TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Smart City Solution: AI-based Analysis of Public Parking Spaces

Authors	Başak Erdamar, Katarina Golubovic, Ashish Kumar, Niels Schlegel, Nastassia Vasileuskaya
Mentor(s)	M.Sc. Tanja Pfaffel, Dr. Thorsten Philipp msg systems
Co-Mentor	M.Sc. Philippe Sünnen
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2020

Abstract

The city of Kirchheim aims to improve it's architecture towards a Smart City via AI-based analysis of video camera data. In this paper, the identified features of a smart city include monitoring of parking spaces, creating a heatmap of pedestrians and bicyclists as well as counting and speed detection of vehicles. All detection tasks are computed with a state-of-the-art 'You Only Look Once version 3' (YOLOv3) algorithm. To overcome limitations like overlapping of objects and long-distance detection, the neural network models are retrained with labelled data. Optimized models efficiently detect cars whose surface is not fully visible and bicyclists in the distance. After processing 30 hours of video data, datasets for future analysis have been exported which are visualized and implemented in a web application within a Dash framework. The main observations include: bicyclists drive frequently on pedestrian sideways. Pedestrians cross the street at random spaces. The chance to find a free parking lot is best on Saturday and worst on Thursday. Cars and trucks drive at nearly 15% higher speed on the weekend than on workdays. Based on our conclusions we discuss and propose future improvements in traffic infrastructure for the city of Kirchheim.

Contents

1 Introduction

According to projections, nearly 70% of the world will live in urban areas by 2050 [22]. As a result of the rapid expansion of major cities, an increase in the number of vehicles and the flow of the people, there is an urgent need to create smarter cities in order to make urban areas more liveable and truly sustainable. A smart city is an urban development using Information and Communication Technology (ICT) and the Internet of Things (IoT) to provide useful information to effectively manage resources and assets. This includes data collected from citizens and mechanical devices, that are processed and analyzed to monitor and manage traffic and transport systems, power plants, water supply networks, waste disposal, etc. High-quality surveillance systems, detection of the objects and tracking human activity has become increasingly important for monitoring and analyzing traffic and infrastructure of a city. These technologies are highly used to define a strategy for the development of smart cities.
The main goal of this project is to support Kirchheim on its way to become a Smart City, being an integral part of it's 2020 smart city model and digitisation strategy (see figure 1). Kirchheim bei München is located in Bavaria. It is a district of Munich and approximately 13,000 people are living there.
In order to achieve smart mobility in the city of Kirchheim, this project is covering three main topics: parking analysis, traffic and mobility analysis and integration into a web application. The third chapter briefly explains the used data. The fourth chapter will introduce the architecture of the state-of-art algorithms and their implementation. The following chapters will go more deeply into the implementation of models for each of the previously defined topics. Finally, results from models will be analyzed, visualized and commented.

Figure 1: Camera view of the traffic around the market place of Kirchheim

2 Working Mode

As the scope of the project was not specified in detail, this uncertainty could be best approached via an agile process. The main components within this project included:
- a backlog refinement and prioritization took place at the beginning of the project
- routines of 3 meetings (Mon, Wed, Fri) per week, in which the Kanban board and arising issues were discussed and mitigated
- routines of 1 customer meeting per 3-4 weeks, in which the current prototypes have been presented and feedback and requirements from the customer gathered
With this approach, a customer needs- and problems-focused development could be achieved. For example, the detection of bicyclists, a web application instead of a mobile app and a counting analysis of in and outgoing cars were set to a higher priority during the process due to customer feedback. Especially the coordination of image labeling and merging algorithms remained crucial throughout the project which could be accelerated by means of continuous feedback rounds. An initial brainstorming session delivered several wide-scope features. The first internal prioritization round focused on the analysis of the electric recharging point, an app for predicting the parking places for inhabitants and the walking behavior of pedestrians. The first customer meeting clarified a lot of open questions and the direction of the project.

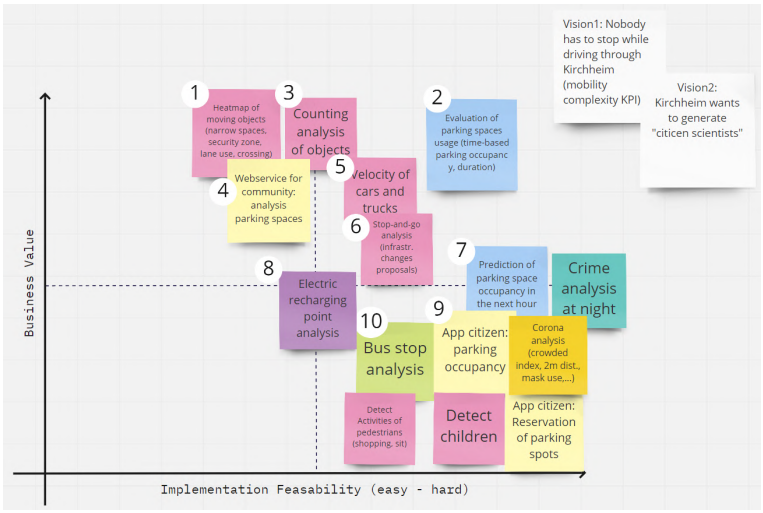


Figure 2: Prioritization of team's ideas after customer feedback

The vision of Kirchheim is that nobody has to stop while driving through Kirchheim. Therefore the infrastructure shall be changed in a way that minimal disturbance of the flow of traffic participants is achieved. Moreover, not only Kirchheim's architects should have access to the ongoing analysis data but also the inhabitants in order to participate in the decision process as "citizen scientists" (quote of Kirchheim representative). During the first customer meeting, the desire was expressed for lesser and slower but still fluent traffic for vehicles in the city center, minimal interference between bicyclists and trucks or cars and changing the infrastructure based on analytical data. Therefore a heatmap, a counting and speed analysis, a parking situation analysis and a web application for data visualization were prioritized which will be described in detail (see figure 2).

3 Data

3.1 Description of the Data

The data used in this project is a collection of video recordings from a surveillance camera. In order to track and analyze the traffic on the street, Kirchheim city administration has installed a camera which recorded traffic movement for around two weeks. Data set covers the period from the 18th of February 2020 till the 5th of March 2020, 24 hours a day. In this period 4595 videos have been obtained. Each video is 5 minutes long. Videos have been delivered to a collaborative company in two different qualities. High-quality videos have a resolution of 3840x2160 with the frame rate 25 frames/second. The size of the high-quality videos is 733GB. On the other hand, low-quality videos have a resolution of 640x360 with the frame rate 5 frames/second. The size of the low-quality videos is 8.7 GB.

Figure 3: High vs. Compressed vs. Low quality

3.2 Data Preprocessing

Due to the unavailability of sufficient computational power and through customer feedback it was decided to analyze the data only for 5 days (24-27th Feb and 29th Feb) in the time slots of (7:00 - 9:00, 12:00 - 14:00, 16:00 - 18:00). To further overcome the hardware limitations videos were compressed from 3840x2160 to 960x540 using HandBrake software [8]. Different solutions discussed in this report for parking, heatmap, counting and speed estimation use different video quality for analysis based on runtime of corresponding algorithms. From here onwards, high quality video will refer to 3840x2160, compressed quality to 960x540 and low quality to 640x360 (figure 3).

4 Methodology and Algorithms

4.1 Algorithms for the Object Detection

Conducting literature survey the team came across some approaches to the problems of intelligent traffic systems based on R-CNN models [4]. Therefore, Mask R-CNN [15] was chosen as the base model but during preliminary analysis, it was realized that the model was taking too long to process the videos with this algorithm. Thus, researching further we finalized on You Only Look Once (YOLO) [21] that is not only faster but equally good

in accuracy also.
R-CNN family based models employ a two-stage algorithm: obtaining region proposals and classification. YOLO is an effective one-stage algorithm that is able to process images at much higher Frames Per Second (FPS). R-CNN and its variants mainly work on the concept of generating a number of region proposals for the input image and processing each proposal through the network, whereas in the case of YOLO, the input image is processed through the model just once and that saves a lot of computation time, achieving 20 FPS for YOLOv3-608 [31]. In terms of accuracy, it is at par with the other state-of-the-art object detection models with the mAP 57.9 for COCO (Common Objects In Context) dataset on YOLOv3-608 [31].

4.2 Metrics Used to Compare Algorithms

To compare the algorithms we performed accuracy analysis with the project specific dataset as displayed in the figures 4 and 5. Performance metric chosen is mean average precision (mAP) which is a standard metric to evaluate the performance of detection models. mAP is the average of the average precision of all the classes. Analysis is done by labeling 130 frames from the videos for the classes belonging to 4 objects namely car, bus, truck, person. Both the models were almost unable to detect bicycles, so it was excluded from the comparison part otherwise it would display the skewed mAP. From the graphs we could easily infer that YOLOv3 outperforms Mask R-CNN not only in mAP (by 11.5%) but also for the AP of each class. Runtime for Mask R-CNN on Tesla K80 GPU for 5 minutes high quality video was 90 minutes and for YOLOv3 it was 70 minutes.

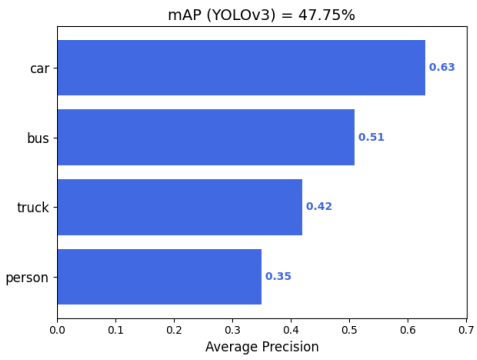


Figure 4: mAP for YOLOv3

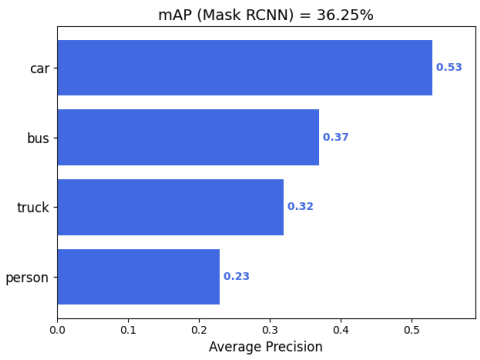


Figure 5: mAP for Mask RCNN

4.3 YOLOv3

YOLO is a single convolutional neural network that simultaneously predicts multiple bounding boxes and class probabilities for those boxes [20]. YOLO algorithm splits the input image into a grid of cells and predicts the bounding boxes and their corresponding class probabilities. In figure 6 after obtaining pre-processed image (reduced to 608 x 608 resolution) from original image, it is being divided into 19x19 grid. For each cell 5 bounding boxes are being predicted that store information in a 85 dimensional vector (assuming model trained on 80 classes).

Boxes with low confidence scores are filtered out using non-maximal supression [11], shown in figure 7. The output, in figure 8, is an array of 6 values giving class probability (p_c), bounding box coordinates (b_x, b_y: midpoint, b_h: height b_w: width) and detected class (c).

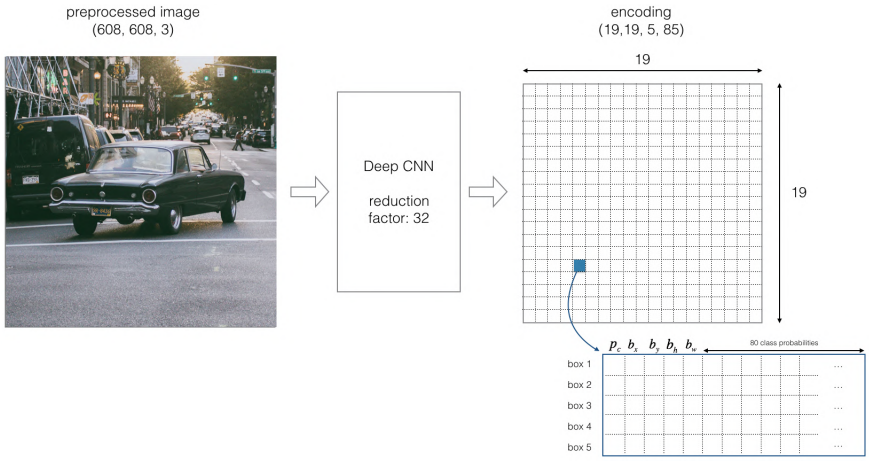


Figure 6: YOLOv3 schematic

Figure 7: Non-maximum supression

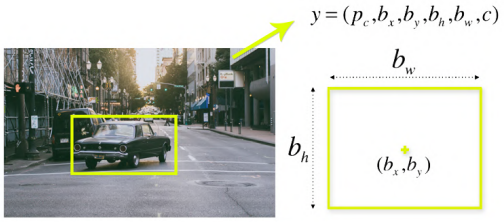


Figure 8: YOLOv3 output

YOLO has 4 different versions with YOLOv4 just released recently (April 2020). We went ahead with YOLOv3 [20] as it has a lot of open source implementations and pretrained weights available on large-scale datasets and it also, already includes a lot of improvements from YOLOv1 and YOLOv2, which performed not so good on small-sized object detection.

4.4 Transfer Learning

Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned [28]. YOLOv3 algorithm is trained with 328,000 images, to classify 80 classes. For developing smart traffic monitoring models we used 6 of them (person, car, bus, truck, motorbike, bicyclist). Though the model is classifying instances rather precisely, the challenging angle of the camera still affects the results of detection. After applying the algorithm to the Kirchheim data, results showed that the bicycles get detected only close-by to camera. Also, most of the time only the persons riding bicycle get detected. For the parking area topic, cars in the back of the street and cars, whose surface is mostly occluded with a bigger car, were not

detected. Thus, the best solution to overcome these challenges was to retrain the model with labelled dataset extracted from videos of Kirchheim city. In order to facilitate the detection for the Neural Network, not only the bicycle detection itself should be improved but instead, the detection of a bicyclist (person + bicycle) should be detected as one class. This would give the network more features to perform the detection. For parked cars, retraining will be focused on improving model so that it is able to detect cars even based on small visible parts of the cars (e.g. roof).
Pre-trained weights for all convolutional layers will be downloaded and used. Weights used in the last Fully Connected (FC) Layer are going to be retrained, due to the fact that they are randomized initially. All convolutional layers have already established a set of inferior features that are sufficient for the detection, while the last layer detects higher and more detailed features like cars, persons and bicyclists. Weights are retrained via 'backpropagation through time', a method based on gradient descent.
The labelling tool used for annotating the training pictures is LabelImg [29]. The tool allows easy generation of the training data in the correct format that the YOLO algorithm requires. The result of the labeling is the directory with images and .txt file (1) with the classes and coordinates for each image. The name of the image and the .txt files must be the same. Additionally, the directory creates a file with a class number and name. For information about configuration files see Retraining of Appendix.

[category number] [object center in X] [object center in Y] [object width in X] [object width in Y] (1)

It is recommended to set the number of iterations as: number of classes times 2000 iterations (e.g. 4000 for 2 classes; 1 iteration = 64 images) [1]. The more images are labelled the less likely overfitting is probable. To generalize well, pictures from a variety of environments are favorable. Thus, pictures at night (cars have the light on) and day, at the rain (different clothing) and sun conditions (shadows) are captured. Also, a considerable amount of pictures (approx. 10%) should be from open source image datasets, so the overfitting problem decreases.
The retraining has been conducted on Google Colab (free version). The pre-trained weights (darknet53.conv.74) are downloaded beforehand. The learned weights are saved intermediately (every 100 iterations) so in case of connection issues, the training process can be started from the last saved version of weights. Depending on the GPU processor allocation (google allocates different GPU with different performance parameters randomly) the training can take at least 6 hours. With the Google Colab Pro version, the more powerful GPUs (Tesla T4) are allocated and the 10 hour limit can be exceeded.

5 Application of Algorithms and Results

5.1 Detection of Parked Cars

Existing parking detection methods can be divided into three types: counter-based, sensor-based and image-based [4]. Counter-based methods are restricted to deployment in gated parking facilities, and they work by counting the number of vehicles that enter and exit the parking facility. Senor-based models usually rely on physical detection sensors that are placed above or below parking lots. Huang et al further divide image-based

methods into car-driven and space-driven methods [12]. Space-driven methods detect changes in highlighted parking lots in an image frame. These models can be partially automatized if the proposed model is able to extract lines, or any different sign, that defines each parking lot. If not possible, it requires the manual task of labeling each parking lot. Besides the comprehensive effort for obtaining and labeling image datasets for each parking lot, space-based methods are not highly scalable, as they require extensive labeling and retraining the models for every distinct parking facility [4]. Car-driven methods rely on detection and the tracking of the car. Car-driven methods quantify the number of occupied parking spaces based on the number of detected cars in the specified area.
Due to the lack of specific remarks of each parking lot and having the camera located in the position from which individual parking lot cannot be seen in every moment (like the camera from above would provide), the car-driven method was the best approach. The main idea is to implement the model that counts the number of occupied parking spaces by combining detection of the parked cars and tracking how many cars have crossed custom-defined parking lines.

Automated image-based models usually detect many difficulties due to environmental variations. While exploring the raw data, following challenges have been observed:

- The quality of the video affects cars detection. Running model on the low-quality videos gives lower accuracy. The model takes input images with lower resolution and as result cars, that are far away from the camera, are not always detected. Running model on high-quality videos gives high accuracy, while the speed performance decreases due to a higher number of frames per second that need to be processed (see figure 10).

- The parking area has to be defined because in this part of the project we want to count only cars in the parking area, that are stable and not on the street. One of the alternative solutions was to detect only stable objects, in order to differentiate between moving cars from the street and stable cars in the parking area. However, the camera view covers many other places with non-moving cars that were not a priority of Kirchheim. Thus, the area of parking space will be defined.

- The angle of the camera records cars parallel. It is impossible to detect cars that cannot be seen on the videos as they are overlapped by bigger cars (see figure 9). Additionally, it is very difficult to see and distinguish between cars on the behind part of the parking area.

- Parking lots are not defined by any remarks or signs. Because of that, the maximum number of parking spaces vary between 18-19. The maximum number is based on the size of the cars that currently occupy the parking area. In this model, we set the maximum number of parking lots to 18.

- Between two parking areas, there is a garage drive-in space. Crossing lines for tracking has been defined accordingly so that the drive-in area is excluded. This way cars that are going to the garage will not be counted as cars that occupied the parking area.

Figure 9: Big cars overlap cars behind

Figure 10: Different video quality

5.1.1 Transfer Learning

Previously explained challenges of the raw data have been solved with a retrained model. The custom dataset used for retraining contains images with six classes of interest - person, car, bus, truck, motorbike, bicyclist. All images, for this purpose, represent the snapshots on different videos. The goal was to have a model that will detect cars also in the situation in which only the small part of the car is visible (e.g. roof). For this task, we have been focusing on the images that present highly critical situations. This refers to a situation in which many cars overlap each other from a viewer perspective. To have a variety of images in the dataset, 25% of night images and images with fewer cars in the parking area were used. In order to have a model that can detect only partially visible objects, the bounding box has to be set accordingly. Every box is set to label only the visible part of the car. Labelling of the cars on the parking spaces represents the labelling of the objects that are not moving. This means that situation of the parking area is not changing in every frame. As a result, we were able to get 209 different images of the parking area. The retraining takes around 20-25 hours (set up: Google Colab, given GPU - Tesla P100-PCIE-16GB, 209 pictures, 6 classes, 12000 iterations). Google Colab provided a maximum GPU runtime of 12 hours. Thus, retraining has to be continued with previously saved weights.

5.1.2 Methodology

Detection of parked cars - For the chosen car-driven method, the first task was to build a model that will detect cars on every frame. YOLOv3 algorithm was used for cars detection with retrained weights (see the chapter Transfer Learning). After applying new weights the accuracy of detection has been increased (see figures 11 and 12). To better understand the goals of Kirchheim, we have discussed with the customer which parking area is the area of interest. Based on given videos two parking areas can be detected. The first one represents the parking area in front of the buildings, that can be used free of charge by anyone. The second area is located in front of the Townhall, and it is intended for car-sharing parking, special parking for disabled people, electrical charging and mayor parking. Analyzing these activities was not of interest to the customer. As a consequence, the built model was only focusing on car detection in the parking area that can be used by all citizens. For running a model, an area that defines the parking area has to be sent as an input parameter. Thus, the first step is defining the region of interest - ROI (see yellow area on the figure 12). The implemented model provides a configuration possibility in which coordinates from the frame can be obtained. After starting the model, the window with loading frames will open. Users can draw the desired area and all chosen coordinates

will be stored in the log file and printed on the console. Coordinates are used for defining the ROI. This can be easily adapted for different locations in the future. After entering, coordinates of the polygon are sent as a parameter to an object detection function. The detection function finds all cars in the frame and returns coordinates of bounding boxes and confidence numbers. The number of parked space is finally calculated by comparing if the center of each box of the car lies in the defined ROI area. If the polygon with defined coordinates contains the coordinates of the center of the box, the car is parked.

Figure 11: Before retraining: 9/11 cars detected on the parking area

Figure 12: After retraining: 11/11 cars detected on the parking area

Tracking of parking process - Multiple testing and observing showed that cars are best detected during performing the action of the parking. Cars are successfully detected while being on the street and driving in or out the parking area. On the contrary, cars could not

be detected while being parked behind the various number of cars. The main reason for that is because some cars were completely overlapped with bigger cars in the front, and angle of camera could not allow viewer to see those cars. This car have been calculated via tracking algorithm. The tracking algorithm used in this project is correlation filter based tracking (CRST) [17]. The CRST algorithm has been integrated into the OpenCV library as a DNN (Deep Neural Networks) module. It is a process of maintaining the vehicle's trajectory in every frame of the video. After the bounding box is detected as an output of the YOLO, the tracker locates the object starting from the initial bounding box. Tracking is used to establish whether the car has crossed the custom-defined parking lines (see two red lines A and B on the figure 12). Intersection of lines and the bounding boxes of the cars, is defined when at least one of the sides of bounding boxes intersect with the custom-defined lines. The intersection of two lines is implemented based on the defined orientation of an ordered triplet of points (p, q, r) in the plane [7]. To prevent that the bounding box of the parked car intersects the line in every frame (e.g. the box of the bigger car overlaps the defined parking line), the bounding box has been reduced to a smaller box, that is $\frac{2}{3}$ of the original size. While tracking we observe which side of the box intersects the line first: if the left side intersects the line, we can conclude the car is leaving the parking lot, if the right side intersects the line - the car is parking. According to this, we update the number of free and occupied parking spaces.

Combining detection and tracking algorithm - Using only one approach, detection or tracking, for sensitive problems like this, will always lead to some inconsistency. Keep in mind that using the only detection can lead to failure under special circumstances like bad weather, fog, object overlapping object of interest. On the other hand, running object detection only on the first frame and tracking subsequently could lead to losing track of one object or moving faster than the tracking algorithm can catch it. To overcome these problems both approaches have been integrated, where detection algorithms run on the following way: the object detection runs on the defined ROI (defined parking area) on the every N frame (number of the frames determines how often do we want to run detection). Between every detection number of parked cars is calculated based on tracking algorithm, which adds and subtracts cars that are entering or leaving the parking area. To prevent future failures of the algorithm, minimum and maximum number of parking spaces has been defined, and number of current occupied spaces cannot go below 0 or be grater than maximum number (in this case 18). Before running the model, we are defining input parameters for functions. These parameters contain information about path to the video, defined parking area, defined parking lines, etc. In the Inputs part of the Appendix you can see some examples and explanation of each parameter. This allows users to adapt this model to a different locations.

5.1.3 Results

Comparison between low quality and high quality data - The model has been run on 30 hours of videos. Because of the required computational power, we have decided to run it on low-quality videos. Local PC with CPU (Intel i7, 16GB RAM) was used. In order to establish how much quality affects results, the algorithm runs on compressed videos on the LRZ infrastructure. Results for low quality, compressed quality and ground

truth are shown in table 1. In the table 1 we take a look only on the cars on the parking space.
For the comparison we need the following:

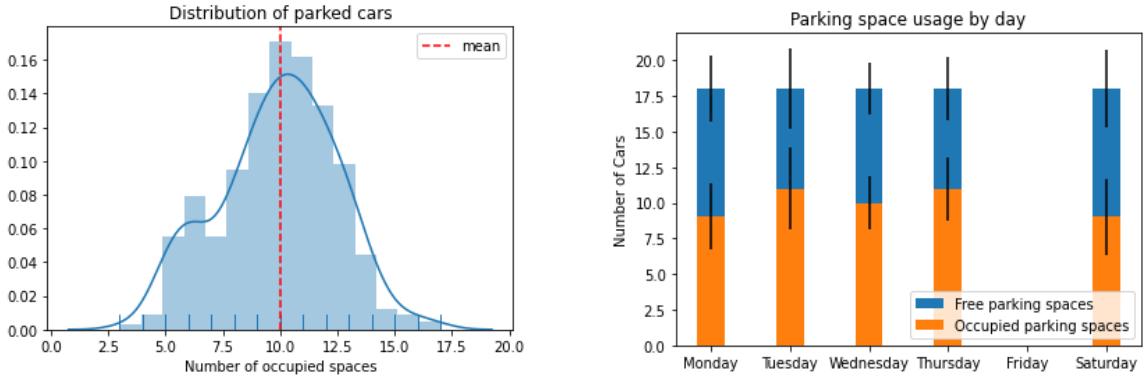
$$\text{Accuracy} = \frac{\text{No of correctly detected cars on the parking space}}{\text{No of ground-truth cars on the parking space}} \tag{2}$$

Video	Minute	Ground-truth	No of detected cars on the low quality	No of detected cars on the high quality	Accuracy on the low quality, %	Accuracy on the high quality, %
2020-02-24	07:13:00	8	8	8	100	100
2020-02-24	08:05:40	11	11	11	100	100
2020-02-24	12:05:30	7	7	7	100	100
2020-02-24	13:06:40	8	7	7	87,5	87,5
2020-02-24	16:05:10	16	13	15	81,3	93,8
2020-02-24	17:11:25	9	8	9	88,9	100

Table 1: High vs. low quality accuracy comparison

From the table 1 it can be concluded that the algorithm on the high quality gives better results. However, running the algorithm on the low quality can be done without losing so much information, if one does not have enough computing power.

Analysis - The output of the algorithm is a .txt file with created data table contains the following columns: date (in YYYY-MM-DD format), time (in HH:MM:SS format), number of free parking spaces, number of occupied parking spaces. An overview distribution of the occupied spaces is shown on the figure 13a. The mean of parking space usage is 9.89 out of 18 parking spaces with a standard deviation of 2.61, suggesting no need for additional parking spaces at the first look. Average parking space usage during each measured weekday is considered in figure 13b. Standard deviations for each measured day are represented with black lines, for better comparison, on the top of the bar representing their respective mean. Initially, we can observe that the parking spaces are more heavily occupied on Tuesday till Thursday compared to Monday and Saturday.
On Thursday a relatively lower standard deviation for occupied spaces are seen, suggesting that Thursday is busier than Tuesday. On Saturday, low mean and high standard deviation is shown, which, combined with the following speed detection results, suggests that our location is not a stopping point on this day of the week. Further analysis results for times of day are displayed in figure 14. On weekdays, there is an increased parking space usage around 8:20. On Tuesday, additionally, an increase between 11:30 and 12:30 can be seen. On Saturday even though mid-day usage is not diverging from the rest of the days, early morning and in the evening until 16:40, parking space usage is visible less than the rest of the measured days. On Monday, Tuesday, Wednesday and Thursday, from the earliest measured hours of the day until around 8:20 but then again a decrease except for Thursday, suggesting that the measured area could be a quick stopping point on early hours.



(a) Parking space usage distribution (b) Daily average parking

Figure 13: Parking spaces analysis

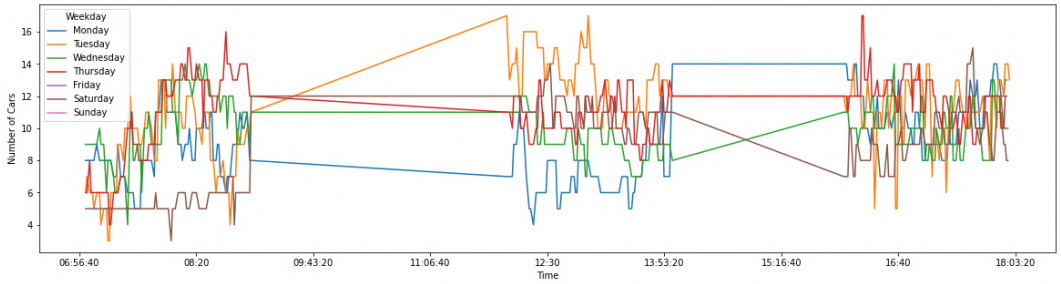


Figure 14: Parking space usage for times of day

Outlook - For the future improvement of the infrastructure of the parking spaces, we suggest extracting the data for a longer time of the period and running the models with higher computation power. This will give a better insight into the parking situation at a different time of the month/year. One could analyse parking occupancy during the corona crisis or holiday months. For the best accuracy, the surveillance camera should be located so that all cars are seen at every moment. This way 960x640 resolution of the video will be sufficient for the detection.

5.2 Traffic and Mobility Analysis

The customer has expressed interest in following topics:

- What is the pedestrian's street crossing pattern?
- Which pathways are usually used by bicyclists and pedestrians?
- How close do bicyclists and cars/ trucks interfere?
- What is the average speed of vehicles? Are there potential stop and go areas?
- How many vehicles pass through Kirchheim on hourly/daily basis?

In order to answer the requested demands, a vision of several features that should be implemented has been created without taking any time restrictions into account. In the

ongoing process, some of the features had to be cancelled due to difficulties with the more prioritized features.

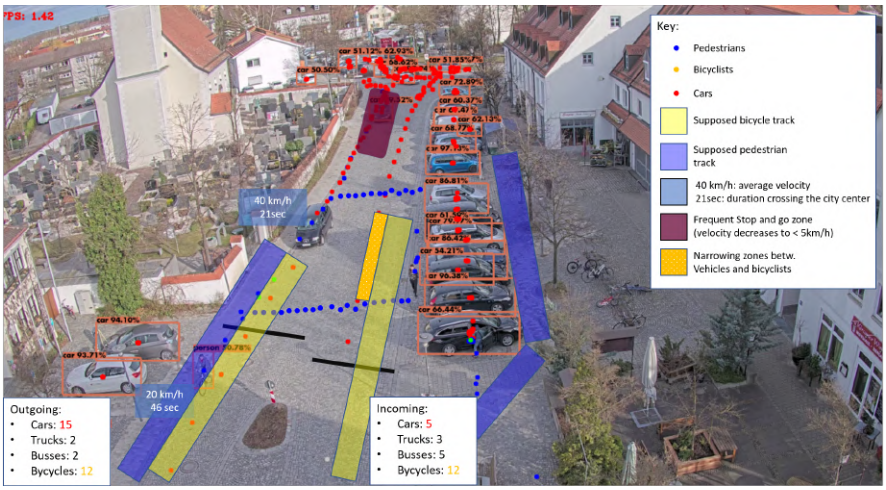


Figure 15: Vision of the implementation of the Traffic and Mobility Analysis Track

Therefore, the following computer vision tasks are necessary to implement:
Object Detection: Detect one or more objects (in our case: cars, trucks, bus, motorbike, bicyclist, pedestrian) and return the position and frame enveloping the object.
Object Tracking: Detecting not only an object but also keep track of it, so at very frame, the algorithm determines whether it is the same object which it has seen in the frame before.
Object Distance and Speed: Using the movement of the object and the corresponding pixel/distance ratio the speed of an object can be inferred.

5.2.1 Transfer Learning

Retraining has been conducted with 750 labelled images with 2 classes (bicyclists and pedestrians). The initial retraining has been not successful because of the little amount of labelled data (ca. 100-200) and too little data variety. The latter problem resulted in pedestrians being labelled as bicyclists once they left the sidewalk and crossed the street as most labelled data consisted of pedestrians on sidewalks and bicyclists on the street. To ensure better generalization capability of the neural network, data of motorcycles, scooters and also bicyclists images of google images have been used in order to retrain the YOLOv3 network. For other computer vision tasks, the open images dataset v6+ with prelabelled images can reduce labeling time but there are mostly high-resolution images of closed-by bicycles available. The training takes about 6-10 hours for a sufficiently converged loss function (setup: Google Colab GPU, 750 images, 2 classes, 4000 iterations) (see figure 17). Applying the new model on a test data set of 33 images resulted in an AP of 47.29 % for bicyclists and 25.15 % for pedestrians (in comparison to 0 % and 15.90 % with the basis YOLOv3 model applied on this test data). This significant improvement can be observed between figure 16 and figure 18.

Figure 16: YOLOv3 basis version

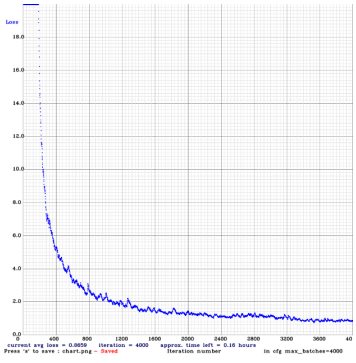


Figure 17: Converging Loss function

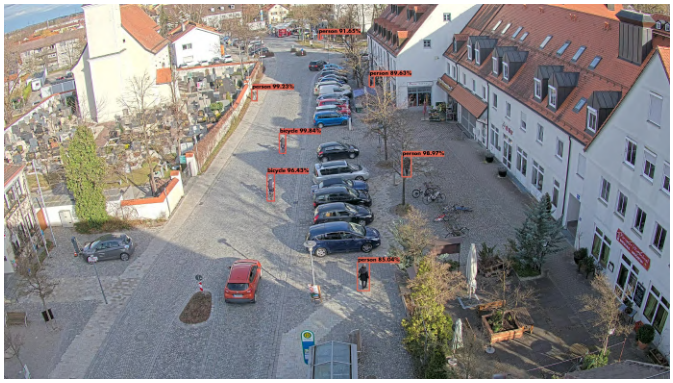


Figure 18: YOLOv3 after retraining for 2 classes

5.2.2 Heatmap

Problem Description:
Pedestrians and bicyclists face unnecessary delays and dangerous road conditions, e.g. pedestrians crossing the street without traffic lights or bicyclists interfering with automobiles in narrow street environments. The heatmap aims to show where the different objects are moving and where supposed and actual tracks are diverging. E.g. a pedestrian is supposed to walk on the crosswalk (ger. "Zebrastreifen") or traffic light and not 5 meters away from the traffic light. It should detect all objects with high confidence for the area in the front until the crossing in the background.

Literature:
Hussein et al. analysed the pedestrian safety at a signalized intersection in New York City by extracting the walking speed and the frequency of pedestrians crossing the street during the "Don't walk" session [13]. He et al. tackled the challenge of foreground and background detection errors by evaluating regions of interest (ROI) via a heatmap and then run a second detection algorithm on these ROI, thus the detection algorithm could concentrate on the foreground areas [10]. Pu et al. analysed the interference of bicyclists and vehicles by determining the vehicle flow in dependence on the density of bicyclists. The results showed that a higher density of bicyclists lead to more passing maneuvers and therefore to a decreased velocity of vehicles [19].

Methodology:
As mentioned before, the YOLOv3 darknet framework offers a well-established and customizable base for detecting the desired 6 object classes. After the DNN has detected the objects in one image frame, the midpoint (x,y-coordinated) of the detection frame is extracted. The coordinates are saved successively with a timestamp in an array (time stamp, x,y-coordinates, object class). If the video is generated or recorded at every frame the current saved points are put on top of the image frame, thus the image is filled continuously. To visualize the data better, the heatmap can be filtered according to class and time (see chapter Application).
As computational limitations arose, only every 25th frame (at HQ-video data every second) appeared to generate sufficient dense point cloud for the heatmap. Therefore, a counter runs up to 25 and resets to 0, at which point the detection algorithms start again. On a local PC with CPU (intel i7 and 16GB RAM) the average duration to process the algorithms takes 1 min per 1 min video (for the retrained model with 2 classes).

Results and Observations:
The point cloud in the background gets less accurate due to detection errors and more dense due to the distance and view angle (more movement per pixel). The points are generated in the middle of the frame which leads to some offset of the points (e.g. on the left side the points get drawn on the wall). For further application, the points could be drawn at the bottom center of the detection frame for more accurate results.

Figure 19: Heat map of detected bicyclists

On figures 19 and 20, the more often points of pedestrians and bicyclists are depicted the warmer the color gets. On the figure 19, it can be seen that the bicyclists are quite often seen in the spaces behind the parking spaces on the right side suggesting that these spaces are frequently used to park bicycles.
From 19, it could be recommended that the city of Kirchheim add more spaces for the bicycles to park, even more surely with the support of more data to come. Secondly, it can be observed that bicyclists tend to ride on the pedestrian sidewalk, especially on the left side of the image. As a result, it could be considered to add an extra bicycle lane.
Moving on to figure 20, the heatmap of pedestrians, it is highlighted in red that the

Figure 20: Heat map of detected pedestrians

pedestrians tend to walk near the wall, away from the street on the left sidewalk. On the right sidewalk, detected pedestrians are somewhat evenly distributed, likely taking advantage of the large sidewalk. From these two information, it can be concluded that a larger sidewalk on the left would be useful for pedestrians. Furthermore, there is no pattern of pedestrians on the street, suggesting they tend to cross the street at any point evenly. To ease the crossing procedure and reduce waiting times, an additional cross walk could be inserted.
A possible further implementation step would be to retrain the YOLOv3 model or the recently released YOLOv4 for 6 classes including cars and busses with the high number of labelled bicyclists. By this, a distance barometer between automobiles and bicyclists can be implemented. Also, to extract the direction of movement of the points in the heat map, a tracking algorithm can be implemented. With this, the number of bicyclist on the wrong side of the street could be counted. In order to get a more precise analysis of tracks occupation, a time related evaluation of density could be applied (e.g. how many pedestrians walk on average within a certain time in a certain area).

5.2.3 Vehicle Counting

Problem Description:
Counting the number of vehicles belonging to different classes like car, bus, truck, etc., can help the city administration to keep a track of the traffic inflow and outflow from the city. It is equally important to be aware of peak hours, days with respect to congestion on roads for proper traffic management and planning to minimize the discomfort and loss in time for the commuters [9, 6]. To provide our customer a tool to tackle this problem, the counting task is performed on specific days and hours as explained in the chapter Description of the Data with the goal to provide some insights into critical time-frames and recommendations based upon that.

Detection and Tracking Algorithms:
Counting task can be broadly subdivided into two components: detection and tracking. Detection deals with identifying and locating instances of objects of a certain class in the

image whereas the goal in tracking is that each target is associated in the consecutive frames of the video, and the same target is assigned the same ID for different frames. For detection, YOLOv3 has been used based on the outcomes of the chapters Algorithms for the Object Detection and Metrics Used to Compare Algorithms.
Some methods for tracking include conventional approaches like Filtering [25], Background Subtraction [23], Optical Flow [24]. For vehicle detection, we already have many models with very high accuracy in recent years, but for vehicle tracking, many algorithms do not focus on the appearance characteristics of the target and some fail to track multi-scale targets.
The tracking algorithm employed here is Simple Online and Real-Time Tracking (SORT) [2] which is a very efficient and fast-tracking algorithm. It computes the intersection over union (IOU) between two frames by detecting the frames at two consecutive moments. Hungarian method [16] and Kalman filter [30], are employed to handle the data association components of the tracking problem and motion prediction respectively, and this approach achieves an accuracy comparable to the state-of-the-art online trackers. The SORT algorithm could achieve the speed of 260Hz that is much faster than other similar tracking algorithms. The state of the target can be determined and associated using its position and each frame's detection box. The SORT algorithm assigns an ID to each target that is identified and tracked successfully.

Methodology:
The counting solution implemented here mainly incorporates three parts: object detector, tracker and counter. First of all, an object detector (in our case YOLOv3) performs detection frame by frame for a given video. Thus, generating an array of bounding boxes and associated confidence scores for each frame. This array is further passed on to the tracking algorithm (SORT) through the tracker instance. Based on the bounding boxes coordinates tracker keeps a track of vehicles in successive frames. Additionally, the detector updates the tracker over the time to make sure the vehicle is still tracked correctly. Finally, the counter counts the vehicles as soon as they cross the virtual counting line drawn across the road as shown in figure 21 and displays the vehicle numbers on top left and top right corners of the output screen. Line intersection is checked with the relative positions of coordinates.

Performance Evaluation:
Performance analysis for the applied method on 1 hour data from randomly selected videos and days is depicted in the table 2. Main observations include the wrong detection of some cars as trucks (5%), on a few occasions the bus goes undetected when it is being observed from the backside, trucks are mostly detected correctly. The tracking algorithm worked exceptionally well during this analysis, counting inaccuracy was only generated due to the limitations of the detector which could be improved by retraining on more images.

5.2.4 Speed Estimation

Problem Description:
High speeding vehicles pose a great risk not only to the driver but also to the surrounding

Vehicle	No. of correct detections	Ground-truth	Accuracy (%)
Car	250	272	92
Bus	19	23	82
Truck	47	50	94

Table 2: Performance of vehicle counting method

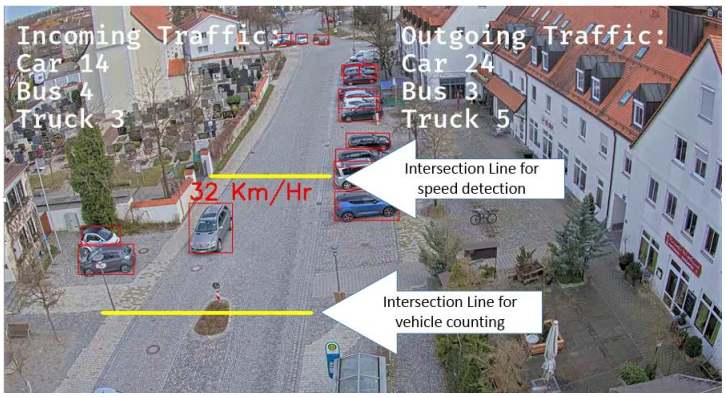


Figure 21: Vehicle counting and speed estimation

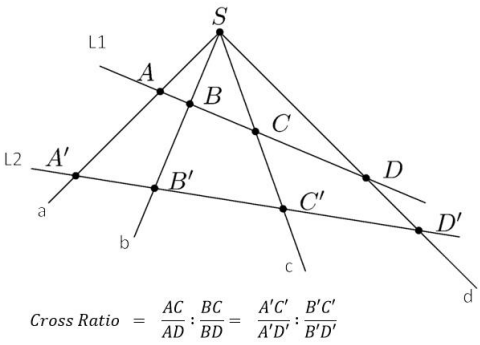


Figure 22: Cross-ratio calculation

vehicles and pedestrians on the road. We should have speed checks where slowing down makes sense especially in the vicinity of children's play area or kindergarten, in a busy street, in front of the premises like shopping complexes, hospitals, etc. Speed estimation could be an effective way of surveillance against the vehicles breaching the prescribed speed limits, as along with the speed of the vehicle its number plate could also be detected and saved in the system log to undertake punitive action against the offenders. A solution is implemented here to perform speed estimation of vehicles driving through the road in Kirchheim city.

Background Study and Approach:
Similar to the counting procedure as discussed above, for the speed estimation also, we need to perform both the tasks of detection and tracking. Additionally, we need an approach to estimate the distance travelled by the detected target vehicle within a specified time frame without losing its track through the intermediate frames of the video. With the stationary background, in image-based analysis this could be accomplished by computing the euclidean distance between the coordinates of the old and current position of the target vehicle. Calculating this distance becomes quite straightforward if the vehicle is close-by to the camera but for the points located far-off from the camera, we may end-up computing false distance due to the concept of perspective projection while creating an image from the points in world coordinates [18]. The properties such as orientation and length, are not invariant under projection. Under perspective projection, the transformation from world to image coordinates is nonlinear, and therefore hard to invert between the two coordinate systems.
One of the approaches to overcome this issue is to backproject the image points into

the world coordinates through camera calibration, which is an estimation of intrinsic and extrinsic camera parameters [14]. Once we obtain the points in world coordinates we can simply apply the euclidean distance formula. Due to the unavailability of information on these parameters another approach is discussed here, known as cross-ratio [3]. Cross-ratio is defined for any four points that are collinear, lying on a straight line and that line should be intersecting 4 different concurrent lines. It is represented as the double ratio of distances between those 4 points. According to the figure 22, if we consider any set of 4 points (A,B,C,D) lying on a straight line (L1) and another set of points (A', B',C', D') on a different line (L2) and L1, L2 lines of both the sets intersecting the same concurrent lines (a,b,c,d), then their cross-ratio is preserved due to the same angles subtended by the two sets of points at the point S.

Methodology:
Detection and tracking algorithms applied here are the same as in the counting task, YOLOv3 and SORT respectively (explained in chapter Vehicle Counting). Implementation of the solution proceeds with the detection of the target vehicle using YOLOv3 and saving the coordinates of the bounding box for this frame. The vehicle is allowed to move forward for the next 10 frames and its position is continuously tracked through SORT. We pre-define 3 collinear points (A', C', D') and (A, C, D) in middle of the lane of road in the image (figure 23) and world (figure 24) coordinate system and compute the relative distances between them. The detected point could be regarded as the 4th point (B') and then, we could apply cross-ratio for these 4 points and the same 4 points in the world coordinates. By equating the two cross-ratios we can compute the unknown distance (BC) travelled by the vehicle between the 10 frames according to the world coordinates. For better accuracy, the midpoint of the detected bounding box is considered as the 4th point for distance measurement. Also, through the frame-rate of the video (FPS), we could determine the time lapsed in those 10 frames. Finally, with the distance-time relationship speed of the vehicle could be reported.

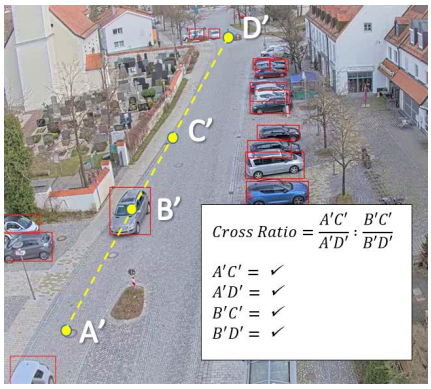


Figure 23: Image cross-ratio

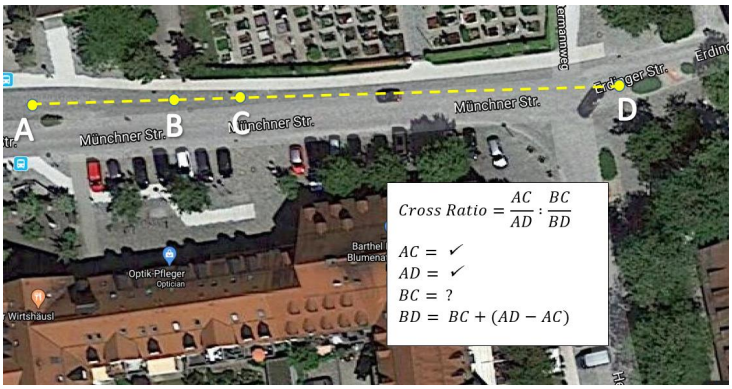


Figure 24: Cross-ratio in real world

Performance Evaluation:
Due to the unavailability of the ground truth data on the speed of the vehicles, validation of the results could not be executed. Although observing the vehicles relative to each other, it could be concluded that the implemented solution is performing the speed de-

tection qualitatively well. Violation of collinearity condition for the cross-ratio on certain occasions when the vehicle is not driving exactly in the middle of the lane might contribute to some inaccuracy in the results.

Analysis and Observations:
In figure 25, the average minute by minute speed for each day is displayed without mak-

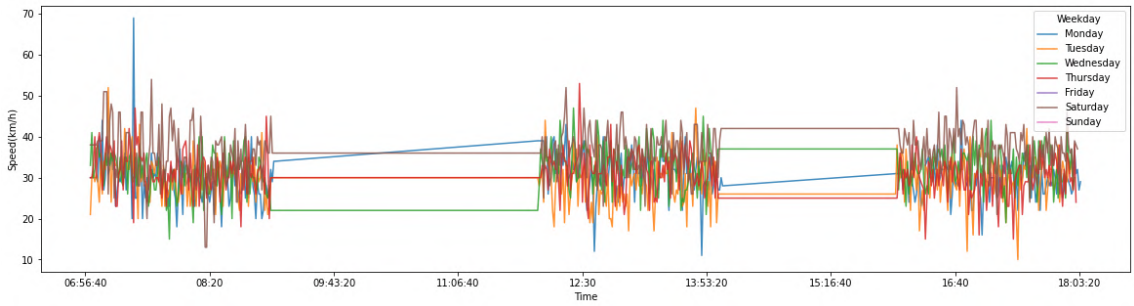


Figure 25: Detected speed by the times of day

ing a distinction in the measured classes; car, truck or bus, in order to get an overview for the days of the week. Taking measured times of day into consideration, no significant deviations are seen for any day. On mornings, there is a tendency to increasing speed starting from early hours and decreasing until 8:20. Sundays, together with Wednesdays a larger speed is seen in the afternoon hours around 16:40. Although, the increase is not as visible on Wednesday as it is on Sunday. This could also explain a lower average of parking space usage displayed on 13b on Sunday and Wednesday, compared to Monday, Tuesday and Thursday. From these two points, it can be deduced that on Saturday (and possibly on Sunday if our team was able to make measurements) the road is used to drive through instead of being a stopping location. This could also be related to the school nearby. To conclude, a consistent driver behavior is observed with no extreme cases for each day and hour. Nevertheless, it is advisable for the city of Kirchheim to make more traffic controls in the early morning hours, especially on Mondays.

On figure 26, daily average speed for each class is displayed. On the first sight, it can be clearly seen that cars and trucks tend to speed more on Saturday than they do from Monday to Thursday. There is a clear increase in truck speeds on Saturday, suggesting possible speed limitations for trucks can be useful. We can see a relative consistency on speed of buses throughout the measured days. However together with each class, busses also tend to speed slightly more on Wednesdays, compared to Monday, Tuesday and Thursday.

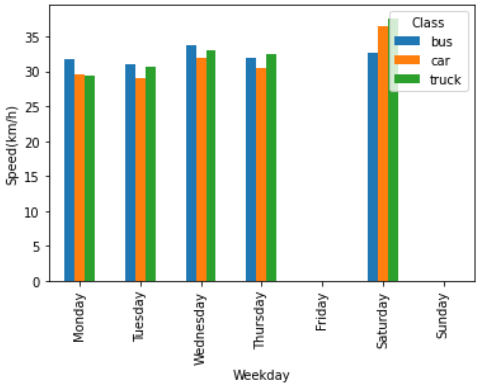


Figure 26: Daily average speed for each class

6 Overview of Outputs and Data Preparation

Three kinds of data from three different sources have been analyzed in the previous sections: parked cars detection, object counting and speed detection.
In the charts showing times of day averages for each day measurements needed to be aligned by time. However, the algorithms make periodic measurements and these measurements are not necessarily always at the same hour and minute of the day for each measured day. Resulting null values were filled with the next valid measurement [26]. Thus, measured times can be compared in the same graph for times of day with continuous line visualisations, which allows to examine the changes in time easily. Time values are stored in pandas timestamps which enabled easily parking dates and grouping by dates, hour and minutes[27].
Due to hardware limitations, our project team was only able to run the algorithms for certain periods of time, on the selected days that are important to investigate. As a result, there were gaps in the graphs when visualised according to the times of day, represented as lines connecting the measured time periods both in the analysis sections and the line charts in the following application section.

7 Application

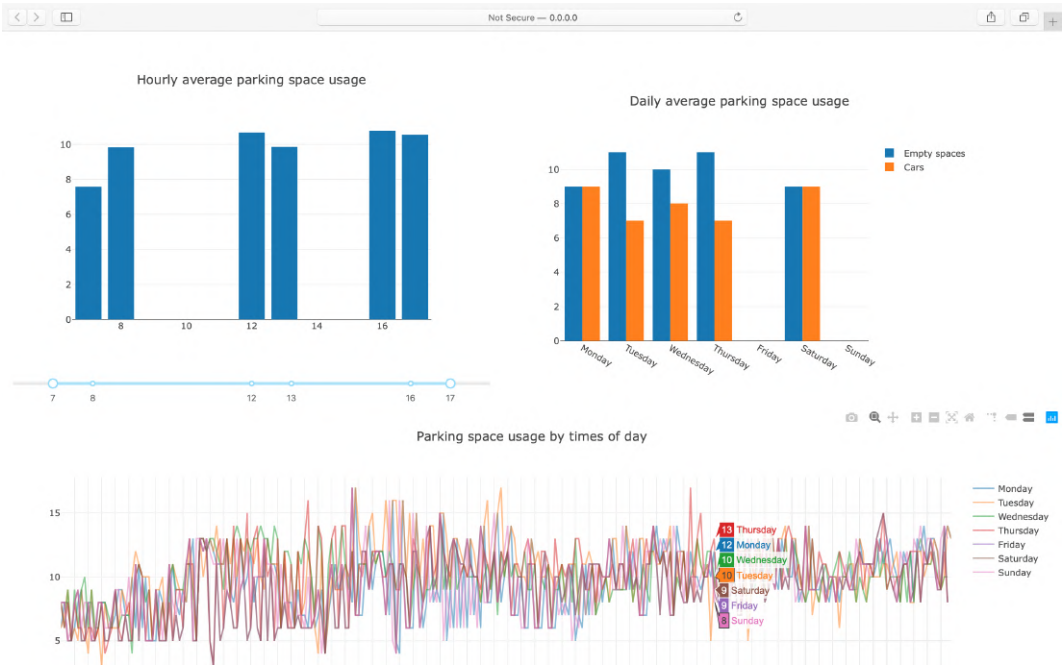


Figure 27: Smart parking analysis screen

7.1 Objective

The need for an application for the city of Kirchheim was initially emerged from the need for an analysis tool for the infrastructure planners to better plan their next steps for the

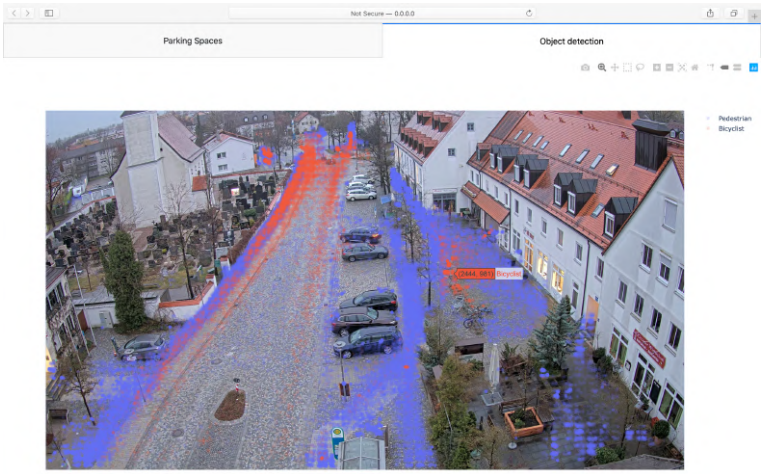


Figure 28: Traffic and mobility analysis screen

city. This idea for an application for planners came with the alternative future vision of a mobile application for inhabitants to be able to check parking space availability, recent traffic information and even more information that could possibly be available in the future. Considering these two goals for the application, two approaches were tested out.

7.2 Methodology

Initially, a React native application was tried all together with a database, an API to transfer the data to the front-end application and a JavaScript based front-end application that uses React library for front-end components which are the buttons, tabs and screens the user interacts with. Using these libraries, it would be possible to compile the application as a mobile application as well as a website with a mobile-friendly interface and three simple screens for home page, smart parking, object counting and speed detection. However, this architecture would possibly require a costly deployment and a relatively complicated setup compared to the following approach. Therefore, according to the feedback from the customer and the advancements in the project, it was more preferable to move on to another framework.
In the end, the Dash framework was used together with a back-end container and other python libraries for a more data-centered approach. In the following sections, a more detailed description of the architecture and the features are provided.

7.3 Framework, Implementation and Architecture

Dash is a python library that enables programmers to use Plotly plots to create applications. The main advantage of this platform was the possibility to use other python libraries such as pandas, which is also the platform the previous analysis is done and the platform containing the dataframes to be visualized.
The application directly imports CSV files and turns them into pandas dataframes, doing all of this in a single script, therefore, enabling easy setup and deployment possibilities.
The application is wrapped in a Docker container so that it is possible to deploy it to popular cloud provider platforms such as Azure, Amazon Web Services and Google Cloud

Figure 29: Traffic and mobility analysis screen

without the need to make any configurations. In the Docker container, all of the python library requirements and file structure for the application are prepared and ready to deploy.

7.4 Usage

The application consists of two screens: the users can switch between the screens using the simple tabs as seen in figures 27 28 and 29. Each screen includes Plotly graphs wrapped in HTML elements designed in accordance to the Dash layout documentation[5].
On the smart parking screen in figure 27 user can explore bar charts displaying average numbers of occupied parking spaces, focus on certain hours of day by using the slider or simply by selecting the interesting part of the graph and zooming in. The zoom in functionality is included in each of the graphs except for the heat map. It is also possible to look at the overall average of the days as well as minute by minute view of the parking space usage. In the minute by minute view, users have the possibility to select the days of interest and filter out the others simply by clicking on the names of the day displayed on the right.
On traffic and mobility analysis screen, on figure 28, users can view the heat map of pedestrians and bicyclists or filter out either one of these classes by simply clicking in the class name on the right. The same screen has further details about the heat map as well. On figure 29, average counts of pedestrians and bicyclists per date are displayed on the upper left graph. It could also be useful for users to zoom into specific dates, for example to analyse the pedestrian traffic on holidays.
On traffic and mobility analysis screen, users can further inspect average measured speed for cars, busses and trucks on each week day. On figure 29, on the upper left graph, users can compare speeds for each class, for instance to acknowledge that the cars are simply

passing by without slowing down on the weekends. Furthermore, users can inspect the measured speeds by times of day, select certain days and zoom into certain hours. An example case for such interest could be for inspecting the speed of vehicles passing by from Monday to Friday when the pupils are released from school, and determine speed limits if necessary.

7.5 Comments, Future Possibilities

It has been the most efficient approach to use the python library Dash to implement this application from a data analyst point of view. For the future, provided the integration with the analysis pipeline, it would be easy to extend the application according to the findings from the further analysis without having to change the data structure. Plotly also provides the interactive graphs with the possibility of filtering on the client-side, which allows the user with the opportunity to get the information he or she needs without having to make additions the application for every single need.

8 Conclusion and outlook

Through this project we have built models for the very specific problems raised by the customer. The solutions developed include heat map of objects, counting and speed detection, smart parking, and integration of the results from each task into a common web application. All these solutions correspond to a very active area of research in the computer vision field i.e., object detection and tracking. These solutions could help to undertake proactive measures for the problems that might be arising due to congestion on the street or parking lot in future. Heat map helps us to judge how different areas of the street are used by different objects or are there any narrow spaces because of which people are taking longer or riskier routes. Speed and counting solutions can keep a check on speed violations or traffic inflow and outflow. Detection of parked cars can assist to decide whether the parking space is sufficient or there is a need to build extra infrastructure.
Regarding future work there is a huge potential for each of the methods proposed to be extended to similar problems such as charging point usage, public transport utilization by observing the crowd at bus-stop and so on. Interestingly, some of the solutions discussed here could be very handy in surveillance and monitoring during the pandemic situation like COVID-19. Possible next steps could include, combining the retrained model for pedestrian detection with the camera calibration approach (for the crowd far-off from camera) to monitor how close people are walking to each other and if the social distancing norms are being violated. Vehicle counting and parking space occupancy could also indicate how strictly lockdown is being followed by keeping a track of the number of vehicles on the street.

Bibliography

[1] AlexeyAB. *Yolo-v4 and Yolo-v3/v2 for Windows and Linux*. URL: https://github.com/AlexeyAB/darknet/. (Accessed: July 2020).

[2] Z. Bewley A.and Ge and L. Ott. "Simple online and realtime tracking". In: *In Proceedings of the IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA*. 2016, pp. 3464–3468.

[3] Peyton Burlingame. "Cross Ratio". In: *Pittsburg State University, Paper and Posters Presentations* (2017).

[4] Bill Yang Cai et al. "Deep learning-based video system for accurate and real-time parking measurement". In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 7693–7701.

[5] Dash. *Application layout*. May 2020. URL: https://dash.plotly.com/layout. (Accessed May 2020).

[6] Sean Fleming. *Traffic congestion cost the US economy nearly $87 billion in 2018*. 2019. URL: https://www.weforum.org/agenda/2019/03/traffic-congestion-cost-the-us-economy-nearly-87-billion-in-2018/. (Accessed: June 2020).

[7] GeeksforGeeks. *How to check if two given line segments intersect?* Feb. 2020. URL: https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/. (Accessed: June 2020).

[8] *HandBrake*. URL: https://handbrake.fr/. (Accessed: July 2020).

[9] Dennis Hartman. *The Effects of Traffic Congestion*. 2017. URL: https://getawaytips.azcentral.com/the-effects-of-traffic-congestion-12304680.html. (Accessed: June 2020).

[10] Miao He et al. "Pedestrian detection with semantic regions of interest". In: *Sensors* 17.11 (2017), p. 2699.

[11] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. *Learning non-maximum suppression*. 2017. arXiv: 1705.02950 [cs.CV].

[12] Ching-Chun Huang and Sheng-Jyh Wang. "A hierarchical bayesian generation framework for vacant parking space detection". In: *IEEE Transactions on Circuits and Systems for Video Technology* 20.12 (2010), pp. 1770–1785.

[13] Mohamed Hussein et al. "Automated pedestrian safety analysis at a signalized intersection in New York City: Automated data extraction for safety diagnosis and behavioral study". In: *Transportation Research Record* 2519.1 (2015), pp. 17–27.

[14] MathWorks Inc. *Camera Clibration*. URL: https://in.mathworks.com/help/vision/ug/camera-calibration.html. (Accessed: July 2020).

[15] He K. et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017).

[16] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Architecture Logistics* 2 (1955), pp. 83–97.

[17] Alan Lukezic et al. “Discriminative correlation filter with channel and spatial reliability”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6309–6318.

[18] Ken Power. *Perspective Projection*. URL: http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/perspective_projection.html. (Accessed: July 2020).

[19] Ziyuan Pu et al. “Evaluating the interference of bicycle traffic on vehicle operation on urban streets with bike lanes”. In: *Journal of advanced transportation* 2017 (2017).

[20] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).

[21] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[22] Hannah Ritchie. *Urbanization*. 2018. URL: https://ourworldindata.org/urbanization. (Accessed: July 2020).

[23] N. Seenouvong et al. “A computer vision based vehicle detection and counting system”. In: *Proceedings of the 2016 8th International Conference on Knowledge and Smart Technology (KST); Chiangmai, Thailand*. 2016, pp. 224–227.

[24] S. Selim, A. Sarikan, and Ozbayoglu Murat. “Anomaly Detection in Vehicle Traffic with Image Processing and Machine Learning”. In: *Procedia Comput. Sci.* 140 (2018), pp. 64–69.

[25] S. Sravan M. et al. “Fast and accurate on-road vehicle detection based on color intensity segregation”. In: *Procedia Comput. Sci.* 133 (2018), pp. 594–603.

[26] the pandas development team. *Filling empty values*. July 2020. URL: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html. (Accessed July 2020).

[27] the pandas development team. *Working with Time Series*. July 2020. URL: https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html. (Accessed July 2020).

[28] Lisa Torrey and Jude Shavlik. “Transfer learning”. In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[29] Tzutalin. *Labelimg*. URL: https://github.com/tzutalin/labelImg. (Accessed: June 2020).

[30] Nie X. “Detection of Grid Voltage Fundamental and Harmonic Components Using Kalman Filter Based on Dynamic Tracking Model”. In: *IEEE Trans. Ind. Electron* 67 (2019), pp. 1191–1200.

[31] *YOLOv3 website*. URL: https://pjreddie.com/darknet/yolo/. (Accessed: July 2020).

Appendix

Retraining

Retraining the model, regarding only specified classes, requires a configuration of certain files that carry information about how and what to train. Following files are changed:

- cfg/obj.data - contains the information on how many classes are trained on, what is the train and validation set, and what file contains names of the classes of interest.
- cfg/obj.names - has a list of all categories of interests.
- cfg/yolov3.cfg - the file has information about the architecture of the Neural Network. Changing the number of classes of interest affects the architecture of the Neural Network and attributes have to be updated accordingly. The number of batches = 64 - means that for every training step 64 images are used. Subdivision = 16 the batch will be divided by 16. Classes should be set on the number of classes on which the model is trained. Filters - number of filters is calculated as filters = (number of classes + 1 + 4) * 3.

Inputs

VIDEO(String)	Path to the video
DROI(Array)	Coordinates of the region of the detection
USE DROI(bool)	True for detection on the DROI, otherwise detection on the whole frame
DROI PARKING(Array)	Coordinates of the parking space
COUNTING LINES(Aray)	The coordinates of the parking lines
DI(int)	Detection interval, means detection on every DI-frame
DETECTOR(String)	"yolo"
TRACKER(String)	"csrt"
MCDF(int)	Maximum consecutive detection failures
MCTF(int)	Maximum consecutive tracking failures
RECORD(bool)	True, for video as output

Table 3: Input parameters for .env file