Bachelor Project

Czech
Technical
University
in Prague

F8

Faculty of Information Technology
Department of Applied Mathematics

Autonomous Car Chasing

Pavel Jahoda

Supervisor: Ing. Jan Čech, Ph.D.
Field of study: Knowledge Engineering
May 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Car chasing simulation in Carla
Student:	Pavel Jahoda
Supervisor:	Ing. Jan Čech, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

*Get familiar with the literature on
a) physics simulation systems for autonomous driving,
b) autonomous driving for other vehicles (or trajectory) tracking,
c) 3d detection of objects in a vehicle surrounding.
*In Carla, create a scenario with two vehicles: followee, follower. The followee will be driven by an automatic agent on a pre-programmed trajectory. The follower will have an on-board stereo-camera.
*Design and test an algorithm controlling the follower to track the other vehicle and keep the predefined distance as close as possible. The input will be
a) 3D position of the followee,
b) detection of the followee in the camera image,
c) raw camera image.
In cases (a-b) evaluate the algorithm with both exact and perturbed data.
*(Optional). Test the algorithm on the RC platform developed in the department of cybernetics FEE. The followee (another manually driven RC car) can be equipped with a special target to ease the perception part of the problem.

References

1. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun. {CARLA}: {An} Open Urban Driving Simulator. Proc. Annual Conference on Robot Learning, 2017.
2. Hu et al. Joint Monocular 3D Vehicle Detection and Tracking. In ICCV, 2019.
3. L. Novák, Vehicle Detection and Pose Estimation for Autonomous Driving. Master's thesis, FEE CTU in Prague, 2017.
4. F. Jan, Trajectory Tracking for Autonomous Vehicles, Master's thesis, FEE CTU in Prague, 2018.

Ing. Karel Klouda, Ph.D.	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Head of Department	Dean

Prague January 14, 2020

Acknowledgements

I would like to express my deep gratitude to my supervisor Assistant Professor Ing. Jan Čech, Ph.D. for his patient guidance and willingness to devote his time to this work. I would also like to thank ToMi team (Michal Bahník, Dominik Filyo, Martin Vlašimský, and others) who have assembled the subscale vehicle platform on a 1:5 RC car used as the chasing car. Furthermore, I would like to thank doc. Ing. Martin Hromčík, Ph.D. for lending me his 1:10 RC car that was used as the chased car.

Finally, I would like to extend my thanks to my parents for their support throughout my education and to my girlfriend Vanda for her support and for helping me film a promotional video of the autonomous car chase.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location, and quantity.

Abstract

A car chase is a scenario that requires dynamic maneuvers, fast reactions, and strategic trajectory planning. It tests the limits of an autonomous driving system.

We developed an autonomous driving system that can chase another vehicle using only images from a single RGB camera. At the core of the system is a novel dual-task convolutional neural network capable of simultaneously performing object detection as well as coarse semantic segmentation. The system has been firstly tested in CARLA simulation. We created a new challenging publicly available chasing dataset collected by manually driving the chased car. Using the dataset, we showed that the system benefits from using coarse semantic segmentation. The system that uses the semantic segmentation was able to chase the pursued car on average 10% longer than other versions of the system.

Finally, we integrated the system into a subscale vehicle platform built on a high-speed RC car and demonstrated its capabilities by autonomously chasing another RC car.

Keywords: self-driving car, RC car, multi-task learning, autonomous driving, deep learning, CARLA, simulation

Supervisor: Ing. Jan Čech, Ph.D.
Czech Technical University in Prague
Faculty of Electrical Engineering
Center for Machine Perception

Abstrakt

Pronásledování auta je situace, která vyžaduje dynamické manévry, rychlé reakce a strategické plánování trajektorie. Zároveň nám umožňuje testovat limity autonomního řízení.

Vytvořili jsme autonomní řídící systém, který dokáže pronásledovat další vozidlo pouze na základě RGB obrazu z jedné kamery. V jádru tohoto systému je nová víceúlohová konvoluční neuronová síť, která dokáže simultánně detekovat objekty a hrubě sémanticky segmentovat obraz. Tento systém byl nejprve otestován v simulátoru CARLA. Pro tyto účely jsme vytvořili nový veřejně dostupný pronásledovací dataset. Dataset byl vytvořen manuální jízdou pronásledovaného vozidla. Za použití tohoto datasetu jsme ukázali, že systém funguje lépe když využívá hrubou sémantickou segmentaci. Takový systém dokázal pronásledovat druhé vozidlo v průměru o 10% déle než verze systému které sémantické segmentace nevyužívají.

Na závěr byl tento systém integrován do platformy nazvané „subscale vehicle platform“ připevněné na velmi rychlé RC auto na kterém byl systém testován pronásledováním dalšího RC vozidla.

Klíčová slova: samořídíci auto, RC auto, víceúlohové učení, autonomní řízení, hluboké učení, CARLA, simulace

Contents

Figures

Tables

Chapter 1
Introduction

An autonomous car driving system capable of making fast and accurate decisions is important for making car transportation a safer activity. The reaction times of the system (especially in high speeds) have an impact on the human trust and acceptance of automated vehicles. The key aspects of driving autonomously include an understanding of the surrounding environment, detecting other cars, and interacting with them on the road. In this paper, we present a novel neural architecture that gives information about other cars as well as the surrounding drivable surface. We then test an autonomous driving system, that we developed, in these aspects by using a car chasing scenario.

A similar scenario – car-following – has been studied for more than half a century. Car-following models describe how vehicles should follow each other in a traffic stream. Frequently these theoretical models assume having precise data such as speed, distance, and acceleration at every timestamp [1]. However, with the advancements of machine learning and computer vision more practical car-following models have been developed and tested. These models use sensors such as LIDAR and camera to estimate the distance between the two cars [2]. The estimated information is then used to maintain a safe distance between the two cars while following the trajectory of the front car. However, the car-following models have typically not been tested in scenarios involving high speeds and sudden fast acceleration and deceleration.

On the other hand, a car chase – a vehicular hot pursuit of suspects by law enforcers – typically involves high speeds and therefore dynamic driving maneuvers are necessary. In our scenario, a vehicle being pursued is driven by a person, while the vehicle that is chasing it is being controlled by an artificial intelligence-based system. Solving an autonomous car chasing problem requires a system that solves many sub-problems from computer vision, planning, and control theory. The system is an integration of the solutions to these sub-problems. The architecture of our system is similar to the DARPA Urban Challenge vehicles [3, 4, 5] and consists of three parts: perception and localization, trajectory planning, and a trajectory controller.

The thesis has the following structure. First, a theoretical background of the chasing algorithm is outlined. This includes related work described in Section 2 and also a description of each part of the system in Section 3. Then, an experiment Section 4 follows. The section includes evaluation in CARLA – an open-source simulator for autonomous driving research [6]. Additionally, it has real-world testing of the system deployed in a radio-controlled (RC) car. The real-world evaluation consists of two parts. First, it has an evaluation of the novel neural network's ability to detect objects and segment images. This is done using a collected dataset that will be presented in the experiment section. Second, it describes a live test chase in which the goal is to maintain a determined distance between two RC cars. However, unlike in the car following problem, following the same trajectory is not required. Finally, Section 5 concludes the thesis.

Chapter 2
Related Work

2.1 Path Following

A path is understood as a set of waypoints $w_0, \dots, w_n$. Given a path, a moving vehicle with an initial position $p = w_0$ and an initial vehicle heading angle ψ. A path-following algorithm tries to change ψ at each timestamp $i \in \langle 0, n \rangle$ to minimize the sum of cross-track errors. The cross-track error is a distance between the vehicle and the closest point on the path. Furthermore, it attempts to minimize the differences between the heading angles ψ_i and the angle formed by waypoints w_i and w_{i+1} called the LOS angle ϕ [7].

In 1985, Carnegie Mellon University [8] demonstrated the first autonomous path-following vehicle. It works by finding a look-ahead position on the path and subsequently calculating the vehicle's heading angle ψ as a curvature κ of an arc as depicted in Figure 2.1. Using the Pythagorean theorem and the fact that $\kappa = \frac{1}{r}$, it can be deduced that $\kappa = \frac{2 \cdot x}{p^2}$.

A year later after introducing the pure pursuit algorithm, Carnegie Mellon University's Navlab was able to follow road at a maximal speed of 28 km/h [10]. Two decades later, autonomous cars were able to finish a 212 km off-road course at the DARPA Grand Challenge. The winning team introduced a novel path-following algorithm called Stanley method [11]. Stanley method is a non-linear function that takes into account the difference between vehicle heading angle ψ and the LOS angle ϕ, gain parameter k, as well as the cross-track error e. Denoting the vehicle speed u_i at time i, the steering control is given by:

$$steer = (\phi - \psi)_i + \arctan \frac{k \cdot e_i}{u_i} \tag{2.1}$$

Solving the path-following problem is an important step to achieving a fully autonomous driving system. We acknowledge that there are many approaches and path following algorithms [12], but for the sake of brevity, we have highlighted only two.

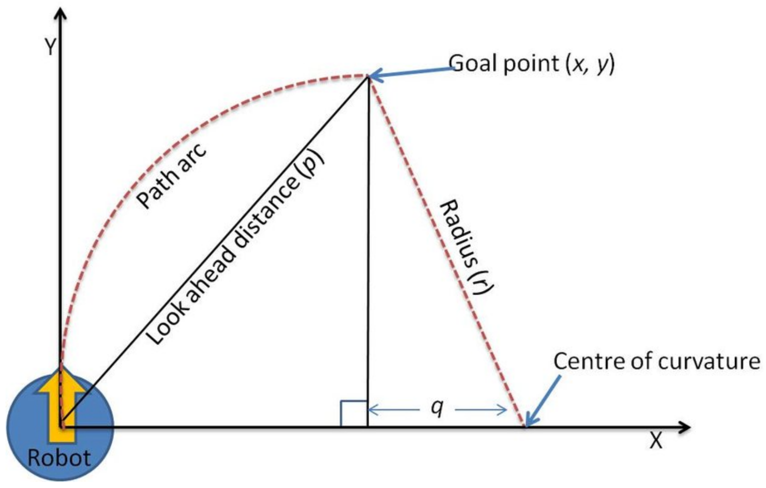


Figure 2.1: The geometry of the pure pursuit path following algorithm, adopted from [9]

2.2 Car Following and Chasing

The goal of a car following problem is to maintain a predefined distance to a followed vehicle. Typically, the problem is solved with the goal of creating an autonomous convoy or a system capable of autonomously driving in highway traffic. Generally, in the car following problem it is expected that the leading car drives smoothly without aggressive turning or sudden braking. On the other hand, in the car chasing problem, the pursued car actively attempts to get away from the chasing car. While the chasing car also tries to maintain a certain distance between the two cars, the ultimate goal is not to lose the pursued car. For this reason, car chasing is a much more challenging problem.

The first scientists who attempted to solve the car chasing/following problem realised that an autonomous system with such capabilities needs to be able to perform many different tasks such as localization and planning. Many attempts focused on solving different parts of the system independently.

In the 1990s, the first attempts of localization of the “lead” chased car were made [13, 14, 15]. Schwarzinger et al. used a database of different car models to match the tracked vehicle to a vehicle model [13]. This gives information about the size of the pursued vehicle. Subsequently, this information is used to estimate the distance and relative position of the pursued car.

In 1991, Kehtarnavaz et al. developed the first autonomous car following system called BART [16]. The system was able to follow the leading car at 20 km/h speed. The system works by changing the velocity of the following car based on the velocity and the distance of the followed car. The authors stated that estimating the velocity of the lead car is a challenging problem. While a new Radar/Lidar sensor fusion solution for velocity estimation has been developed [2], we address the issue by proposing a solution that controls the throttle based only on the distances between the two cars.

2.2.1 Reinforcement Learning

Reinforcement learning is a subfield of machine learning. It deals with what actions should an agent in an environment make to maximize some reward function. An example would be an agent driving a car on a road. In this scenario, the reward could be based on how long the agent is able to drive on the road. The idea is that eventually, the agent would learn from trial and error to match the correct responses (optimal steering value) based on the perceived state (where it is on the road).

With the rise of reinforcement learning in recent years [17, 18, 19], reinforcement learning based solutions for autonomous following and driving have been developed. Michels et al. developed an autonomous driving system capable of driving an RC car in an unstructured environment [20]. They used a supervised learning approach to estimate depth in an image. Then, they used a simulator to learn a control policy to steer based on the depth information from camera images. Zhu et al. developed a solution that imitates human-like velocity control during highway car following [21, 22].

Reinforcement learning algorithms are often trained (at least partly) in a simulator. This limits the ability of the system to drive based on the quality of the simulator. Pan et al. acknowledge the problem and address it by creating a network that converts non-realistic virtual image input into a realistic image with a similar scene structure [23]. Furthermore, they prove that a reinforcement learning agent benefits from training on more realistic images.

We have decided not to use a reinforcement learning approach. A major drawback of reinforcement learning is the black-box nature. On the other hand, since our system has three parts, we can analyze issues and figure out which part caused the issue, i.e., if it was caused by perception, planning, or control. This might not be possible using reinforcement learning. We also believe, that our approach is more suitable for possible hardware changes to the subscale vehicle platform attached to the RC car.

2.3 Target Following Using Unmanned Aerial Vehicles

Unmanned aerial vehicles (UAV) are used in contemporary military operations. The use of UAVs has been steadily increasing for several years [24]. They have been used for remote mapping, data acquisition, target selection, and target chasing [24, 25]. Target chasing and target following using UAVs is a similar problem to the car chasing problem. Generally, the architecture of an autonomous UAV is similar to the architectures used in autonomous vehicles [26, 27]. It is usually divided into three parts: perception and localization, planning, and a controller.

There are two main differences between chasing a target with a UAV compared to vehicle chase. Firstly, UAV moves in more directions, which also means that altitude control is necessary. Secondly, frequently there is an assumption that no obstacles would prevent the UAV to move in all directions [26, 27, 28].

As an example of an autonomous UAV target chasing, Telière et al. developed an autonomous quad-rotor UAV that chases a moving RC car [27]. They set the camera attached to the UAV in a way that the heading vector of the camera is perpendicular to the ground. Then, they estimate the distance error between the center of the image and the moving target. Finally, with the assumption that there are no obstacles, the system moves the UAV to minimize the measured error. This is similar to our approach as we also move the chasing car based on estimated distance between the chasing and chased car. On top of that, our system performs semantic segmentation and plans a trajectory based on information about the surrounding environment.

Chapter 3

Method

3.1 Overview

The system architecture is divided into three parts: perception and localization, trajectory planning, and a trajectory controller. The architecture is depicted in Figure 3.1.

In the perception part, we introduce a novel neural network that detects objects and segments an image at the same time. It is able to find a 2D bounding box of the chased car. Furthermore, it segments an image into a coarse grid of $S \times S$ cells and makes a classification about each cell whether it is a drivable surface or not. The detection and segmentation are done with a single pass of the image through the neural network.

Then, we estimate the angle and distance between the two cars based on the detected bounding box of the chased car. In case no bounding box is found we extrapolate the distance and angle.

After the perception and localization, the trajectory is planned. At this stage, we take advantage of the image segmentation. If only a drivable surface is between the two cars, we try to drive straight at the last position of the chased car. Otherwise, we use the image segmentation and find a direction where it is possible to drive and that is also closest to the position of the chased car.

Finally, with the trajectory planned, we calculate the steer and throttle to drive the vehicle. We use a PID controller [29] to control the throttle. It takes into account the current as well as the previous distances between the chasing and chased car. A Pure Pursuit [8] inspired algorithm is developed to control the steering.

Details of the method will be described in the following sections.

3.2 Computer Vision

3.2.1 Dual-task Neural Network

In this section, we introduce a novel neural network that solves two problems at once – object detection and semantic segmentation.

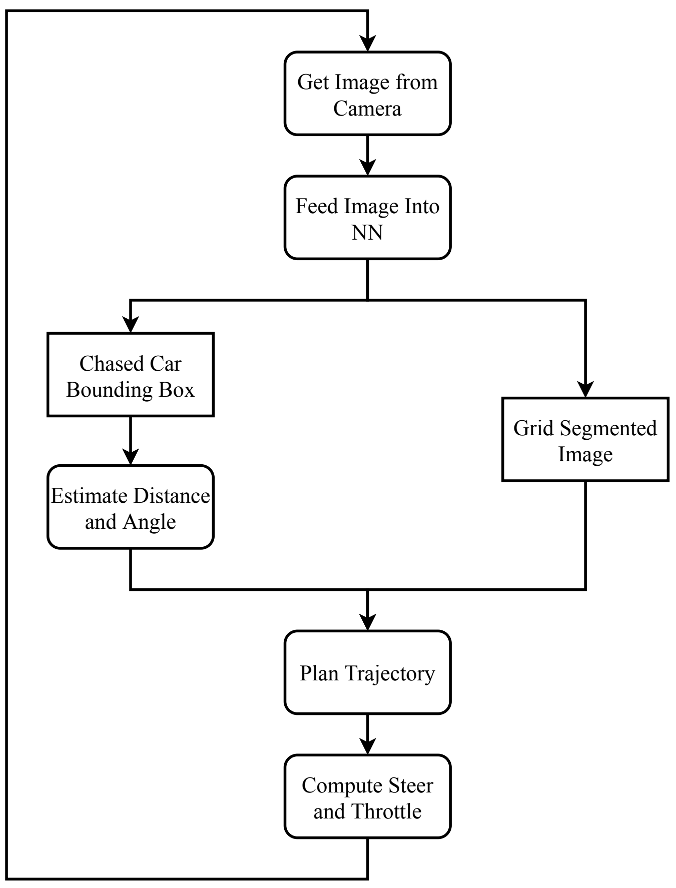


Figure 3.1: Simplified overview of the autonomous driving system

The neural network shares the same backbone for both tasks – a 53 layer feature extractor called Darknet-53 [30]. Attached to the feature extractor are two sets of layers – one that gives the output for the object detection and the other that gives an output for the image segmentation. The architecture of the neural network is depicted in Figure 3.2.

The network is trained by alternating optimization – in every second batch, the network is optimized only for detection, while the segmentation is optimized in the remaining batches. The neural network uses different loss functions depending on the batch. During training, when the network receives a training batch of data for semantic segmentation, the output for the object detection is ignored, and consequently, only the shared layers (the feature extractor) and the segmentation layers are trained. Equivalently, when the network gets a batch containing object detection data, it updates the weights only on the shared layers and the detection layers.

During inference, the network works as follows. It is given an image, which passes through the network just once. From this, we get an object detection output as well as the semantic segmentation output. This means that while the training is slightly slower than training a single-task neural network, the extra cost is negligible during inference since the backbone features are

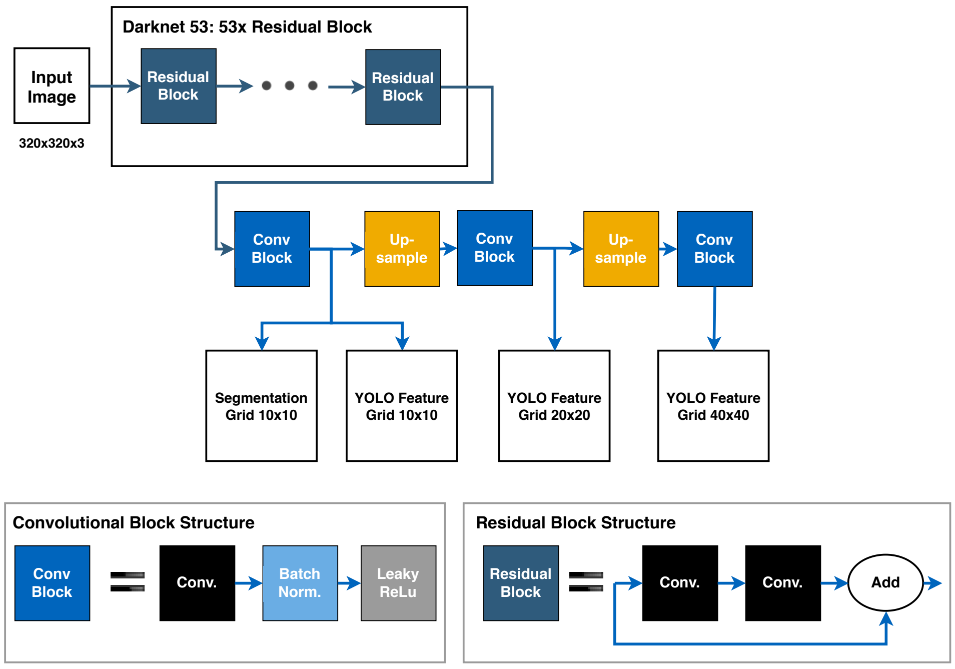


Figure 3.2: Architecture of the novel dual-task convolutional neural network. The three YOLO Feature Grids are used to predict bounding boxes at 3 different scales (described in Section 3.2.2). The Segmentation Grid is the output of the coarse semantic segmentation.

shared.

For autonomous driving, the neural network architecture has many advantages over using two separate networks. First, as was mentioned, it is much faster. That is crucial when driving autonomously because we want to process as many frames per second as possible and have fast reaction times to ultimately prevent accidents and crashes. Second, it uses less memory, because most of the network weights are shared. This is potentially beneficial for the overall speed since more RAM is available, less disk swapping is required. Finally, it makes integration of the detection and segmentation into our autonomous RC car system, having limited computational power, a lot simpler.

3.2.2 Bounding Box Detector

In our work, a bounding box is the smallest 2D axis-aligned rectangle that encloses all pixels belonging to the chased RC car in an image. The detection is performed by a convolutional neural network [31] with YOLOv3 neural network architecture [30].

The first version of the YOLO published in 2015 [32], made a great improvement to the previous state-of-the-art object detection solutions with its ability to predict bounding boxes of multiple object classes with incredible inference speed. It works by reasoning globally about the whole images and all its object classes.

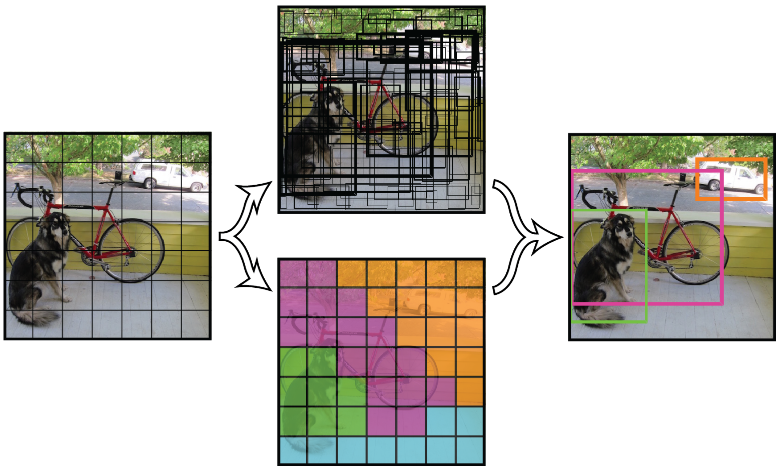


Figure 3.3: Diagram showing YOLO neural network mechanism, adopted from [32]

The YOLO neural network divides an image into $S \times S$ grid of cells. Each cell makes K bounding box predictions. Each prediction is a vector containing width, height and the (x, y) coordinates of the center of the bounding box. For each object in the image, the cell that has the center of the object inside is responsible for predicting its bounding box. This can be seen in the top middle image in the Figure 3.3. Concurrently, each cell outputs C values where each value represents a predicted probability of whether an object from a class C_i is in the cell. The resulting class map from these predicted probabilities is seen in the bottom middle image in the Figure 3.3. Both of these predictions – the class probabilities and the bounding box predictions – are then used together to make detection and classification of the objects in the image.

During training, YOLO optimizes a multi-task loss function. It concurrently optimizes the bounding box regression as well as the class probabilities predictions.

Two years after introducing YOLO, the second version of called YOLO9000 [33] was introduced. In the first version of YOLO, each cell makes K bounding box predictions of a random size. This was improved by calculating the expected bounding box sizes by clustering the bounding boxes of the training set. These expected bounding boxes are called anchors or priors. Furthermore, YOLO9000 also uses different network architecture called Darknet-19 and enables detection on different image sizes.

The third version of the YOLO architecture [30] improves YOLO in two main ways. Firstly, it adds more layers to the Darknet network architecture. Secondly, inspired by Feature Pyramid Networks [34], it predicts boxes at 3 different scales. To be specific, instead of dividing the image into $S \times S$ grid of cells, YOLOv3 divides it into $S \times S$, $(S \cdot 2) \times (S \cdot 2)$ and a $(S \cdot 4) \times (S \cdot 4)$ grid of cells. This helps the neural network to detect objects across a larger range of scales.

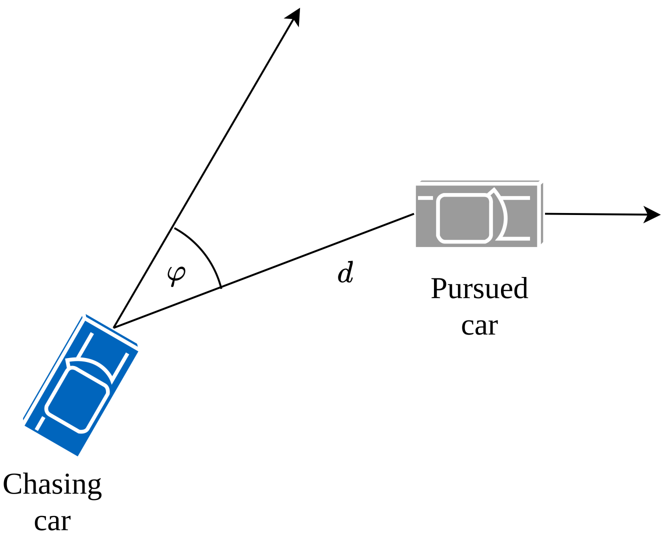


Figure 3.4: Angle and distance estimation: Blue chasing car on the left and pursued car on the right. It shows the ground truth distance d and the angle φ we are estimating using a camera mounted on the chasing car

In our work, we have used the YOLOv3 detector pre-trained on a COCO dataset consisting of more than 330 000 images with objects from 80 different classes [35]. The neural network was fine-tuned on our dataset that has 460 images of the chased RC car. The dataset is described in Section 4.2.1. During fine-tuning, only the final layers that follow the Darknet-53 feature extractor were updated.

3.2.3 Angle and Distance Estimation

One of the most crucial information when chasing another vehicle is its relative angle and distance to the chasing car as sketched in Figure 3.4. In our work, we are estimating the angle between the heading vector of the chasing car and the vector pointing from the center-front of the chasing car to the center of the back of the pursued car. Furthermore, we estimate the distance between the two cars. To be specific, the distance d between the front of the chasing car and the center of the back of the pursued car.

Figure 3.4 shows this problem in a simplified 2D view. To estimate the distance and the angle of the car from an image, we took advantage of existing solutions solving PnP (Perspective-n-Point) problem. The PnP is a problem of estimating position and orientation of a camera from a set of n known 3D points, their corresponding 2D projections in the image, and the calibrated intrinsic camera parameters. This leads us to the following camera projection equation:

$$s\,\mathbf{p_{image}} = \mathbf{K}\,[\,\mathbf{R}\,|\,\mathbf{T}\,]\,\mathbf{p_{model}} \tag{3.1}$$

Where s is a scaling parameter since $\mathbf{p_{image}}$ and $\mathbf{p_{model}}$ are in homogeneous coordinates. The $\mathbf{p_{image}}$ are the coordinates of the points in the image, $\mathbf{p_{model}}$

are the 3D model coordinates, $\mathbf{K}$ is the matrix of intrinsic camera parameters and $\mathbf{R}$ and $\mathbf{T}$ are the rotation and translation we are estimating. This can be further rewritten as:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.2}$$

In our case, the image points p_{image} are the bounding box corners of the detected car. We also estimated the model coordinates p_{model} by measuring the width and height of the chased RC car. We assume that the bounding box encloses only the corresponding image points, i.e., the back of the RC car. We acknowledge that the assumption does not always hold true, but we found the resulting inaccuracies to be negligible. Finally, the intrinsic camera parameters such as focal lengths f_x and f_y were taken from the manufacturer calibration of the ZED camera. Then, we plugged these values into an OpenCV [36] library function SolvePnP (with CV_ITERATIVE flag) which gave us the translation and rotation vectors. Afterwards, we calculated the distance d by calculating the Euclidean distance between the translation vector $\mathbf{T}$ and the origin. Similarly, we extracted the desired estimated angle φ by calculating arc tangent between the negative values of x and z components of the translation vector $\mathbf{T}$.

Finally, since we are using a stereo camera and the camera lens which was used is not in the center of the front of the car, we converted the calculated angle and distance as if the camera was in the center of the front of the car. We achieve this with a simple triangulation. Note that only monocular is made. More precise estimation using stereo vision could be achieved, however, for higher computational cost.

3.2.4 Angle and Distance Extrapolation

In the previous section, we have described how we estimate angle distance based on the bounding box of the chased car. However, in some cases the detector is unable to detect the object in the image due to poor lighting conditions in the scene, or because the pursued vehicle is only partially visible/non-visible due to some object occluding it. Furthermore, there are cases when the chased car is out of the field of view. In these cases, we need to make a reasonable prediction about the position of the chased car.

We extrapolate the angle and the distance of the chased car based on the previous estimates. At the heart of the extrapolation algorithm is an exponential moving average. In fact, we use just six variables to store information about the whole history of estimated angles and distances. The idea will be explained by the process of extrapolating distances. The angle extrapolation is done similarly.

For every frame, we update the exponential moving average variable using the following equation:

$$\hat{e}_i = \begin{cases} \alpha \cdot d_i + (1-\alpha) \cdot \hat{e}_{i-1}, & \text{if bounding box found} \\ \alpha \cdot d_{extrapolated} + (1-\alpha) \cdot \hat{e}_{i-1}, & \text{otherwise} \end{cases} \tag{3.3}$$

Where $\hat{e}_i$ is the exponential moving average variable at ith frame, α is a value in range [0,1] that sets the weight of the last predicted distance. In our algorithm, we have chosen value $\alpha = 0.5$.

If the chased car is detected at ith frame, we set d_i equal to the predicted estimate received from the PnP solver. We then update the variable $\hat{e}_i$ using the estimate d_i and subsequently output the variable d_i as our estimate for ith frame. In case no bounding box was detected, we update the $\hat{e}_i$ variable with $d_{extrapolated}$ variable that is computed using the following linear extrapolation equation:

$$d_{extrapolated} = 2 \cdot d_{i-1} - d_{i-2} \tag{3.4}$$

Afterwards, we output the variable $\hat{e}_i$ as our estimate for ith frame in case no bounding box was detected. Then, we set $d_i = d_{extrapolated}$ for future linear extrapolation computations. Finally, each extrapolated angle is processed by a saturation block that limits the output values. The saturation limits are -175 and 175 degrees.

This method allows us to extrapolate the angle and the distance very fast in a $\mathcal{O}(1)$ time and space complexity with almost no overhead.

3.2.5 Image Segmentation

In comparison to the semantic segmentation that links each pixel of the image to a class label, we segment the image into a coarse grid of $S \times S$ cells. Each cell linked with a class label. In our work, we have chosen $S = 10$ and only two classes. The first class represents a drivable surface while the second class is everything else.

To create a labeled dataset, we used semantic segmented images (an image that has each pixel of the image linked to a class label). Then, we calculate the number of pixels in a cell that belong to the class representing the drivable surface. If the percentage of these pixels is higher than P percentage of all the pixels in the cell, we assign the cell said label. In our work, we have chosen $P = 50\%$.

Since we have only two classes we can represent each cell label as binary 1 or 0. For this reason, we have chosen the logistic loss function to train the network.

$$L_{log} = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i) \tag{3.5}$$

Where N is the number of images per batch, $\hat{y}_i$ are the predicted segmentation values of the neural network for the ith image in the batch and y_i are its labels.

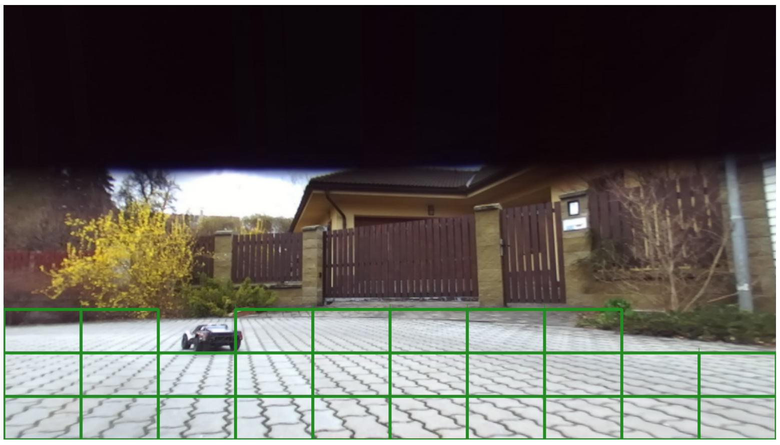


Figure 3.5: Segmented image from the RC car's camera. The image is segmented into 10 × 10 grid of cells where the green cells represent a drivable surface. The image is taken from our test set

3.3 Control And Planning

In this section, we will explain how we control the steering and throttle of the vehicle based on the extracted information from the camera. Conversely to many algorithms [37, 38], we control steering and throttle separately. Steering control is inspired by pure pursuit algorithm and throttle is controlled by a PID controller.

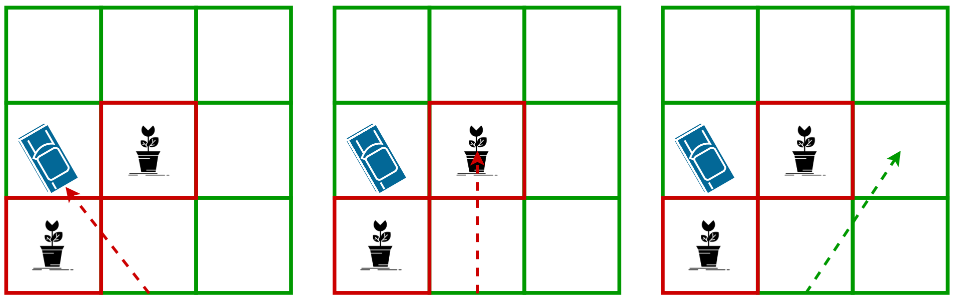
3.3.1 Pure Pursuit Inspired Algorithm

Pure pursuit algorithm is a path tracking algorithm that moves a vehicle from its current position to some look-ahead position on the path. The algorithm was invented in the 1980s as it was used in the first demonstration of an autonomous vehicle capable of following a road using imagery from a black and white camera [8]. The goal of the demonstration was to keep the vehicle centered on the road while it was driving at a constant speed.

The algorithm works as follows. Firstly, it finds the goal point on the path. Then, it transforms the goal point to vehicle coordinates, and finally, it calculates the curvature that will drive the vehicle to the look-ahead position on the path.

In our work, we were inspired by the algorithm. As opposed to the pure pursuit algorithm settings, our car does not move at a constant speed. Secondly, we are not following a path, but rather chasing a moving RC car. Therefore, every time we process an image from the camera, we update our goal position. But unlike in the path-following algorithm, the target position is updated with respect to the current position of the chased RC car rather than to a position on a path.

Conversely to the pure pursuit algorithm, we do not calculate the curvature to control the steering. That is because we need to update the steering every frame based on the position of the chased RC car and take more aggressive



(a) : First proposed path **(b) :** Second proposed path **(c) :** Third proposed path

Figure 3.6: Example of a coarse semantic segmentation based planning using 3×3 grid of cells. First, the algorithm checks if it is possible to drive directly after the car **(a)**. This is not possible as the constructed line segment goes through a non-drivable grid cell. Then, the remaining cells on the same row as the detected car are checked **(b,c)**. The algorithm finally proposes a direction represented by a line segment that goes only through drivable surfaces and that is closest to the direction of the chased car **(c)**.

turns to chase the pursued vehicle. In our algorithm we calculate the steering value with the following function:

$$steer = (\frac{\varphi}{180})^2 \tag{3.6}$$

Value φ represents the angle between the chasing and the chased car shown in the Figure 3.4. The *steer* variable is in [-1, 1] range, where -1 and 1 represent the highest left-turn and right-turn steering angle the car can make respectively. We square the result of the division because the steering control on the RC car is non-linear. We do not square the division in the CARLA simulation.

3.3.2 Semantic Segmentation Based Planning

During inference, we use the coarse segmented grid as depicted in Figure 3.6. First, we construct a line segment from the center of the bottom side of the image to the bounding box of the detected car. If this line segment goes only through cells that are drivable, the trajectory planning phase stays unchanged and we try to follow the car directly as described in Section 3.3.1. If that's not the case, we repeat the line segment construction. This time, however, the line segment goes from the center of the bottom side of the image to other cells on the same horizontal line as the bounding box of the car. When we find a line segment that goes only through cells with drivable surfaces and that is closest to the RC car, we try to drive the car in this direction until it is possible again to chase the car directly.

3.3.3 PID Controller

Despite being introduced in the 1940s, the PID controller is the most common control algorithm. In a 2002 survey of over 11 000 controllers from various industries, it was found that 97% of regulatory controllers utilize PID feedback [39].

In our work, we use a PID controller to control the throttle of the chasing autonomous car. As the name suggests it can be described with the following equation:

$$throttle = P + I + D \tag{3.7}$$

Term P called the proportional term is equal to a value proportional to the current error value ε_i. In our case, the error value is $\varepsilon_i = d_i - d_{desired}$, where d_i is the estimated distance between the chasing and the chased car, and $d_{desired}$ is the desired distance between the two cars that we are trying to maintain. Equivalently to the terms I and D, the term P is equal to a value multiplied by a weight w_p which indicates the influence and importance of the term. Therefore, the term P is described as $P = \varepsilon_i \cdot w_p$

Term I refers to the integration term. It integrates the errors over time. In our case, if the distance between the two cars continues to stay too large for a prolonged period, this term increases its value. It is equal to $I = w_i \cdot \sum_{j=(i-K)}^{i} \varepsilon_j$, where K is the number of frames representing the time we want to consider from the current frame. We used $K = 300$, which corresponds to the last 10 seconds.

The last component of the equation – D refers to the derivation term. Its purpose is to decrease the rate of change and decrease oscillations of the error value. In our work $D = w_d \cdot (\varepsilon_i - \varepsilon_{i-1})$.

To make the PID controller usable based on the dynamics of the system, the weights of the three components must be set. We have picked different weights for the CARLA simulation and the RC car. In the CARLA simulation, we have analyzed over a hundred sets of weights and picked the one that resulted in the lowest average error ($\varepsilon = d - d_{desired}$). These values were $w_p = 0.1$, $w_i = 0$ and $w_d = 1$. For the RC car, we have chosen the values based on trial and error. They were $w_p = 0.25$, $w_i = 0$ and $w_d = 0.07$. Finally, we saturate the output to the [0, 1] range.

The reason for choosing $w_i = 0$ is pretty simple. While higher value of w_i had a positive effect on the average error, it caused the car to crash into the chased car. In many cases, if the car was not catching up it made the car drive very fast and when it reached the desired distance, the car didn't slow fast enough resulting in a crash. Lowering the number of frames the integration part considers, did slightly help this issue. However, we decided not to take the risk of breaking a very expensive subscale vehicle platform attached to the RC car.

Chapter 4

Experiments

4.1 Simulation

4.1.1 CARLA Environment

All the simulation experiments were performed in CARLA – an open-source simulator for autonomous driving research [6]. The experiments were done in CARLA version 0.9.8. CARLA has a wide selection of sensors including camera, segmentation camera, LIDAR on so on. It provides multiple maps with an urban layout. Users can control the weather, as well as many different actors (vehicles, pedestrians) at the same time.

4.1.2 Experiment Setup

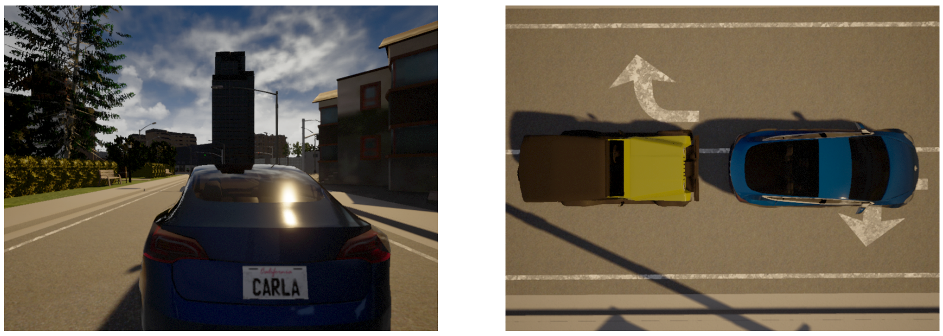
In our work, we chose Tesla Model 3 as the chased car and Jeep Wrangler Rubicon as the chasing car. In each experiment, the pursued Tesla Model 3 was placed in front of the autonomously driven Jeep, as shown in Figure 4.1. At each frame, we simulated a bounding box detection of the chased car. We did this by projecting the ground truth world coordinates of the chased vehicle bounding box to the camera image coordinates. Then, we randomly perturbed the bounding box by adding random exponential noise proportional to the size of the bounding box. Furthermore, we simulated detection failure (false negative) in 10% of the frames. The detection was also not provided when the chased car was out of the field of view.

All the experiments were performed in a synchronous mode in which the simulation waits for the sensor data to be ready before sending the measurements as well as halts each frame until a control message is received [6].

Driving Dataset

To perform the experiments we first collected a driving dataset [40]. We used the Tesla Model 3 and manually drove it around the city using connected steering wheel and pedals Hama uRage GripZ [1]. As the car was driven, its

[1]https://www.hama.com/00113754/urage-urage-gripz-racing-wheel



(a) : View from the stereo camera attached to the chasing car

(b) : View from above

Figure 4.1: Two images showing the beginning of a CARLA experiment from two different perspectives

location and orientation were recorded at every frame. This way, we have collected a database of 20 different drives with varying difficulty (in a sense of chasing the car). We used this database to evaluate the autonomous driving system. At the beginning of the following experiments, a chasing car was placed half of a meter behind the chased car. Then, we updated every frame the location and orientation of the chased car based on the saved coordinates in the dataset. Meanwhile, the chasing vehicle was driven autonomously and trying to maintain a desired distance from the pursued car. Figure 4.2 depicts a full trajectory of an autonomous chase in CARLA.

We divide the dataset into two equal-sized sets based on difficulty. The first set contains simple drives in which the car was driven slowly and without sudden turning and braking. Any changes such as turns were slow. The average speed in these drives is 35.01 km/h.

On the other hand, the second set aims to test the algorithm to its limits. The drives include sudden braking, fast turns as well as cutting corners. Sometimes even drifts were performed. The average speed in these drives is 49.68 km/h with the fastest drive having an average speed of 63.29 km/h.

4.1.3 Segmentation and Extrapolation Evaluation

In the first experiment, we compare different versions of the algorithm. We evaluate the algorithm with its full functionality enabled as described in Section 3, we also test the same algorithm without using coarse image segmentation and finally evaluate the algorithm without the segmentation, and distance and angle extrapolation. We compare these versions on both of the easy and difficult sets described in Section 4.1.2. We evaluated these versions using 5 indicators. Namely, an average root-mean-square error (RMSE) of the desired distance between the two cars and the actual distance between the two cars. The average mean absolute error (MAE) of the desired and the actual distance. Note that the RMSE and MAE are averaged over the drive trials of the test set. We also measure the average number of crashes

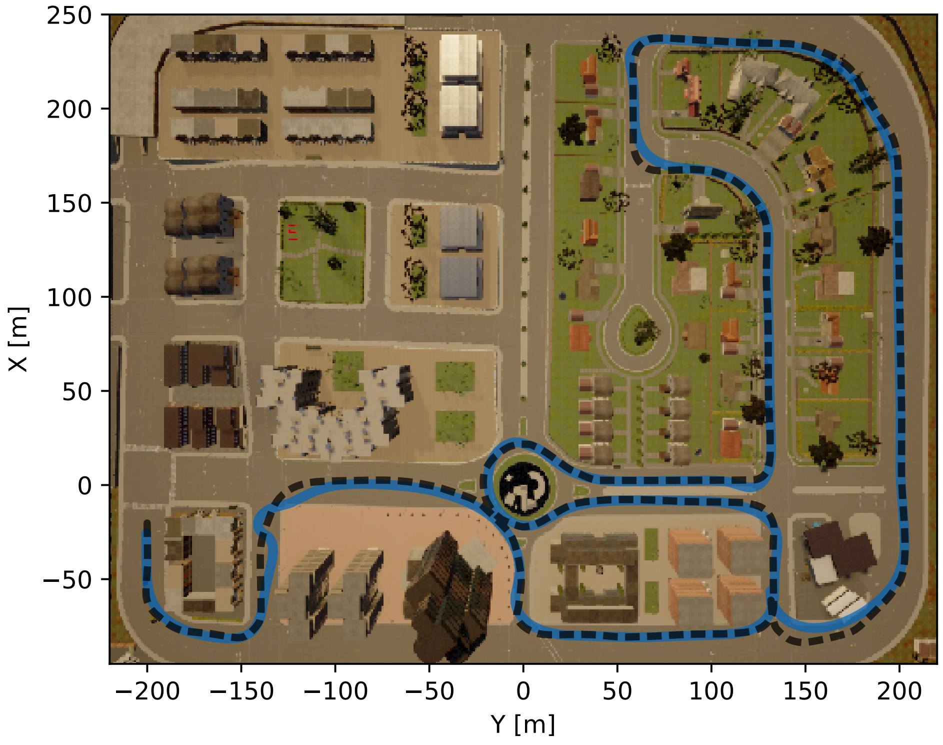


Figure 4.2: Example of a difficult drive trajectory in the CARLA simulator. The full blue line represents the trajectory of the chasing car and the dashed black line represents the trajectory of the pursued car.

	Finished	Avg. Completion	Crashes	MAE	RMSE
Full Algorithm	10	97.48%	0.1	9.28	10.91
W/o Segmentation	10	97.41%	0.1	9.26	10.90
W/o Segm. + Ex.	9	97.03%	0.11	9.34	11.23

Table 4.1: Comparison of the different algorithm versions on the easy driving set

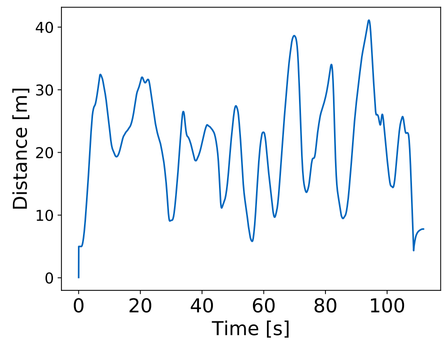
per drive. A crash can be a collision with the environment (building for example) or with the pursued car. Furthermore, we measure the average drive completion, which is based on how long the autonomous system was chasing the pursued car. To be specific, at the last position of the chasing car, we calculated the closest point on the chased car path. Then, we calculated the percentage that the part of the path from the beginning point to this point accounts for. Finally, we evaluate the total number of finished drives. We consider the chase to be finished if the drive was at least 95% completed.

Table 4.1 shows results on the easy test set. All three versions of the system performed well with even the simplest version finishing most of the drives.

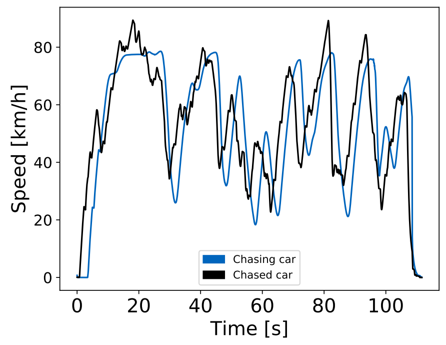
Table 4.2 compares the versions using the difficult driving set. We can see, that the full versions of the algorithm performed significantly better than the remaining two versions. It achieved almost 10% higher average drive completion than the next best-evaluated version. We attribute this to the

	Finished	Avg. Completion	Crashes	MAE	RMSE
Full Algorithm	4	**63.84%**	1.5	14.39	18.30
W/o Segmentation	4	54.76%	1.25	14.30	18.22
W/o Segm. + Ex.	3	53.29%	1	14.49	18.89

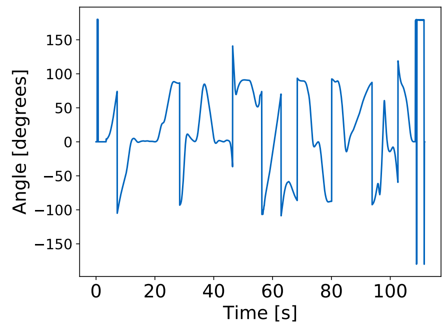
Table 4.2: Comparison of the different algorithm versions on the difficult driving set



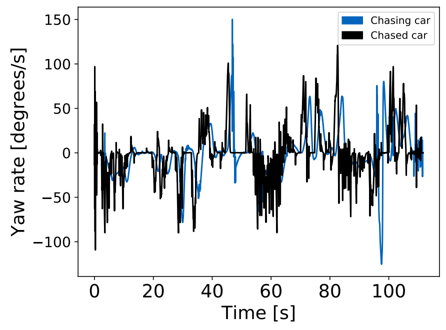
(a) : Distance between the two cars



(b) : Car speed of the two cars



(c) : Angle between the two cars



(d) : Yaw rate of the two cars

Figure 4.3: Different metrics of the car chase shown in Figure 4.1 in time.

use of coarse semantic segmentation. Figure 4.3 depicts experimental results of the full system during a difficult drive shown in Figure 4.1.

4.1.4 Impact of the Detector Quality on Chasing Success

As we can see in Table 4.3 the recall of the detector is significantly lower than its precision. Therefore, in this experiment, we investigate the effects of detectors recall on the ability of the autonomous driving system to chase another vehicle. The failed detections had a uniform distribution over the frames of the sequence. Furthermore, we investigate how each of the three versions of the system copes with different detection recall levels. Figure 4.4 plots the average drive completion (of the 20 drives) described in Section 4.1.3 against the detection recall.

From Figure 4.4 we can see, that the average drive completion steadily

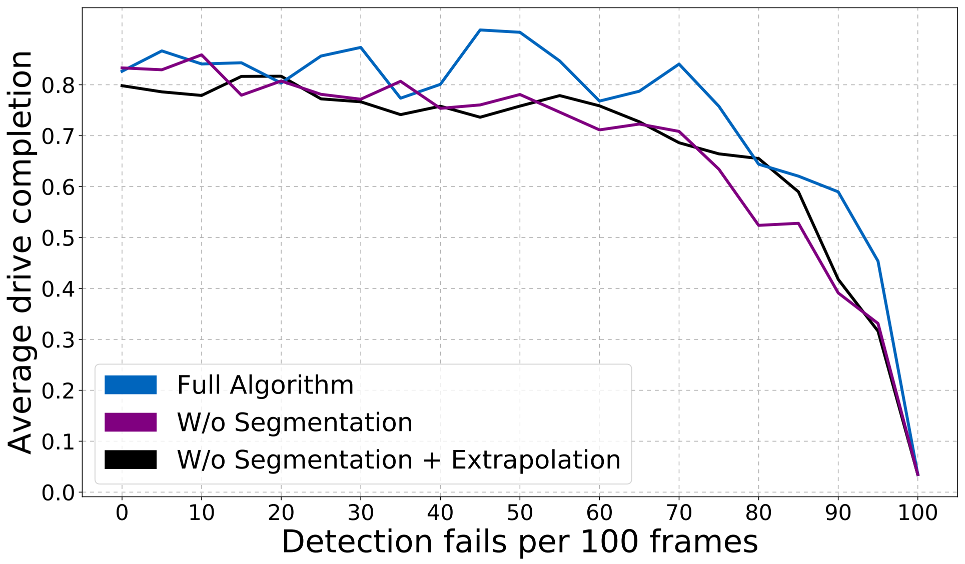


Figure 4.4: Average drive completion plotted against detection recall. All drives included

declines with decreasing detection recall and the system starts to fail once the detection recall is at 25%. Out of the three versions of the algorithm, the full version which uses coarse semantic segmentation and angle and distance extrapolation of the pursued car performs the best. It maintains above 80% drive completion until the detection recall gets below 25%. We attribute this achievement to the use of coarse semantic segmentation. We base this assumption on the fact, that the version of the system that uses extrapolation but does not use segmentation performs similarly to the version which does not use extrapolation nor the segmentation.

Figure 4.5 plots the average drive completion on the difficult driving set against detection recall. On this driving set, the full version of the algorithm performs even more significantly better than the other two versions. When the recall is at 60% or lower, the average drive completion of the full version is almost 20% higher than the average drive completion of the remaining two versions.

4.1.5 Qualitative results

While performing the experiments, we saw that the use of coarse semantic segmentation allowed the system to stay longer on the road in situations when the pursued car was not detected for a prolonged period. For example, when the pursued car drove behind a building, both versions of the algorithm that use extrapolation correctly predicted the position of the pursued car. However, the algorithm that did not use segmentation drove straight into the building and collided as it did not have any information about its surrounding environment. Only the full version of the algorithm drove around the building and continued the chase.

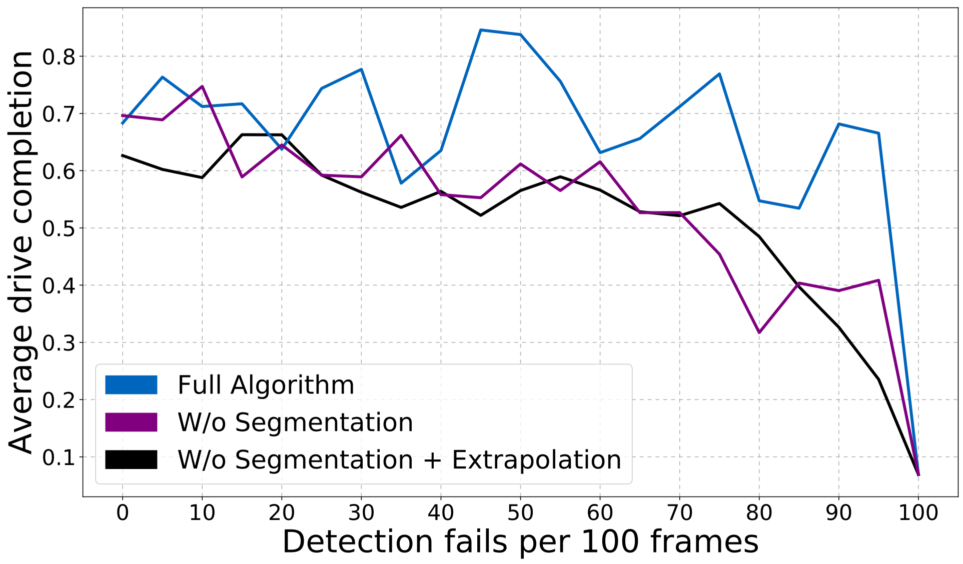


Figure 4.5: Average drive completion plotted against detection recall. Only difficult drives included

4.2 Real Tests

4.2.1 Vehicle Detection

Dataset

In order to train and evaluate an object detector capable of predicting a 2D bounding box around the chased RC car in an image, a dataset was needed. The collected dataset [40] consists of **460** manually annotated images. All image annotations are in the PASCAL VOC data format introduced in the PASCAL Visual Object Classes Challenge [41]. Each image has its annotation that contains pixel coordinates of the bounding box corners for each object in the image. In our dataset, a single object per image is present – the chased RC car. The data were collected from multiple locations around Czechia, on various surfaces, and also under different lighting conditions. The majority of images were collected by the ZED Stereo camera attached to the chasing RC car. Around 5% of the collected images were taken with smartphone Xiaomi Mi 8. A few images from the dataset are shown in Figure 4.6.

Detection Results

The collected dataset was split into three sets. The majority of 80% of the images were randomly selected into a training set, 10% were randomly selected into a validation set and the remaining images were used as a test set. To evaluate the model, we calculated multiple statistics – recall, precision, XY loss, and WH loss. To explain recall and precision we need to first define IoU (Intersection over union). IoU measures how much the predicted bounding

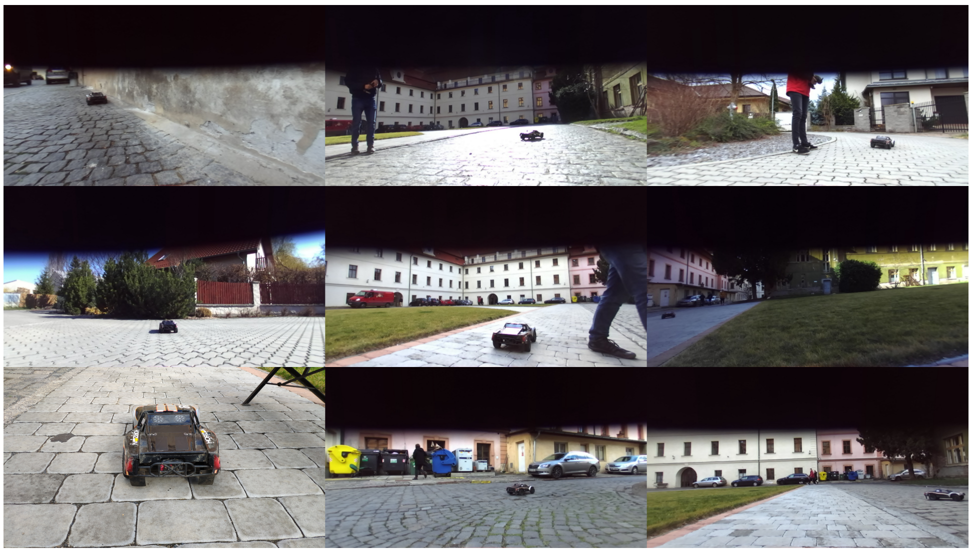


Figure 4.6: Detection dataset. Overview of the environment variability present in the dataset

	Precision	Recall	XY loss	WH loss
Test set	99.8%	97.7%	0.066	0.227

Table 4.3: Evaluation of the detector on a test set

box overlaps with the ground truth (annotated bounding box). It is defined as

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}} \tag{4.1}$$

We say that the prediction is true positive (tp) when IoU between the predicted bounding box and the ground truth bounding box is greater than 0.5. We say that it is false positive (fp) if it is less or equal to 0.5. When a ground truth bounding box is present in the image and the model failed to detect the object, we classify it as false negative (fn). Finally, we define recall as $recall = \frac{tp}{tp+fn}$ and precision as $precision = \frac{tp}{tp+fp}$. We calculate XY loss as a mean squared error (MSE) between the center of the predicted bounding box and the center of the ground truth bounding box. WH loss, on the other hand, is the difference between width and height of the predicted bounding box and the ground truth bounding box. When calculating both of these values, coordinates and image dimensions have been scaled to the [0, 1] range.

During training, the model with the smallest sum of the WH and XY losses on the validation dataset has been selected. The evaluation of the detector on the test set is shown in Table 4.3. Results indicate that the pre-trained model detects the chased car accurately. This is achieved with only a few hundred training examples. Examples of the detection, together with angle and distance estimates are shown in Figure 4.7.

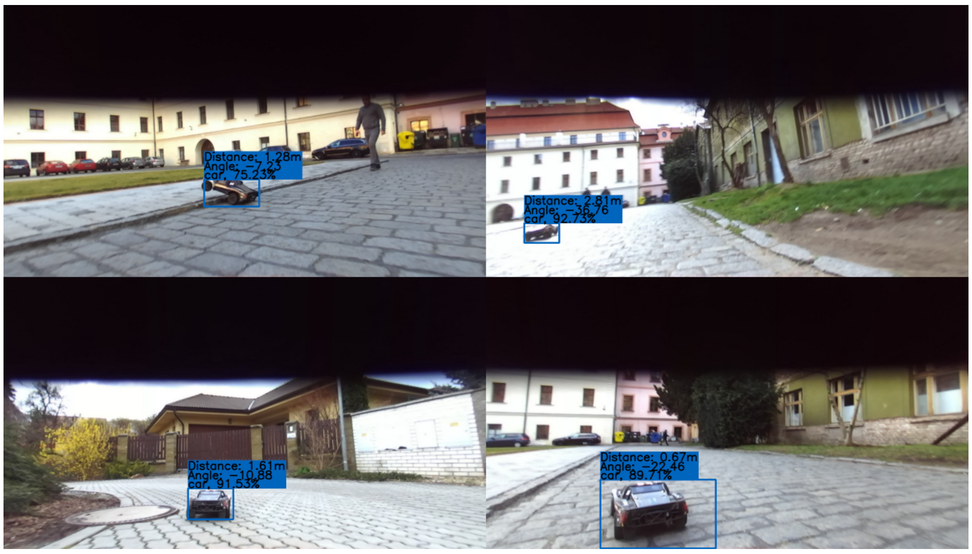


Figure 4.7: Detection bounding boxes, distance, and angle estimation performed on images from the RC car camera. The images are part of the test split of the collected dataset

4.2.2 Segmentation Results

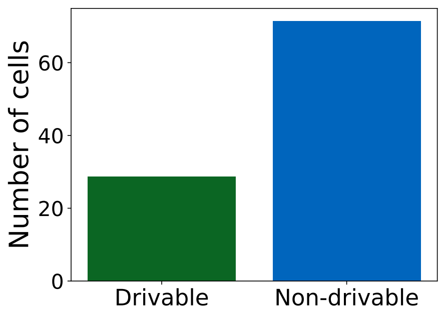
To train and evaluate the coarse semantic segmentation part of the dual-task neural network, we have annotated the collected detection dataset presented in Section 4.2.1. As explained in Section 3.2.5, each image is segmented into a 10×10 grid of cells. Therefore, we treat this as a classification problem, where the network performs 100 classifications per image. We report, that the neural network achieved 94.4% classification accuracy on the test set.

4.2.3 Test Drive Using RC Cars

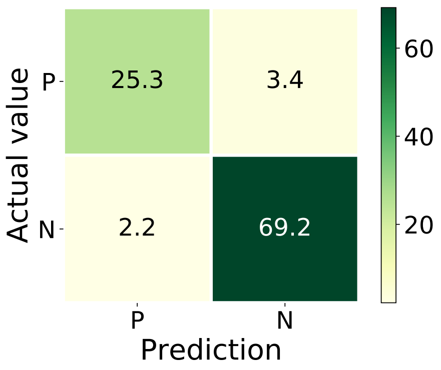
RC Cars Description

Chasing RC car. For the deployment of our autonomous chasing algorithm, a subscale vehicle platform called Toyota Mini (ToMi) was used [42]. The platform is built around a large 1:5 scale RC car “Losi Desert Buggy XL-E 4WD” [2]. This electrically powered car has 0.9 x 0.5 meters length and width dimensions with reported maximal speed of up to 80km/h. The platform has a Raspberry Pi with a Navio-board[3] that generates pulse width modulation (PWM) signals for the throttle and steering to the servomotor. It is also equipped with a ZED stereo camera[4] for taking color images, NVIDIA Jetson AGX Xavier graphics card[5] for image processing and neural network inference.

[2]http://www.losi.com/Products/Default.aspx?ProdID=LOS05012T2
[3]https://emlid.com/navio/
[4]https://www.stereolabs.com/zed/
[5]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/



(a) : Class distribution



(b) : Confusion matrix. Value P represent Drivable surface while N represents the non-drivable surface

Figure 4.8: Confusion matrix of the coarse segmentation predictions and class distribution of the test set. The numbers represent the number of instances per image, i.e., out of 100 cells in a single image

Finally, it is equipped with SSD to provide additional storage, GPS, and IMU.

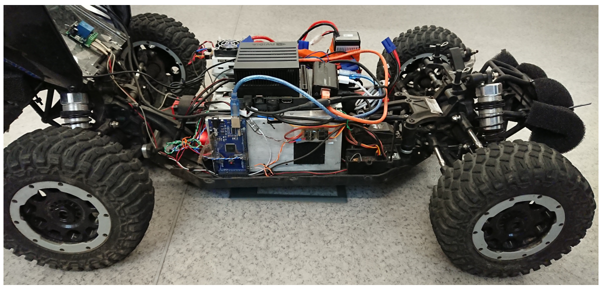
The whole simplified process flow is as follows. First, an image is taken by a camera, which then goes to the graphics card where it is analyzed by our algorithm. Then, based on the image analysis, both steer and throttle values are updated. These values are transmitted from the graphics card to the Raspberry Pi which then sends a signal to the servomotor that actuates the vehicle.

Chased RC car. The RC car used as the chased vehicle is a 1:10 scale “Losi XXX-SCT Brushless RTR”[6]. This electrically powered car has 0.55 x 0.29 meters length and width dimensions with reported maximal speed of up to 55 km/h. Both vehicles are shown in Figure 4.9.

Qualitative results

We have performed live tests using a version of the system that did not have the dual-task neural network and therefore did not have the option to perform coarse semantic segmentation. The system was tested on an empty roundabout. During live tests, several safety measures were available. Multiple software conditions were implemented to stop the system. For example, in case the ZED camera stopped capturing images. We also limited the maximum speed of the autonomous vehicle. This was done by limiting the pulse width modulation (PWM) signals sent to the servo motor. On top of that, a remote emergency relay that can return manual control was available. We tested the system on multiple days during multiple weather conditions. The chasing trajectories driven on a road leading to the roundabout included

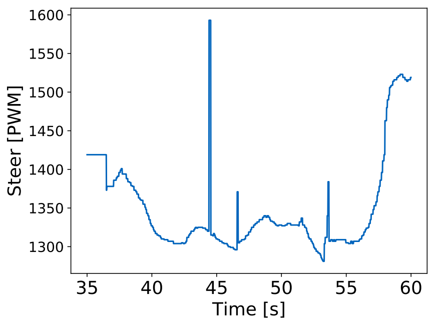
[6]http://www.losi.com/Products/Default.aspx?ProdID=LOS03002



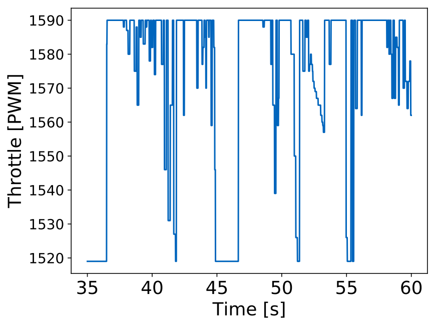
(a) : Inside of the chasing RC car

(b) : Chasing RC in the back and the chased RC car in the front

Figure 4.9: The hardware inside of the autonomous chasing RC car on the left and both RC cars used for testing on the right



(a) : Steer values in time



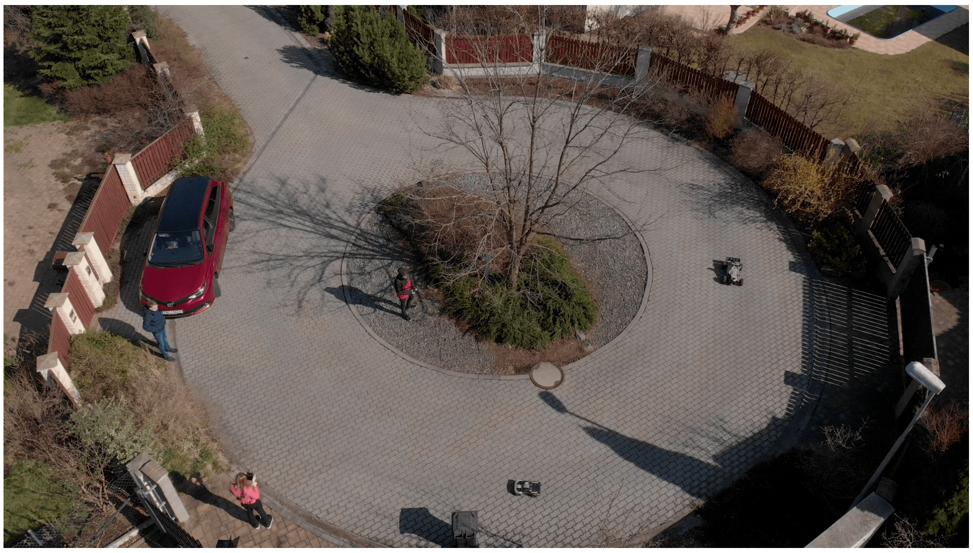
(b) : Throttle in time

Figure 4.10: Steer and throttle PWM signals during the live test. The time in the graph corresponds to the time in the video of the live test [43]. The sudden spike in the graph **(a)** is caused by a false positive detection in a single frame

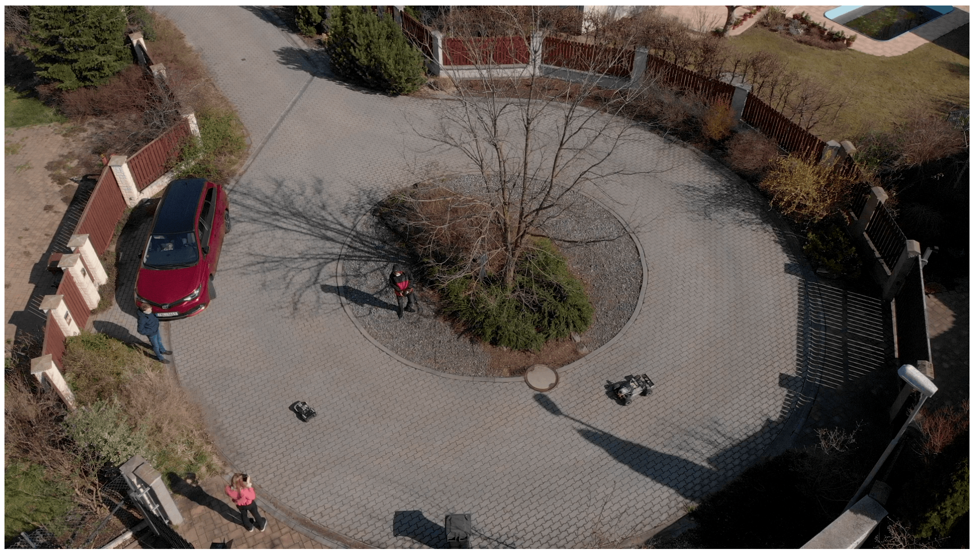
a straight line and a U-turn. On the roundabout, the trajectories consisted of arc segments and full circles.

The autonomous system followed the other car smoothly without jerky movements. For the most part, it was able to successfully chase the other RC. It was maintaining the desired distance when chasing the other car on the roundabout as well as when the pursued car was driving in a straight line. If the chased car stopped, so did the autonomous system. The only limitation of the system comes from its current reactive nature, which in some instances affected the system's ability to perform U-turn on a narrow road.

The autonomous chase was documented and a highlight video is available online [43].



(a) : Image taken at t seconds



(b) : Image taken at $t + 1.9$ seconds

Figure 4.11: Drone photos of the live test. The RC car on the right is driven autonomously, while the RC car on the left is manually controlled with a transmitter. The person in the center of the image controls the chased car. The person closest to the red Toyota holds an emergency device that can stop the autonomous car and the third person controls the drone. Video of the autonomous car chase is available online [43]

Chapter 5
Conclusion

We have presented a novel dual-task convolutional neural network that concurrently detects objects and predicts coarse semantic segmentation. The network was trained using alternating optimization. We have developed a system, capable of autonomously chasing another vehicle, that uses the dual-task network. We have extensively evaluated different versions of the system in CARLA simulator. We have created a new challenging publicly available chasing dataset collected by manually driving the chased car. We have shown that the system benefits from using the coarse semantic segmentation. Especially if the detector has low detection recall.

We have collected and annotated three new publicly available datasets [40]. In addition to the CARLA driving set, we have collected an object detection dataset consisting of 460 images of a pursued RC car as well as a semantic segmentation dataset consisting of 460 images segmented into drivable and non-drivable parts.

Finally, we have implemented the system on a subscale vehicle platform built on a high-speed RC car. We have demonstrated and documented [43] the system's ability to chase another car.

Currently, the system shows robust chasing capabilities despite using only information from a single RGB camera. One of the system limitations is its reactive nature. We believe that the system could improve by using a more sophisticated trajectory planning algorithm that would include predictive modeling of the chased car. In the future, we would also like to evaluate the system more extensively with RC cars.

Bibliography

[1] M. Brackstone and M. McDonald, “Car-following: a historical review,” *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 2, no. 4, pp. 181–196, Dec. 1999. [Online]. Available: https://doi.org/10.1016/s1369-8478(00)00005-x

[2] D. Gohring, M. Wang, M. Schnurmacher, and T. Ganjineh, “Radar/lidar sensor fusion for car-following on highways,” in *The 5th International Conference on Automation, Robotics and Applications*. IEEE, Dec. 2011. [Online]. Available: https://doi.org/10.1109/icara.2011.6144918

[3] J. Ziegler *et al.*, “Making bertha drive—an autonomous journey on a historic route,” *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014. [Online]. Available: https://doi.org/10.1109/mits.2014.2306552

[4] A. Broggi *et al.*, “PROUD-public road urban driverless test: Architecture and results,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, Jun. 2014. [Online]. Available: https://doi.org/10.1109/ivs.2014.6856478

[5] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, 2009.

[6] A. Dosovitskiy *et al.*, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[7] P. Sujit, S. Saripalli, and J. Sousa, “An evaluation of UAV path following algorithms,” in *2013 European Control Conference (ECC)*. IEEE, Jul. 2013. [Online]. Available: https://doi.org/10.23919/ecc.2013.6669680

[8] R. Wallace *et al.*, “First results in robot road-following,” in *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI’85. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985, p. 1089–1095.

[9] N. Kumar and Z. Vámossy, “Robot navigation with obstacle avoidance in unknown environment,” *International Journal of Engineering &*

Technology, vol. 7, no. 4, p. 2410, Sep. 2018. [Online]. Available: https://doi.org/10.14419/ijet.v7i4.14767

[10] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer, "Toward autonomous driving: the cmu navlab. i. perception," *IEEE Expert*, vol. 6, no. 4, pp. 31–42, 1991.

[11] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge." *J. Field Robotics*, vol. 23, pp. 661–692, 01 2006.

[12] J. M. Snider, "Automatic steering methods for autonomous automobile path tracking," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-09-08, February 2009.

[13] M. Schwarzinger *et al.*, "Vision-based car-following: detection, tracking, and identification," in *Proceedings of the Intelligent Vehicles 92 Symposium.* IEEE, 1992. [Online]. Available: https://doi.org/10.1109/ivs.1992.252228

[14] H. Schneiderman, M. Nashman, and R. Lumia, "Model-based vision for car following," in *Sensor Fusion VI*, P. S. Schenker, Ed. SPIE, Aug. 1993. [Online]. Available: https://doi.org/10.1117/12.150245

[15] R. Sukthankar, "RACCOON: A real-time autonomous car chaser operating optimally at night," in *Proceedings of the Intelligent Vehicles 93 Symposium.* IEEE, 08 1993, pp. 37–42. [Online]. Available: https://doi.org/10.1109/ivs.1993.697294

[16] N. Kehtarnavaz, N. Griswold, and J. Lee, "Visual control of an autonomous vehicle (BART)-the vehicle-following problem," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 3, pp. 654–662, 1991. [Online]. Available: https://doi.org/10.1109/25.97520

[17] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[18] M. Moravčík *et al.*, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, Mar. 2017. [Online]. Available: https://doi.org/10.1126/science.aam6960

[19] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Oct. 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1724-z

[20] J. Michels, A. Saxena, and A. Y. Ng, "High speed obstacle avoidance using monocular vision and reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning - ICML 2005.* ACM Press, 2005. [Online]. Available: https://doi.org/10.1145/1102351.1102426

[21] M. Zhu, X. Wang, and Y. Wang, “Human-like autonomous car-following model with deep reinforcement learning,” *Transportation Research Part C: Emerging Technologies*, vol. 97, pp. 348–368, Dec. 2018. [Online]. Available: https://doi.org/10.1016/j.trc.2018.10.024

[22] M. Zhu *et al.*, “Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving,” 2019.

[23] X. Pan, Y. You, Z. Wang, and C. Lu, “Virtual to real reinforcement learning for autonomous driving,” in *Procedings of the British Machine Vision Conference 2017*. British Machine Vision Association, 2017. [Online]. Available: https://doi.org/10.5244/c.31.11

[24] S. Kreps and J. Kaag, “The use of unmanned aerial vehicles in contemporary conflict: A legal and ethical analysis,” *Polity*, vol. 44, no. 2, pp. 260–285, Apr. 2012. [Online]. Available: https://doi.org/10.1057/pol.2012.2

[25] J. Everaerts, “The use of unmanned aerial vehicles (uavs) for remote sensing and mapping,” *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 37, 01 2008.

[26] F. Rafi, S. Khan, K. Shafiq, and M. Shah, “Autonomous target following by unmanned aerial vehicles,” in *Unmanned Systems Technology VIII*, G. R. Gerhart, C. M. Shoemaker, and D. W. Gage, Eds. SPIE, May 2006. [Online]. Available: https://doi.org/10.1117/12.667356

[27] C. Teuliere, L. Eck, and E. Marchand, “Chasing a moving target from a flying UAV,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2011. [Online]. Available: https://doi.org/10.1109/iros.2011.6094404

[28] F. Lin *et al.*, “A robust real-time embedded vision system on an unmanned rotorcraft for ground target following,” *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, pp. 1038–1049, Feb. 2012. [Online]. Available: https://doi.org/10.1109/tie.2011.2161248

[29] J. G. Ziegler and N. B. Nichols, “Optimum settings for automatic controllers,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 115, no. 2B, pp. 220–222, Jun. 1993. [Online]. Available: https://doi.org/10.1115/1.2899060

[30] J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” 2018.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109/5.726791

[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2016. [Online]. Available: https://doi.org/10.1109/cvpr.2016.91

[33] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jul. 2017. [Online]. Available: https://doi.org/10.1109/cvpr.2017.690

[34] T.-Y. Lin *et al.*, “Feature pyramid networks for object detection,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jul. 2017. [Online]. Available: https://doi.org/10.1109/cvpr.2017.106

[35] ——, “Microsoft COCO: Common objects in context,” in *Computer Vision ECCV 2014*. Springer International Publishing, 2014, pp. 740–755. [Online]. Available: https://doi.org/10.1007/978-3-319-10602-1_48

[36] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[37] A. Broggi, S. Debattisti, M. Panciroli, and P. P. Porta, “Moving from analog to digital driving,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, Jun. 2013. [Online]. Available: https://doi.org/10.1109/ivs.2013.6629615

[38] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, “Optimal trajectory generation for dynamic street scenarios in a frenet frame,” in *2010 IEEE International Conference on Robotics and Automation*. IEEE, May 2010. [Online]. Available: https://doi.org/10.1109/robot.2010.5509799

[39] L. Desborough and R. Miller, “Increasing customer value of industrial control performance monitoring -honeywell’s experience,” *AIChE Symposium Series*, vol. 98, 01 2002.

[40] P. Jahoda, “Autonomous car chase,” https://github.com/JahodaPaul/autonomous-car-chase, 2020.

[41] M. Everingham *et al.*, “The Pascal Visual Object Classes (VOC) Challenge,” *Int. J. Comput. Vision*, vol. 88, no. 2, p. 303–338, Jun. 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[42] M. Bahnik *et al.*, “Visually assisted anti-lock braking system,” in *IEEE Intelligent Vehicles Symposium*. IEEE Intelligent Vehicles Symposium, 2020, To appear.

[43] P. Jahoda, “Autonomous car chasing | czech technical university in prague,” https://www.youtube.com/watch?v=SxDJZUTOygA, 2020.