelines, there are not many suitable competitors for actual real-time (30 frames per second or better) detection. 30Hz DPM is one of the detection systems that can run real-time. Another system called Fast YOLO is the fastest object detection method on PASCAL with 52.7% mAP, which is more than twice as accurate as the other solutions on real-time detection. YOLO achieves 63.4% with in real-time performance. The fastest DPM solution only reaches subreal-time performance, missing the mark by the factor of 2. It also has relatively low accuracy. The rest of the solutions do not reach real-time performance. [7]

Generalizability: Person Detection in Artwork

Even when gathering training data from environments of real use cases, not all can be predicted, so real-world test data can differ a lot from training data. YOLO was compared with other systems on the Picasso Dataset and the People-Art Dataset. R-CNN has a high AP on VOC 2007, but it drops a lot when applied to artwork. Selective Search is tuned for natural images, which is why it tends to fail at artwork. DPM performs well on artwork and YOLO works good on VOC 2007 and AP decreases less than on other systems. Since artwork and natural images both share the size and shape of objects, YOLO and DPM have generally not much issue with them. [7]

3.1.2 YOLOv2

Combining Different Types of Datasets

Currently there are different types of datasets, one of them being detection datasets, which are limited in comparison to classification and tagging datasets. Bigger detection datasets contain hundreds of thousands of pictures with hundreds of tags. Classification datasets contain millions of images with hundreds of thousands of categories. The goal is to scale detection to the level of object classification. Labelling images for detection is very expensive, unlike labelling for classification and tagging, where the latter tends to have user-supplied tags for free. For this reason, it is unlikely that detection datasets will reach the same levels as classification datasets any time soon. A new method is proposed that views object classification in a hierarchical manner, allowing it to combine different datasets. A new training is also required to accommodate this. YOLOv2 is the manifestation of this - a real-time object detector capable of detecting more than 9000 object categories. [8]

Better

YOLOv1 has multiple shortcomings, for example its tendency to make more localization errors than Fast R-CNN. YOLOv1 also lacks recall in comparison to region proposal-based methods. As such the improvements are mainly focused on recall and localization. The trend in computer vision moves towards larger and deeper networks. Better performance is sought by training larger networks or combining multiple models. YOLOv2 is taking a different approach to this by simplifying the network and making representations easier to learn. By adding batch normalization to all the convolutional layers in YOLO, mAP is improved by 2%. The batch normalization also eliminates the need for other forms of regularization. [8]

YOLOv1 uses fully connected layers on top of the convolutional feature extractor for coordinate predictions. In YOLOv2, the fully connected layers are removed and anchor boxes are used instead to predict the coordinates of the bounding boxes. One pooling layer is removed, so the output of the convolutional layers is increased. The network is also adjusted to work on a 416 x 416 resolution, down from 448 x 488, because this gives an odd number of locations in the feature map, creating a single center cell. This is good because objects tend to be located at the center of an image, so having a location right in the middle to predict objects is more useful than having four locations nearby. The downsampling occurs by the factor of 32, which outputs a feature map of 13 x 13 with an input image of 416 x 416. Using an anchor gives a slight decrease in accuracy. YOLO predicts 98 boxes per image, however, the anchor box allows prediction of more than a thousand boxes. Without anchor boxes, the intermediate model sits at 69.5 mAP and 81% recall. With anchor boxes the model achieves 69.2 mAP and 88% recall.

The increase in recall indicates room for improvement. [8]

Bringing anchor boxes brings two issues. The box dimensions are hand picked, which means if better dimensions are picked in the beginning, it will be easier for the network to learn better predictions. However, instead of doing it by hand, k-means clustering is run on the training set bounding boxes to automatically find better priors. Good priors are sought because they will eventually lead to good IoU scores and a good trade-off between model complexity and high recall is found at $k = 5$. Higher IoU scores indicate better starting conditions for the model to start learning. The second issue with bounding boxes is the model's instability, especially during early iterations. Most of it comes from predicting the (x,y) locations of the box. YOLO uses the approach where the location coordinates relative to the location of the grid cell are predicted. This will ensure the ground truth will be between 0 and 1. Logistic activation is used to guarantee this range. [8]

In order to improve YOLO even more, a passthrough layer is added, which brings features from a layer before at a 26 x 26 resolution. The passthrough layer puts high resolution features together with low resolution features by stacking adjacent features into different channels. This results in a 13 x 13 x 2048 feature map, which can be concatenated with original features. The detector can use this map to gain access to fine grained features. The result is a 1% performance increase. [8]

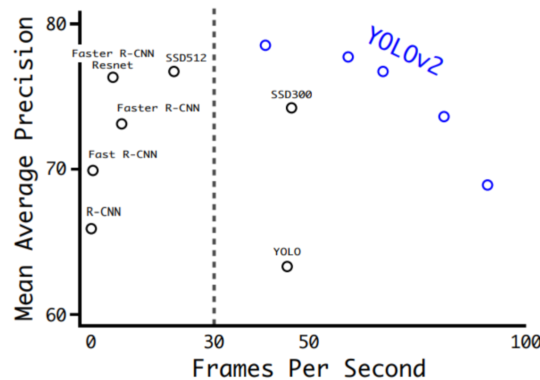

Figure 3.2: Image displays YOLOv2's performance on the VOC 2007 dataset. The dotted line marks the threshold for real-time speed. [8]

YOLO also underwent a flexibility training called multi-scale training. Since YOLO only uses convolutional and pooling layers, it can be resized at will. After every few iterations, the input image size is changed to a new semi-random value (a pool of sizes with delta 32 is predetermined 320, 352,... 608). It does set limits to the smallest and largest sizes, but it delivers the benefits. When the network is resized, training continues. Thanks to the multi-scale training, the network learns to predict well regardless of the input size. At low resolutions YOLOv2 can run very fast with a very good mAP, which makes it good for weak GPUs or high

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Table 3.1: YOLOv2's performance on the VOC 2012 dataset is on par with other detectors, but it is 2-10 times faster [8].

framerate videos or environments where real-time speeds have to be guaranteed. At high resolutions, YOLOv2 is a state-of-the-art detector working at real-time speeds with a mAP of 78.6 on the VOC 2007 dataset (see Fig. 3.2) and on the VOC 2012 dataset YOLOv2 is achieving the same results as other detectors while being a lot faster (see Table 3.1). [8]

Faster

YOLOv2 is designed to be fast from ground up. Being fast is crucial for applications in robotics and self-driving cars, where a lot can matter on a single millisecond. Most detection frameworks use VGG-16, which is a base feature extractor. It is powerful and accurate, but very complex. It requires 30.69 billion floating point operations at single pass on a single image at a 224 x 224 resolution. YOLO uses a custom network based on the GoogLeNet architecture. This uses 8.52 billion floating point operations, but its accuracy is a little lower than VGG-16's. YOLO achieves 88.0% on ImageNet compared to 90.0% on VGG-16. [8]

YOLOv2 uses a new classification model. It uses 3 x 3 filters like VGG and has twice the number of channels after every pooling step. The global average pooling is used to make predictions. Also, 1 x 1 filters are used to compress feature representations between 3 x 3 convolutions. The final model has 19 convolutional layers and 5 maxpooling layers. The new model is called Darknet-19. It requires 5.58 billion operations and achieves 72.9% top-1 and 91.2% top-5 accuracies on ImageNet. [8]

The network is trained for classification on the standard ImageNet 1000 class dataset. After the initial training it is fine tuned at a larger size and additional training is added. At a higher resolution the network reaches 76.5% top-1 accuracy and 93.3% top-5. In addition, the network is modified for detection by replacing the last convolutional layer with three 3 x 3 convolutional layers with 1024 filters and 1 x 1 convlutional layers after each one of them. The network is trained for 160 epochs. [8]

Stonger

YOLOv2 is trained on both detection and classification datasets. Training on both poses a few problems, such as merging the appropriate labels together. For example, detection datasets have only general labels, such as ”dog” or ”cat”, but classification datasets go deeper, by identifying the breed of a dog as well. When a breed of a dog is classified, the fact that it is a dog still holds true, but for networks they are mutually exclusive. A multi-label model is used to combine two different datasets. [8]

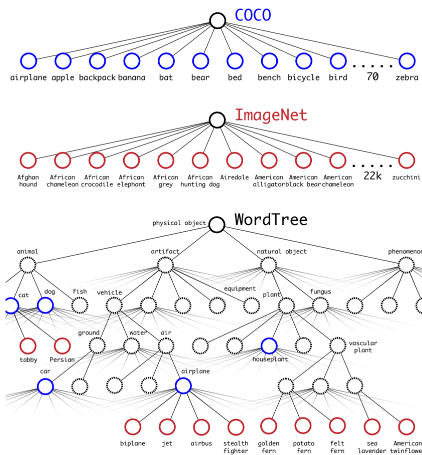

Figure 3.3: A simplified view of the WordTree [8].

In order to accommodate this approach, a hierarchical tree is built based on the concepts of ImageNet and working through the graph of WordNet, which is where ImageNet is deriving its labels from. WordNet is built as a graph, not a tree, which is why a custom solution is required. At first, visual nouns are analysed and their paths in the WordNet graph tracked all the way to their common root node, which in their case is ”physical object”. A lot of objects are linked together via one path only, so these paths are added to the tree first. The remaining objects are checked for the shortest paths, so that they would grow the resulting hierarchical tree (WordTree) as little as possible (see Fig. 3.3). During training, if some image has a label with a specific dog breed, it also gets the general ”dog” label. Using the same training parameters, Darknet-19 achieves a top-1 accuracy of 71.9% and 90.4% top-5. In the case where the network is unable to determine the breed of the dog, it will still output a high confidence for the label ”dog”. WordTree can be combined with other datasets, such as COCO. Thanks to its diversity, most datasets will work. [8]

3.1.3 YOLOv3

Improvements

Improvements in YOLOv3 came mostly from other people. The first improvement was applying bounding box prediction. YOLOv3 uses logistic regression to predict a objectness score for every bounding box. If the bounding box prior overlaps a ground truth object more than other bounding box priors, it will be 1. If the bounding box prior is not the best, the prediction will be ignored. Second improvement applies class predictions. Objects in bounding boxes are predicted as to what it may be using multilabel classification. Independent logistic classifiers are used instead of softmax because the latter is not needed for good performance. It also inhibits uses of domains such as the Open Images Dataset that has overlapping labels. Softmax assumes that each box corresponds to only one class, which might not be the case. The multilabel approach is favored. [9]

Predictions Across Scales

YOLOv3 predicts across 3 different scales. Features are extracted from scales with a concept similar to feature pyramid networks. Several convolutional layers are added, last of which predict a 3D tensor encoding bounding box, class predictions and objectness. A feature map is then taken and upsampled 2 times, which is merged with a feature map from earlier in the network. This gives better semantic information from upscaled features and finer-grained information from earlier maps. [9]

Feature Extractor

A new network was formed by merging YOLOv2, Darknet-19 and the newfangled residual network. The new network is larger, uses successive 3x3 and 1x1 convolutional layers with some shortcut connections. The new network is called Darknet-53, because it has 53 convolutional layers in total. The network is more powerful than its predecessor Darknet-19 and more efficient than ResNet-101 and ResNet-152. Darknet-53 is 1.5x faster than ResNet-101 and 2x faster at similar performance than ResNet-152. Darknet-53 reaches the highest measured floating point operations per second, which means it utilizes the GPU's power more efficiently. [9]

Results

YOLOv3 performs really well and on par with SSD variants, except it is 3x faster. This is in COCOs average mAP metric, where RetinaNet is still a bit further

ahead. According to the mAP metric at IOU= .5, YOLOv3 performs extremely well. YOLOv3 is a strong detector at producing decent boxes for objects, but struggles to get boxes perfectly aligned with objects as the threshold of IOU rises. With new multi-scale predictions, YOLOv3 struggles with medium and large sized objects, in reverse to the results of the first versions where YOLO performed well with large objects, but struggled with smaller objects. Across the AP50 metric, YOLOv3 is faster and better than other detection systems. [9]

3.2 Kalman Filter

3.2.1 Basics of a Kalman filter

A Kalman filter is a prediction filter, which tries to guess the possible locations and vectors of an object based on the data it is given. It performs really well in cases where variables are always changing and where random unseen variables might occur. For example, it would be perfect to track the movements of a flying drone in the air as it rises, taking into consideration the velocity, which can be predicted rather well, and random air movements, which are nearly impossible to predict most of the time. A Kalman filter takes little memory because it only needs to remember the last state and it is very fast, making it a perfect solution for real-time detection systems such as Darknet. [10][11]

3.2.2 Performing Predictions

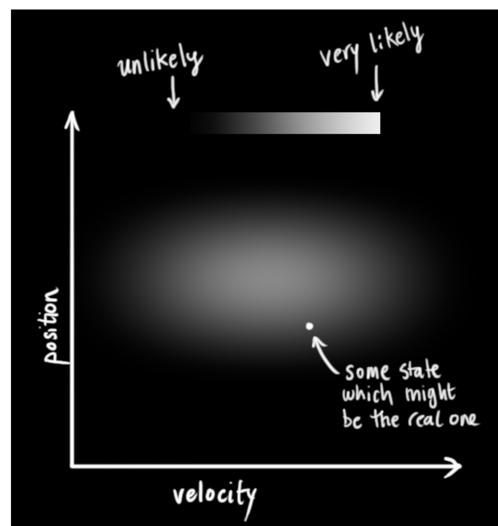

Figure 3.4: Gaussian distribution of possible states x [10].

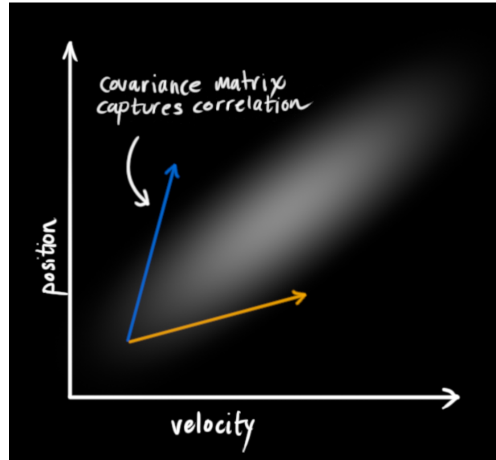

Figure 3.5: Gaussian distribution after taking correlation between p and v into account. Covariance matrix can capture it. [10]

A drone object O has a state x with two attributes - position p and velocity v. At first, these attributes can be unknown. A Kalman filter assumes that both of these attributes can have a Gaussian distribution. They both have a mean value (the actual center of Gaussian distribution, which is also the most likely state, see Fig. 3.4) and an uncertainty marked as variance. A Kalman filter does not know that position p and velocity v are correlated - the faster object O moves, the further it goes, meaning that the next position p_n measured will be further away from the last measured state of p. This correlation is important because it is basic physics and it is important for a Kalman filter to learn this because this will allow to extract more information out of uncertain measurements, giving the possibility of better predictions. This is achieved with a covariance matrix P_k, where k represents the time frame in which attributes were predicted. Each element P_{ij} in the covariance matrix P_k represents a degree of correlation between the i-th state variable and the j-th state variable (see Fig. 3.5). [10][11]

The next step is to take the state x at time $k-1$ and predict the next state x_n at time k. This prediction step is represented with a matrix F_k (see Fig. 3.6). A Kalman filter can also consider additional influences, that might not be constant but can still happen at random. For example, if a drone is observed to be moving, accurate predictions about its velocity and position can be made. However, the drone might accelerate or slow down which would affect the velocity, which in turn would affect the position. The drone might also tilt, causing it to change course, which does not necessarily affect velocity but can affect position. If information about these external forces is available, it can be used and applied to the next prediction as correction. [10]

In addition, a Kalman filter can also take into account influences that are unknown. For example, a drone, that is flying in the air, has velocity and position variables, which can be predicted, but random air movements can throw the drone off course.

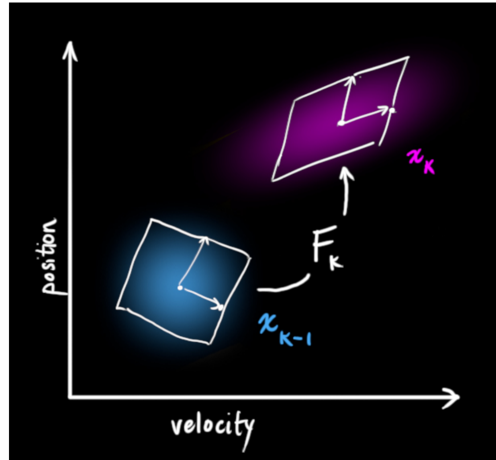

Figure 3.6: Prediction step, where blue gaussian marks possible states of x_{k-1} and pink gaussian marks possible states of x_k [10].

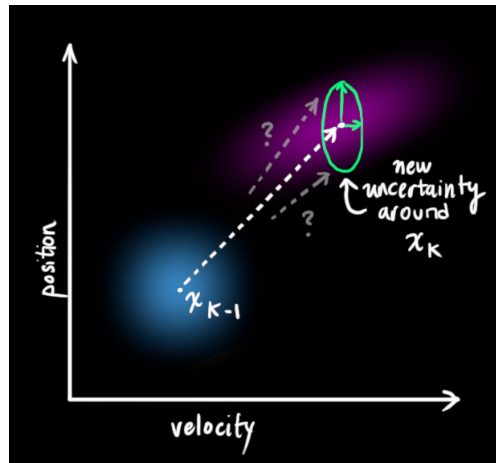

Figure 3.7: Uncertainty Q_k is added to state x_k, giving it wider variety of possible states. [10]

Air movements can not be known ahead, which means that in case they occur, predictions can be off because of them. This can be handled by adding uncertainty with covariance Q_k to every predicted state (see Fig. 3.7 and 3.8). To summarize the entire process of a Kalman filter, the next best predictions are made based on the previous best predictions and then a correction is added to account for known additional influences. New uncertainty is also predicted from old uncertainty, which is later added to the newest corrected predictions with some additional uncertainty from the environment. [10][11]

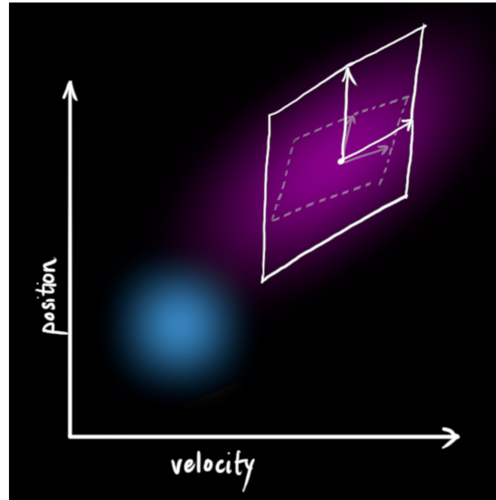

Figure 3.8: Result of states distribution at time k after considering uncertainty Q_k [10].

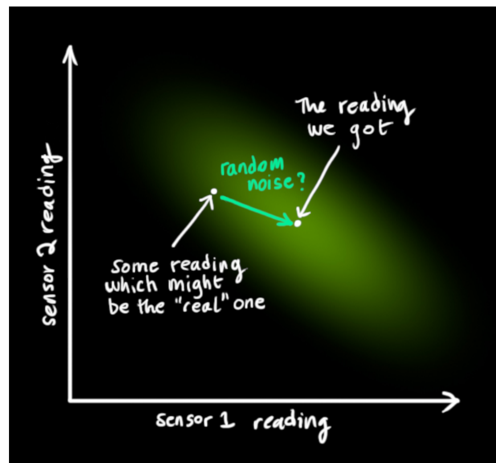

Figure 3.9: Green gaussian distribution marks the states produced by the sensors. Just like with prediction distributions, some states are more likely than others. [10]

3.2.3 Combining Predictions with Measured Data

There can be a random amount of sensors, which can all produce readings about states z_k. The sensors can sometimes be unreliable and produce information that is incorrect, also known as sensor noise (see Fig. 3.9). Sensor readings produce another gaussian distribution which can now be used to calculate the probability of the most likely state by combining the prediction gaussian disribution. This is achieved by multiplying the probabilities of both gaussians, which results in an overlap of the two gaussians. This new gaussian represents a new set of states, that are the best estimates, that can be produced. [10][11]

4. Software Solution

4.1 Darknet

Darknet is an open source neural network framework written in C and CUDA [12]. The original version was developed by J. Redmon, one of the developers of YOLO. The enhanced version of Darknet, developed by AlexeyAB and S. Sinigardi, is used in this thesis. The enhanced version has been adapted to work both on Windows and Linux distributions. In addition, it has detection and training performance improvements, memory allocation improvements, but most importantly, it has an implementation of Kalman filter for object tracking. [13]

4.1.1 Implementation of a Kalman Filter

The Kalman filter is implemented in Darknet. After an object has been detected by YOLO, the detected object is passed to the Kalman filter via a method call. Depending on whether the object has been detected before or not the method varies. This implementation of the Kalman filter checks the location and size of a bounding box. If the Kalman filter sees an object for the first time, it gives it a unique ID but will not release it to Darknet yet. The Kalman filter requires a minimum of three consecutive frames of detection per object before the tracking ID is returned to Darknet, which in this thesis's case happens immediately due to the high frame rate (20 or more per second).

4.2 Module

4.2.1 Pipe Reader

The core of this module is the pipe reader, that reads the output from Darknet. The output from Darknet consists of lines of detected objects and information about them. The pipe reader checks the output for all the detected objects and adds them to an internal list. Since Darknet keeps outputting detected objects

as long as it can detect them, multiple checks are performed to avoid adding duplicate objects. For every detected object, the pipe reader checks if the object already exists in the list. A tracking ID is used to perform this check, because Darknet guarantees a unique ID for every detected object. If the object has not been added to internal list yet, it will be appended now. If the object already exists in the internal list, the object in the internal list will be updated with a new set of location parameters.

4.2.2 Displaying Detected Objects on Video Stream

A separate thread helps to visualise detected objects in an internal list. A stream to the video source is acquired from a camera and is then displayed. Information about detected objects is displayed on the video feed. Detected objects in the internal list are marked with dots. Dots are updated based on the location information of the detected objects in the internal list. Dots are removed, if the object gets removed from the internal list.

4.2.3 Displaying Data in a Table

For the purpose of research and also for future functionality, a separate graphical window is created that displays data about the objects in the internal list in a table. Having output in a console is usually good enough, but for this thesis, it would have been too overwhelming and hard to follow. In its essence, the graphical window is a simple grid based table that shows all the objects in the internal list, including the type of object, tracking ID, location on the video stream, size of the box surrounding the detected object and timer that shows how many seconds have passed since the object was last seen. If the object has not been seen for 6 seconds, it will be removed entirely. 6 seconds was a good trade-off between being sure the object has disappeared out of view and any possible delays there might be with the input from Darknet and thread handler.

4.2.4 Parking Space Handler

The second core of this module directly correlates to one of the goals of this thesis. A separate thread adds new parking spaces upon command and keeps track of all marked parking spaces by checking the state of each and displays it with green (unoccupied) or red (occupied) dots. When the software is first started, it has no data about existing parking spaces. Considering possible deployments, the view to any parking lot can be very different from the view to the parking lot during development and testing. In addition, especially during development, if data about parking spaces is deleted, it must be possible to quickly add parking

spaces again. A solution developed for adding parking spaces should help in most cases where the solution could be deployed and allow to do it quickly.

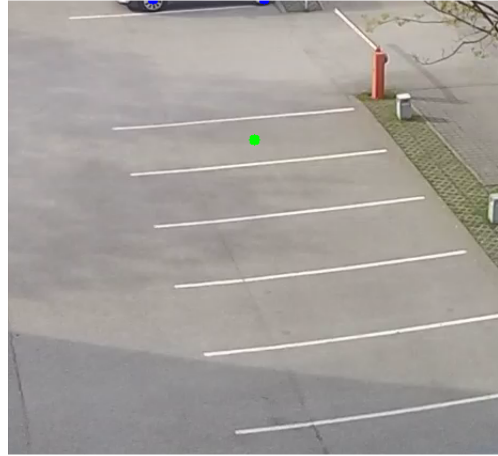

Figure 4.1: An example of a parking space the software is aware of, which is marked with a green circle. Rest of the parking spaces do not exist for the software.

The parking space handler thread keeps its own video stream open in a separate window. This window listens for mouse events, such as movement and left and right button clicks. When a left mouse click is performed, a point with the coordinates of the mouse pointer is stored. After the fourth point is added, all four points are put together and stored in a predetermined file. This way the parking spaces are also available the next time the program is started. When the parking space handler thread starts, it attempts to read parking spaces from a predetermined file. If the file does not exist, no parking spaces will be monitored and the program remains in a clean sheet state. If there is data about parking spaces, they will be stored in the memory and then presented in the video stream window (see Image 4.1). After a parking space is added to the file, the file is read again and any missing parking spaces are stored in the memory.

4.2.5 Parking Space Monitor

This thread checks every 0.5 seconds (time limit is placed to avoid heavy math load), if a parking space is occupied for all marked parking spaces against all detected objects. The method chosen for this task compares the area of a marked parking space, which is in the shape of a quadrilateral, and the area created when the quadrilateral has one more vertex added.

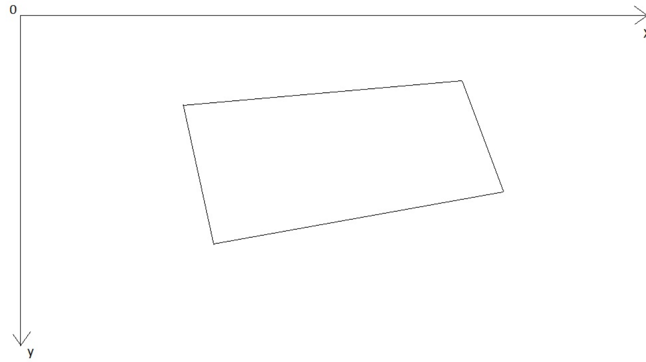

Figure 4.2: The initial state before the special function determines the relative locations of the vertices to each other. An example of a possible quadrilateral is presented with edges between the four vertices with known coordinates. The coordinate axes on this image represent the way they work on video stream windows. All coordinate locations are positive integers.

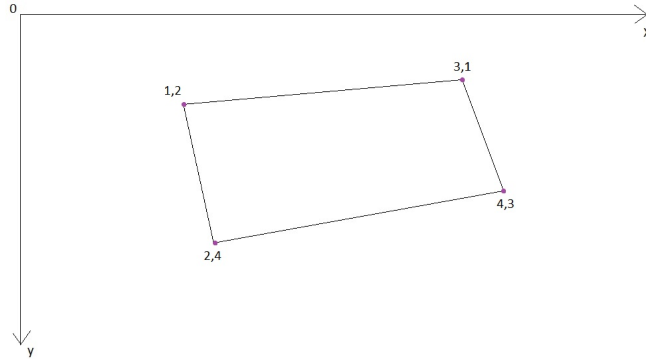

Figure 4.3: The state after ranking the vertices. Ranks are represented in pairs, first one showing the rank of the x-coordinate and second one showing the rank of the y-coordinate.

Determining Relative Locations of Vertices to Each Other

Marked parking spaces have a shape of a quadrilateral. A quadrilateral always has four vertices and if their relative locations to each other is known, the area of the quadrilateral can be found. Since the vertices of a parking space can be added in a random order, the relative locations of the vertices to each other is not known ahead. The relative locations of the vertices to each other are determined in a special function. The function takes four vertices with their coordinates as an argument (initial state shown on Figure 4.2). The coordinates representing values on the x- and y-axis are kept track of separately. In order to determine the relative locations, the function creates a ranking for all x- and y-coordinates - the values with a lower integral number are ranked higher than values with a higher integral number. This ranking guarantees that when the quadrilateral is

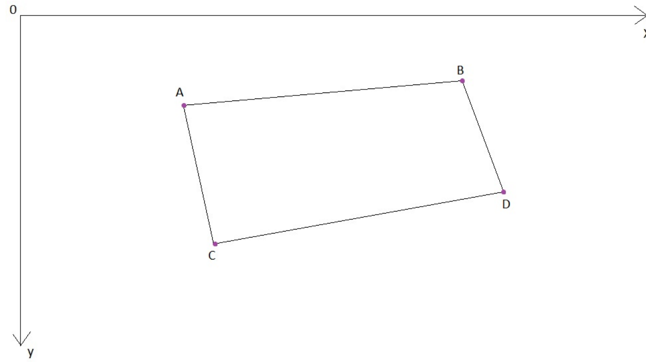

Figure 4.4: The state after the relative locations of vertices to each other have been determined. The letters are always assigned the same way: A - top left, B - top right, C - bottom left, D - bottom right.

approached from the right, the vertices are encountered in the same order as they are in the ranking of x-coordinates (see Fig. 4.3).

The highest ranking x-coordinate is checked first and its according y-coordinate's rank is compared against the second highest x-coordinate's y-coordinate's rank. Depending on the result, the first two vertices are labelled A and C, where A marks the vertix in the top left corner and C marks the vertix in the bottom left corner. The same approach is used on the last two remaining vertices, marking the top right vertix with label B and the bottom right vertix with label D, as shown on image 4.4.

Finding a Quadrilateral's Area

Since the relative locations of vertices are now known, edges can be formed between the vertices appropriately. An extra edge is drawn between points B and C, which splits the quadrilateral into two triangles. The length of these edges is calculated with the following formula:

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.1}$$

To find the area of the quadrilateral, the areas of the two triangles are found. The triangle area is calculated with Heron's formula:

$$s = \frac{L_1 + L_2 + L_3}{2} \tag{4.2}$$

$$area = \sqrt{s \times (s - L_1) \times (s - L_2) \times (s - L_3)} \tag{4.3}$$

Checking Against Detected Objects

The locations of detected objects and the dimensions of the boxes drawn around detected objects are known. The approach developed in this thesis takes a line several pixels above the bottom line of the box. The exact number of pixels depends on the camera angle, camera distance and parking spaces and needs to be fine-tuned. Two vertices (from now on labelled as dots $P1$ and $P2$) are picked on opposite sides on this line several pixels from the end of the line. The exact number of pixels also needs to be fine-tuned. This is necessary because the boxes around detected objects are not fixed and can move slightly from frame to frame even when the object itself is not moving at all. Boxes also usually contain an area slightly bigger than the object itself. By choosing two dots with in the box, we can marginally increase the chance that the dots represent the actual physical presence of a detected object (see Image 4.5).

Figure 4.5: Two marked parking spaces can be seen - one is unoccupied and has a green circle, the other has a car in it and is marked with a red circle. Two blue dots $P1$ and $P2$ (from left to right) are also indicated. These dots are used to determine whether the car is inside the parking space or not.

For this solution, a line 40 pixels above the bottom line and dots $P1$ and $P2$ on opposite sides of that line, both 10 pixels from the end, are chosen. Then the dot $P1$ is added to the quadrilateral representing a parking space. If the quadrilateral remains then $P1$ resides inside it, which in turn means that part of the detected object is occupying the parking space. If adding $P1$ forms a pentagon then $P1$ is not inside the quadrilateral and the detected object is not occupying the parking space. This can be determined by comparing the areas of the quadrilateral and the pentagon.

To calculate the area of the pentagon, the exact same method is used as with the quadrilateral. The lengths of the edges $AP1$, $BP1$, $CP1$ and $DP1$ are found. Combining them with the edges of the quadrilateral, triangles can be formed in a style where one edge is the quadrilateral's edge and two edges are between $P1$ and two of the vertices in the quadrilateral. This results in 4 triangles and the sum of their areas G_1 is compared to the area of the quadrilateral G_2 - if $G_1 > G_2$ then $P1$ is not inside the quadrilateral (see Fig. 4.6). In that case, the same steps are performed with dot $P2$. If both dots are found to be outside of the quadrilateral

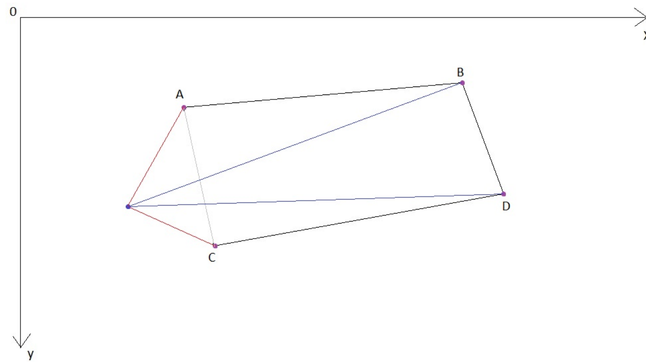

Figure 4.6: A pentagon is formed if $P1$ is outside of the quadrilateral and two new outer edges are drawn between $P1$ and A and C. The edge between A and C is no longer an outer edge.The triangle areas are calculated again and since the result is bigger than the area of the quadrilateral, $P1$ is determined to be outside.

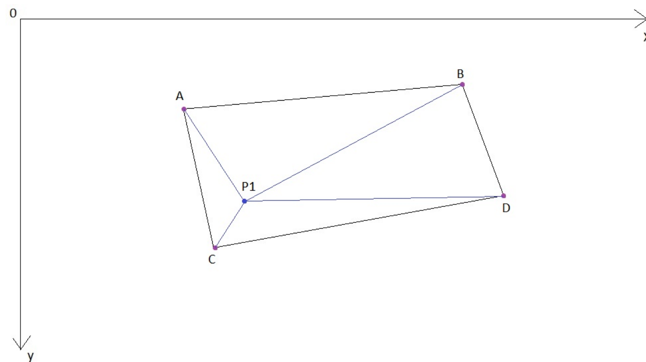

Figure 4.7: This picture shows a situation where $P1$ is inside the quadrilateral. The triangles are formed with $P1$ and their areas are added up. Since the result is not bigger than the area of the quadrilateral, $P1$ is determined to be inside.

then the parking space remains unoccupied, if either of the dots is found to be inside the quadrilateral then the parking space is marked as occupied (see Fig. 4.7).

5. Results

5.1 Efficiency of Parking Space Solution

5.1.1 Adding Parking Spaces

The ability to add parking spaces with desired dimensions and locations offers a flexible way to manage parking spaces that need to be monitored. It is also better than hard coding locations of parking spaces because the code would have to be changed to accommodate any changes, such as camera movements or new parking lots. While the solution presented here has the same issue, manually adding parking spaces on a video stream is much faster and more user friendly than hard coding. While adding, issues can occur when an object is using a parking space, but it is still possible to mark a parking space without seeing the white lines by guessing the estimated locations of them.

Another way to add parking spaces is to use image analysis, which would determine the locations of them independently. This would allow even more flexibility because it could adjust to possible camera movements and learn about new parking spaces at new parking lots with little to no manual input. However, there is no perfect method for learning about parking spaces via image analysis. One possible method would be to analyse the image and search for white lines because white lines are most commonly used for marking parking spaces, with some exceptions. But the problem occurs when there are other white objects, which can distort the results. In addition, if the parking spaces are being used like on image 5.1, not all white lines might be visible, making the whole process more difficult. This method is also sensitive to snow, which can make it impossible to get accurate results. Research on the subject is being done that does offer more options, but getting perfect results is very difficult and a lot of research depends on aerial images or top views [14][15][16][17].

Another method, described by A. Geitgey, is to learn where the detected cars are staying [18]. If a detected car is staying still at some location for an extended period of time (for example 30 minutes) then a presence of a parking space can be safely assumed. In his project, A. Geitgey uses bounding boxes of detected

Figure 5.1: Image analysis searching for white lines would be a lot more complicated in this situation, even if image transformation was to be used.

cars and records the locations as parking spaces. Although this approach can only be used if there are cars present, it is better than white line scanning because parking lots are usually in use, making the presence of cars more likely, which in turn makes the visibility of white lines less likely. However, the method will fail if a car has been parked in a way that does not align with the actual location of a parking space. In this case, the system will learn about the parking space in a wrong manner, which requires additional processing for correction. In addition, if the bounding box of a detected car is not properly around the car, the resulting parking space may be marked smaller or bigger or at an off location than it actually is. With manual marking it is possible to mark the precise location and size of any parking space.

5.1.2 Parking Space Monitoring

Multiple methods for detecting whether a parking space is occupied or unoccupied exist. For example, V. Sialiuk describes a method in his project where he checked for the colour inside marked areas [19]. A baseline asphalt colour was set and a parking space was marked as occupied if a certain percentage of pixels did not match the asphalt colour. It is less complex than the approach used in this thesis, however, it comes with numerous drawbacks. For example, the colour of asphalt is not the same everywhere, so it has to be analysed and changed every time a different parking lot is observed. The colour of asphalt can also change due to weather conditions. In addition, if there is snow on the ground, this approach would not work at all because the asphalt is no longer visible. The area comparison method used in this thesis ignores all those problems, allowing it to work fine with all weather conditions, including snow.

A. Geitgey uses the intersection over union approach, where his solution calculates

the union area of the parking space box and the bounding box of a detected car [18]. If the union area is over a certain threshold, the parking space is marked as occupied. It is less complex than the solution used in this thesis, however, it can make errors when the cars are properly in parking spaces and end up blocking parking spaces behind them. In this thesis intersection over union would not perform well because almost all parking spaces are covered to some extent by cars in front of them from the camera's point of view. A method that can determine, whether a detected car is actually in a parking space that can be seen from the camera, is required. Even if all the parking spaces were in clear view regardless of the amount of cars, the area comparison solution proposed in this thesis would still work, making it more flexible than the intersection over union approach.

The parking space monitoring method used in this thesis has problems as well. It relies heavily on the input it gets from Darknet about the locations of detected objects. Since it is not always sure where exactly the detected objects are, the locations given can change a lot and even slight movements by a few pixels can already affect the monitor's ability to correctly determine whether a parking space is occupied or unoccupied. In addition, if an object that is in a parking space is not detected by YOLO then the parking space will be reported as unoccupied. The stability of the monitoring method also needs further research because during development it sometimes gave false results.

5.2 Problems Encountered and Attempted Solutions

5.2.1 Parking Space is Falsely Observed to be Unoccupied

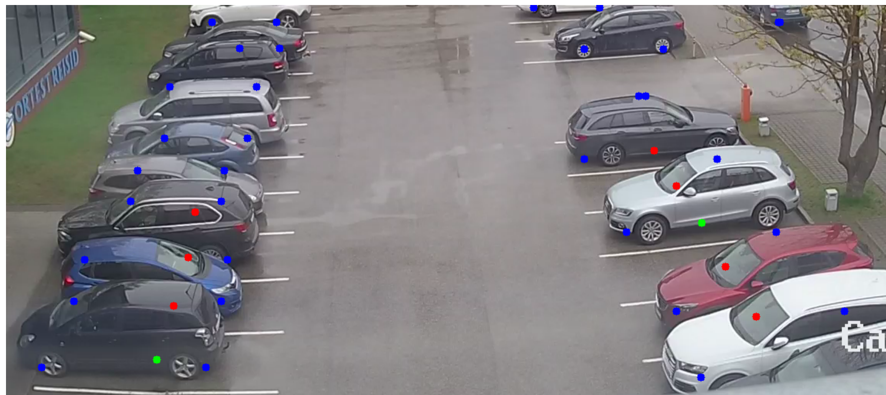

Figure 5.2: Two marked parking spaces are being falsely shown as unoccupied in the lower left corner and mid right (silver Audi), even though the cars are clearly there, including the blue dots $P1$ and $P2$.

Sometimes when a parking space is occupied and is considered occupied by the

software, it quickly changes status to unoccupied for about 10 frames (which is less than a second) and then back to occupied shortly after. This can happen when dots $P1$ and $P2$ are very close to the edge of the parking space quadrilateral and the box marking the detected object moves slightly. However, such behavior has also been observed where the dots clearly stay inside the quadrilateral. The cause for this is unknown. In addition, when a parking space is occupied, the software may report it as unoccupied for longer than 10 seconds even when either of the dots $P1$ or $P2$ can be observed to be clearly inside (see Image 5.2). This has been observed to happen when there are a lot of cars occupying parking spaces, but a clear cause for this behavior is also unknown. A detection threshold has been set to stabilize those occurrences. Parking spaces now have to be considered as occupied for about 60 frames in a row (that is about 2.5 seconds considering the speed between 20..25 frames per second) before they are actually marked as occupied. The same logic is applied when marking parking spaces as unoccupied. This has slightly reduced the issue, where out of 10 marked parking spaces around five were unstable for the entire time the software was working. After the fix, 2 out of 10 marked parking spaces are unstable, changing status between occupied and unoccupied about every 10 seconds. Parking spaces closer to the camera are more stable than the ones further away. This is due to unstable input of the detected objects further away.

5.2.2 New Tracking ID for Already Detected Objects

When objects are detected in Darknet, they are passed through the Kalman filter and given a unique ID that is used for tracking. However, detected objects start getting new IDs randomly at a scale that makes ID based tracking unusable (see Images 5.3, 5.4 and 5.5). This is considered one of the biggest problems because a lot of functionality can only be built upon tracking IDs working properly. It is not always entirely clear why already detected objects are assigned new tracking IDs. While currently there is no solution to this that would remove the issue completely, there are ways that can alleviate the problem because some probable causes have been identified.

Increasing Frame Rate

Increasing the frame rate is one of the best ways to alleviate this issue. Since the frame rate directly correlates to the Kalman filter's performance, providing more frames gives the Kalman filter more material to work with, so it can make better and smoother predictions, thus decreasing the chance of assigning a new ID to a detected object that already has an ID.

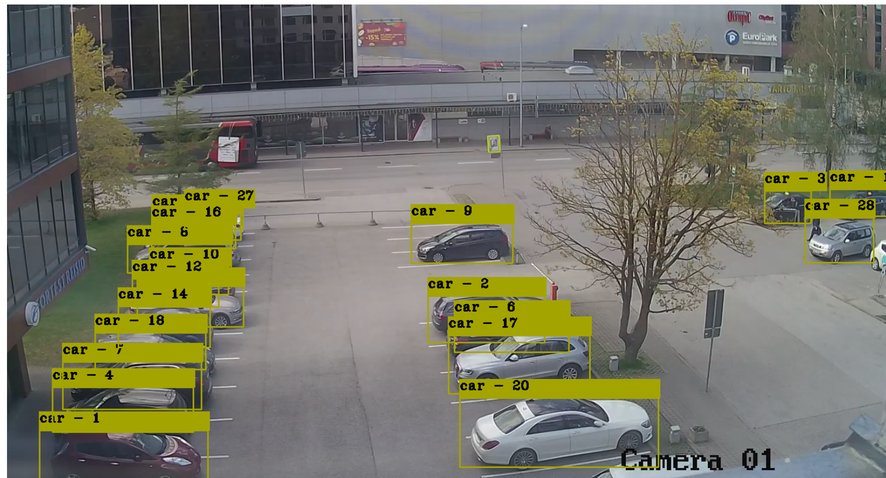

Figure 5.3: This frame snippet was taken a few seconds after the software was started. Several cars can be seen detected and tracked with their unique ID.

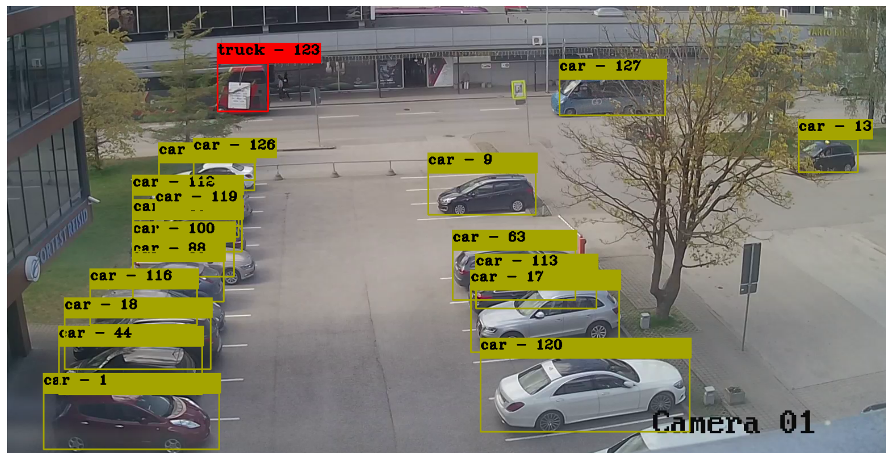

Figure 5.4: This frame snippet was taken about a minute after the software was started. Most cars have already been assigned new IDs despite not having moved.

Finding Optimal Resolution

There are multiple reasons to optimise the resolution of a video coming from a camera. The smaller the resolution the more frames YOLO can analyse per second, increasing performance. However, if the frames are too small, YOLO may start making more mistakes and become less confident about its detections because objects on the video become very stacked up. In addition if the camera does not provide over a certain frame rate then reducing resolution has no effect on it. A bigger resolution provides a clearer picture and YOLO has more data to work with, however, if the resolution is too big, it takes a lot longer to analyse one frame, thus decreasing performance. Finding the optimal resolution, where performance is still real-time, but where YOLO can see objects clearly, is important.

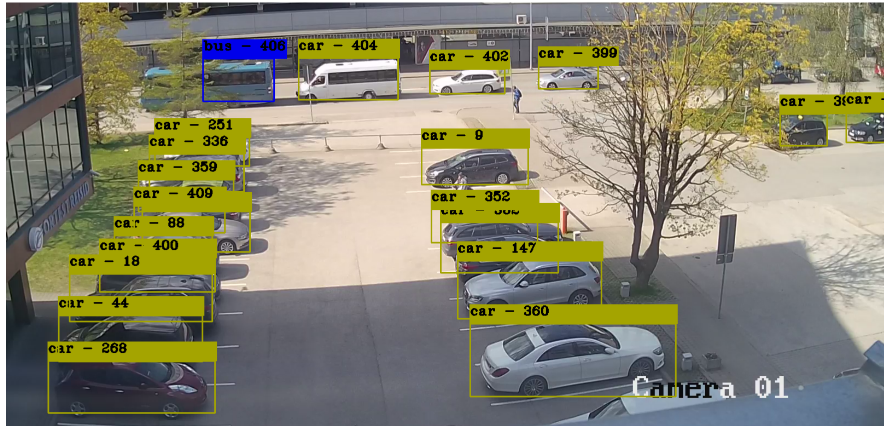

Figure 5.5: This frame was taken about five minutes after the software was started. Almost all cars have received a new ID. The cars that had previously gotten a new ID have received several new ones during those five minutes.

Applying Location Checks in Software

One method to deal with this issue is keep track of detected objects found in one frame. In the next frame, after YOLO detects all the objects it can find, all the new detected objects OBJ_{new} are compared to already known detected objects OBJ_{old}, specifically their locations. If their locations are found to be the same then it is deemed to be the same object and the ID is updated. This method was applied in the module, where a location difference smaller than 30 pixels resulted in updating the ID of OBJ_{old}. However, after applying this method, it became clear that it is not very effective. The reason for this is because situations where OBJ_{old} disappears for YOLO at exactly the same location as it really is, are rather rare (roughly 1/30 of every occurrence, highly depends on the amount of detected objects, their positions and camera angle). This method only works, if the new detection with a new tracking ID for the same object resides with in 30 pixels of the location of the previous detection. The value 30 pixels can be fine-tuned, but there are both upper and bottom limits to it, because at higher values it would no longer make sense or else it starts considering other detected objects that may be close, but are in fact different objects, so they should not be considered at all and at lower values it would be very unsuccessful. Due to its ineffectiveness, the method has been disabled in the module.

Darknet Fails to Detect an Object

This is one of the problems that is most clearly visible and evident. Darknet shows detected objects on a video stream by drawing a box around them and when Darknet no longer detects an object, it will no longer mark it with a box. When Darknet no longer detects an object, the object will loose its tracking ID, because nothing is being passed on to Kalman filter regarding that object. Even

if the object is detected later, it will be treated as an object that has not been seen before and thus will receive a new tracking ID.

5.2.3 Object is Falsely Presented

The top left corner of the video stream is point zero and every pixel on the video stream is with in a positive range, so when an object is detected, its location can never be further than the width and height of a video stream. However, it has been observed during testing that sometimes an object was reported to be at pixel 4294967295 on y-axis. This did not happen often and seemed to occur with objects that were located near the edges of the video stream. A simple check in Darknet has been applied that ignores such impossible occurrences.

5.2.4 Detecting Reflections of Objects

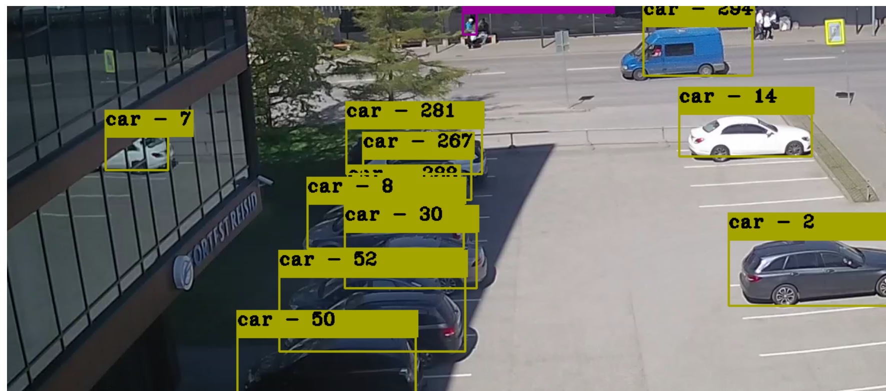

Figure 5.6: This picture shows how reflections of objects may also get detected. Detected object 7 is a reflection of a detected object 14.

In a situation, where there is a building with windows adjacent, it is possible that objects will be reflected with in the camera's view, creating double objects (see Image 5.6). Those reflected objects only serve as interference because a detection of a reflected object provides no usable data to the software. A solution was applied, where the area that gave detections of reflected objects, was marked to be ignored. However, this caused some of the real detected objects to be ignored, because they were sometimes reported to be inside the ignored area. Since the issue happened rarely and was considered not to be too interfering, the fix was removed.

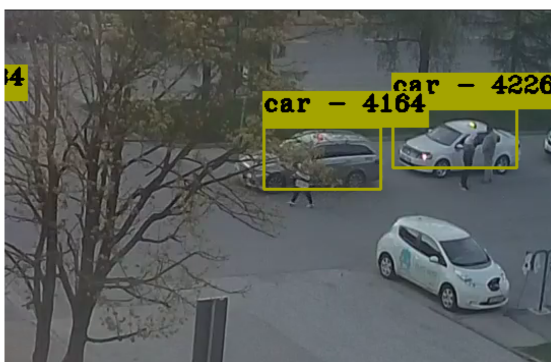

Figure 5.7: There are multiple objects detected on this frame snippet, but the object of interest for this example is the car with the ID 4226. Follow up in image 5.8.

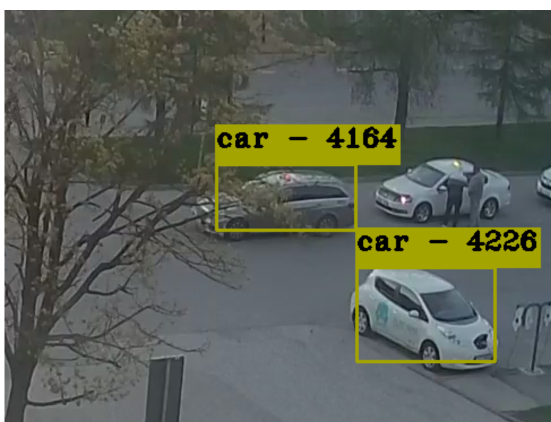

Figure 5.8: This frame snippet shows how a previously undetected car has acquired the ID 4226 from another car. In this case, the ID kept ”jumping” from one car to another multiple times.

5.2.5 Detected Objects Exchange Tracking IDs

An exchange of tracking IDs is an occurrence where at least two objects exchange tracking ID or IDs that had been assigned to them. This can happen in situations where both objects are detected or where one object is detected and the other is not. This tends to happen when objects are stacked very closely together, which makes it harder for YOLO to distinguish them. However, there are also cases where objects are clearly separate, but YOLO is unsure for unknown reasons and reconsiders its detections, as it can be seen on Images 5.7 and 5.8.

5.2.6 Hijacking Tracking ID

This is likely a very specific corner case, but relevant enough for mentioning. In this scenario, a detected object is standing still next to a blind area where YOLO

can not detect objects. An object of similar size and type can sometimes "hijack" a tracking ID if it comes out of the blind area slow enough and for a few frames (around 20-40) covers the previously detected object. This is likely due to YOLO not realising, that a new object has appeared, and if it is moving slow enough, the Kalman filter can fit the movement into its predictions. This is similar to detected objects exchanging tracking IDs, but is caused by objects moving instead of YOLO being unsure.

5.2.7 Occluded Parking Spaces

Figure 5.9: The red arrow marks a black car that is not being detected by YOLO.

If a camera is viewing a parking lot at a lower angle than 90 degrees (like on Image 5.9), then objects further away may be occluded by objects nearer to the camera. This becomes a problem when the software is trying to figure out whether a parking space is occupied or not. Like on image 5.9, the car marked by the red arrow is not being detected by YOLO because it is hidden by the car right in front of it. As a result, it is impossible for the software to know that the parking space is occupied. Theoretically this problem might be alleviated by having more than one camera, but the scope of this solution is beyond this thesis.

5.2.8 Parking violation

There are always some drivers that do not park properly between the white lines of a parking space and end up using space of two or more parking spaces, making all the included parking spaces unusable for other cars (see Image 5.10). With the current method for parking space monitoring, it would not be possible to properly detect these violations. The current method uses two dots to determine whether a car is occupying a parking space - in order to detect violations, several more dots

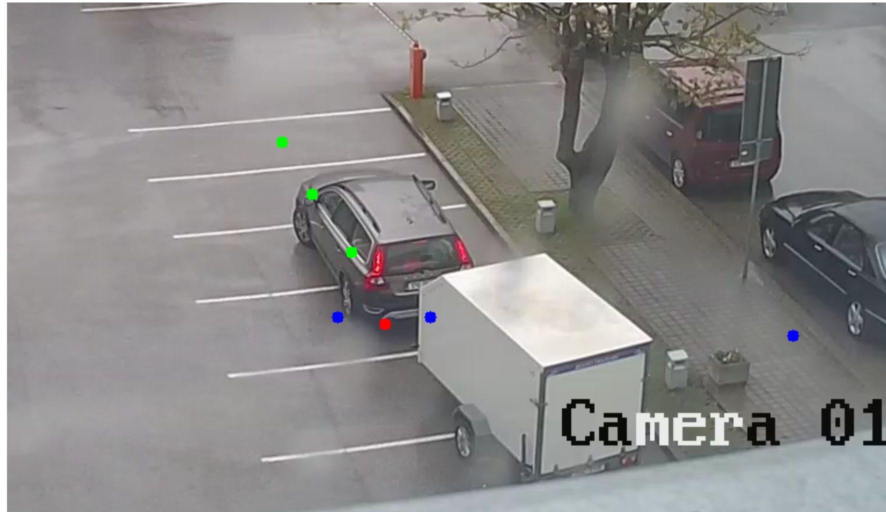

Figure 5.10: Example of parking spaces being used in the not intended way.

should be used. Using more dots would pose additional problems, such as how to avoid false positives. Due to the complexity of this solution and the lack of parking violations (none seen during development), this solution was not attempted.

5.3 Others Problems Not Encountered

5.3.1 Weather Conditions

Weather conditions can have an effect on YOLO's ability to detect objects. Heavy rain, heavy snow fall, fog and so on can hinder the camera's view, giving YOLO an abnormal video feed to work with, from which it might be more difficult to detect objects. In certain conditions this might be alleviated with infrared light, which allows the camera to see more, but the success can vary. If a car happens to be parked in such a way that it reflects sunlight straight into the camera, infrared would not help. Post-processing of a video might help to alleviate in this particular case. Generally not much can be done about weather effects. Only rain was encountered during development and it did not seem to have an effect on YOLO's ability to detect objects.

5.3.2 Unstable Video

Cameras may sometimes produce unstable, shaky videos. This can most likely be caused by the movements of a camera. Unstable video can make it harder for YOLO to detect objects and in turn make it more difficult to check whether a parking space is occupied. Since parking space locations are fixed with pixel

locations on the video feed, moving the camera would displace all the parking spaces. Both hardware and software stabilizers can be used to alleviate this issue. Since no stability issues were encountered during development, no solutions were attempted.

5.3.3 Other Objects in Parking Slots

Instead of cars there may be motorcycles or other types of vehicles occupying a parking space. Since Darknet is trained to detect them, it is not a problem, but might be in case another detector is incapable of detecting such objects. In addition, parking spaces may sometimes be occupied by unusual objects, such as shopping carts and trash cans. Although no such objects were encountered during development, software developed during this thesis could be easily adapted to fix this problem, mainly because Darknet has already been trained to detect those objects. A different detector may, however, struggle with it.

6. Further Possible Improvements

6.1 Implementing a Custom Kalman Filter

In this thesis, a Kalman filter was implemented in Darknet by its developers. Its exact mechanics are not known, because there is no documentation written about it. Since there is no set way to implement a Kalman filter, it is worth trying to develop a more specialised Kalman filter that would work best with this thesis requirements in mind. In order to improve the quality of tracking IDs, additional attributes could be tracked, such as colour and size of detected objects.

6.2 Specialised Training of Darknet

Darknet used in this thesis was pre-trained and since it gave good results for detecting cars and other vehicles, no additional training was done. However, it is worth investigating whether a specialised dataset gathered from parking lots could further improve Darknet's ability to detect cars and other vehicles in conditions that can often be seen in parking lots.

6.3 More Cameras

Additional cameras could provide more information at different angles. With more information it would be possible to make better and more accurate assumptions as to what exactly is going on. For example, if one camera can not see a parking space because it is blocked by a car in front of it, another camera at a different angle might be able to see it clearly. Special cameras could be added, making it possible to scan license plates of entering cars. This in turn would enable additional statistics gathering of cars, such as regularly visiting cars but also blacklisting.

6.4 Additional Functionality

Due to the software's flexibility it is simple to add additional functionality. Most of it can be developed in python, but switching to another programming language could also be considered. Additional functionality could involve a parking lot entry line, which would count entering and leaving cars. This could help to keep track of the number of cars inside a parking lot. It can also help in situations where a car is seen entering a parking lot, but parks at some occluded parking space. If it is known that the car has not exited the parking lot, it can be determined that the car is most likely in some area that is blocked from camera's view. Another useful functionality would be parking space timer, that measures how long a car has been in a parking space. This functionality was already built (see Image 6.1), but due to unstable input it is not reliable.

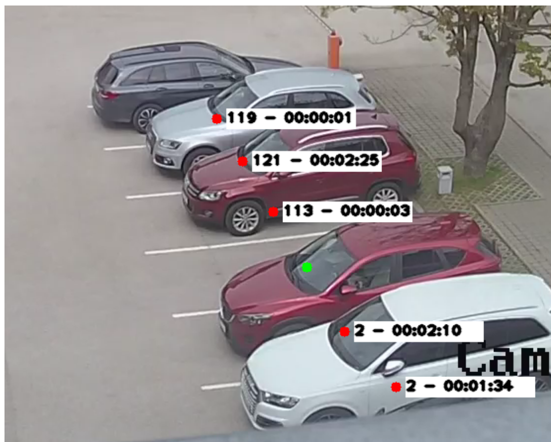

Figure 6.1: Occupied parking spaces display a timer, that shows how long they have been occupied, and the ID of the car in the parking space.

6.5 Capsules Instead of Convolutional Neural Networks

Although still in production, alternatives to convolutional neural networks could be tested, such as capsule networks. The main argument for using capsule networks over convolutional neural networks is their ability to understand the 3D concept of items. While convolutional neural networks need tens of thousands of images of objects in order to be able to recognize them from every angle, capsules only need a fracture of that and can still reach the same level of performance. Initial tests show that using capsules can reduce test errors by 45% compared to CNNs. [20][21]

Conclusion

Neural networks can aid a lot with parking lot management systems and the more research is being conducted on the field, the more sophisticated solutions can be developed. In this thesis an open source neural network named Darknet was used for video analysis provided by a surveillance camera, which was pointed at a parking lot. To process the data from Darknet, a module was developed that took input from Darknet and performed parking space monitoring and ID based tracking. A new method for monitoring parking spaces was developed that to the author's knowledge has not been used before. Kalman filter was used for ID based tracking.

Research conducted in this thesis revealed that although Darknet performs with real-time speeds, the output from Darknet is not stable enough for reliable data processing, because YOLO detection method tends to struggle with localisation. The uncertainty of locations of detected objects can affect the rest of the system along the line. However, when objects were more clearly visible and not stacked together tightly, Darknet was able to provide stable enough data about detected objects.

The method developed for parking space monitoring performed well. If the location of a detected object was given accurately, the method was able to determine whether the object was occupying a parking space or not. This method even worked for parking spaces blocked from the view of the camera. However, the method is sensitive to unstable input of object locations. Additionally, even if the input of object locations was stable, there were some issues with unidentified causes, where a parking space was falsely shown as unoccupied.

Overall, the initial results of the software were promising, but further development is needed to make it a real-life deployable solution.

Acknowledgements

Big thanks to Alexey "AlexeyAB" and S. "cenit" Sinigardi for their help in getting their Darknet solution to work. Without them responding to numerous tickets, this thesis would not be where it is today. Obviously J. Redmon, S. Divvala, R. Girshick, A. Farhadi can not be left out for developing the original YOLOv1 and J. Redmon, A. Farhadi for further improving it, bringing YOLOv3 to life. Thank you, R. Schmid, for proof reading this thesis for grammatical mistakes (despite not knowing anything about the computer vision field) - a lot of articles would be missing if it was not for her keen eye. And last but not least, my supervisors Gholamreza "Shahab" Anbarjafari and Egils Avots for giving me this topic and guiding me through the initial steps.

References

[1] M. Rouse, ”Normal Distribution”, https://whatis.techtarget.com/definition/normal-distribution 09.05.2019, 17:58 (UTC+3 DST)

[2] ParkingDetection, RCE Systems, https://www.parkingdetection.com/ 12.05.2019, 11:50 (UTC+3 DST)

[3] DataFromSky, RCE Systems, http://datafromsky.com/ 12.05.2019, 12:10 (UTC+3 DST)

[4] DataFromSky, ”Advanced Traffic Analysis of Aerial Video Data”, https://www.youtube.com/watch?v=8zXYPHsgfkE 12.05.2019, 12:35 (UTC+3 DST)

[5] A. Khare, ”Automatic Parking Management Video”, https://www.youtube.com/watch?v=y1M5dNkvCJc 02.04.2019, 13:00 (UTC+3 DST)

[6] A. Khare, ”Automatic Parking Management”, https://github.com/ankit1khare/Automatic-Parking-Management 02.04.2019, 13:00 (UTC+3 DST)

[7] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, ”You Only Look Once: Unified, Real-Time Object Detection”, University of Washington, Allen Institute for AI, Facebook AI Research, PDF: https://arxiv.org/pdf/1506.02640.pdf 17.04.2019, 14:30 (UTC+3 DST)

[8] J. Redmon, A. Farhadi, ”YOLO9000: Better, Faster, Stronger”, University of Washington, Allen Institute for AI, PDF: https://arxiv.org/pdf/1612.08242.pdf 19.04.2019, 18:00 (UTC+3 DST)

[9] J. Redmon, A. Farhadi, ”YOLOv3: An Incremental Improvement”, University of Washington, PDF: https://arxiv.org/pdf/1804.02767.pdf 19.04.2019, 18:00 (UTC+3 DST)

[10] Tbabb, Bzarg, ”How a Kalman filter works, in pictures”, https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/ 29.04.2019, 18:00 (UTC+3 DST)

[11] T. McClure, ”How Kalman Filters Work, Part 1”, An Uncommon Lab, http://www.anuncommonlab.com/articles/how-kalman-filters-work/ 02.05.2019, 14:40 (UTC+3 DST)

[12] J. Redmon, ”Darknet: Open Source Neural Networks in C”, https://github.com/pjreddie/darknet 20.11.2018, 18:00 (UTC+2)

[13] AlexeyAB, S. Sinigardi, ”Darknet - Yolo-v3 and Yolo-v2 for Windows and Linux”, https://github.com/AlexeyAB/darknet 15.02.2019, 22:20 (UTC+2)

[14] R. Yusnita, F. Norbaya, N. Basharuddin, ”Intelligent Parking Space Detection System Based on Image Processing”, http://ijimt.org/papers/228-G0038.pdf 09.05.2019, 21:20 (UTC+3 DST)

[15] G. Koutaki, T. Minamoto, K. Uchimura, ”Extraction of Parking Lot Structure From Aerial Image in Urban Areas”, http://www.ijicic.org/ijicic-120202.pdf 09.05.2019, 21:20 (UTC+3 DST)

[16] Y. Seo, C. Urmson, ”A Hierarchical Image Analysis for Extracting Parking Lot Structures from Aerial Images”, https://www.ri.cmu.edu/pub_files/2009/1/tr-09-03-ywseo.pdf 09.05.2019, 21:25 (UTC+3 DST)

[17] X. Wang, A. R. Hanson, ”Parking Lot Analysis and Visualization from Aerial Images”, https://pdfs.semanticscholar.org/712c/bfb8458d41d744b4de753ec8dd6a02cf3c5f.pdf 09.05.2019, 21:32 (UTC+3 DST)

[18] A. Geitgey, ”Snagging Parking Spaces with Mask R-CNN and Python”, https://medium.com/@ageitgey/snagging-parking-spaces-with-mask-r-cnn-and-python-955f2231c400 14.05.2019, 16:10 (UTC+3 DST)

[19] V. Sialiuk, ”Auto Parking Space Detection”, https://www.youtube.com/watch?v=ypAg4PMEtso 20.03.2019, 10:00 (UTC+3 DST)

[20] G. Hinton, S. Sabour, N. Frosst, ”Matrix Capsules with EM Routing”, https://openreview.net/pdf?id=HJWLfGWRb 17.04.2019, 21:40 (UTC+3 DST)

[21] M. Pechyonkin, ”Understanding Hinton’s Capsule Networks. Part I: Intuition”, https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b 17.04.2019, 22:30 (UTC+3 DST)

License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Rasmus Vahtra**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Parking Space Monitoring and ID Based Car Tracking,

   supervised by **Gholamreza Anbarjafari** and **Egils Avots**,

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Rasmus Vahtra
16.05.2019