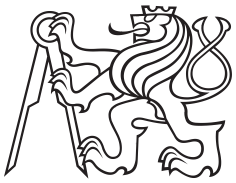
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Semantic Surface Segmentation for RC car model

Michal Bahník

Supervisor: Ing. Jan Čech, Ph.D.
Field of study: Cybernetics and Robotics
May 2020

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Bahník Michal** Personal ID number: **434824**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Semantic Surface Segmentation for RC car model

Master's thesis title in Czech:

Sémantická segmentace povrchu pro RC model auta

Guidelines:

Develop a method for semantic segmentation for the instrumented RC car model (developed at the departments of Cybernetics and Control). Focus on surface classification in front of the vehicle. Propose a neural network and a suitable loss function for training.
Perform ride tests and quantitatively evaluate the experiment. Demonstrate the system such that the vehicle will be automatically driven on the given surface type, e.g. interlocking pavement, gravel path, etc.

Bibliography / sources:

[1] Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Proc. Medical Image Computing and Computer-Assisted Intervention. 2015.
[2] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, Yizhou Yu. FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation. ArXiv preprint arXiv:1903.11816, 2019.
[3] Gabriel M. Hoffmann, Claire J. Tomlin, Michael Montemerlo, Sebastian Thrun. Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing. In Proc. American Control Conference, 2007.

Name and workplace of master's thesis supervisor:

Ing. Jan Čech, Ph.D., Visual Recognition Group, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **13.02.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until:
by the end of summer semester 2020/2021

Ing. Jan Čech, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to thank the supervisor of my diploma thesis, Dr. Čech, for his inspiring advice and comments during the elaboration of this thesis.

Declaration

I declare that I have worked on this thesis independently using only the sources listed in the bibliography. All resources, sources, and literature, which I used in preparing or I drew on them, I quote in the thesis properly with stating the full reference to the source.

In Prague, 22. May 2020

Abstract

This work proposes a system for semantic surface segmentation for autonomous driving control.

A fully convolutional neural network of U-net architecture was trained for semantic surface segmentation. Learning and validation was performed on a manually annotated data set, which was collected within this work. A modified focal loss function was used to improve the accuracy and robustness of the classification. The modification includes weighting of individual pixels according to their position in the image. The resulting model shows higher accuracy and robustness of segmentation than the model trained with cross-entropy, a common loss function.

The semantic segmentation system was implemented into an existing subscale mobile platform based on the RC model. To verify the functionality of the proposed solution, experiments were performed in a real environment. The experiments consisted of autonomous drive of the platform on a selected surface. This was done by processing the segmentation map by simple algorithm resulting in heading reference. Having the reference computed, P controller is used to control the velocity and steering of the vehicle.

The performed experiments showed the ability of the proposed solution to provide a segmentation map with sufficient accuracy for autonomous driving of the subscale vehicle.

Keywords: neural network, CNN, semantic segmentation, UNet, autonomous driving, surface segmentation

Supervisor: Ing. Jan Čech, Ph.D.
Katedra kybernetiky
Fakulta elektrotechnická
ČVUT v Praze
Karlovo náměstí 13
121035 Praha 2

Abstrakt

Tato práce se zabývá návrhem systému pro sémantickou segmentaci povrchu pro autonomní řízení.

K sémantické segmentaci povrchu byla natrénována plně konvoluční neuronová síť U-net architectury. Učení a validace proběhlo na ručně anotovaném datasetu, který vznikl v rámci této práce. Za účelem zlepšení přesnosti a robustnosti klasifikace byla použita modifikovaná fokální ztrátová funkce. Modifikace spočívá ve váhování jednotlivých obrazových bodů podle jejich pozice v obrazu. Výsledný model pak vykazuje vyšší přesnost a robustnost segmentace, než model natrénovaný s křížovou entropií, běžně používanou ztrátovou funkcí.

Systém sémantické segmentace byl implementován do již existující platformy založené na modelu RC auta. K ověření funkcionality byly provedeny experimenty v reálném prostředí. Experimenty spočívaly v autonomní jízdě vozidla po předem daném povrchu. Toho bylo docíleno zpracováním segmentační mapy jednoduchým algoritmem, jehož výstupem je směrová reference. K řízení rychlosti a zatáčení jsou použity P regulátory.

Provedené experimenty ukázaly schopnost navrženého řešení zajistit segmentační mapu s dostatečnou přesností pro autonomní jízdu vozidla.

Klíčová slova: neuronové sítě, CNN, sémantická segmentace, UNet, autonomní řízení

Překlad názvu: Sémantická segmentace povrchu pro RC model auta

Contents

Figures

Tables

Chapter 1

Introduction

For most of a robotic applications, environment sensing is an essential ability. This is especially true for autonomously operating robots, including cars or aerial vehicles. The development of both hardware and software in recent years allowed the use of advanced and highly precise machine perception techniques. A large area of machine perception is machine vision. Machine vision may be described as a discipline of using computer vision in robotic applications. With high resolution cameras, high performance computational technologies (CPUs and GPUs) and advanced machine learning algorithms, the machine vision is nowadays used in a huge variety of industry areas. Examples can be found in medical diagnosis [RFB15b], agriculture [MLS18] and of course in autonomous vehicles development ([SGA+18, YLCT20]). All these applications require robust and reliable systems for tasks including classification, recognition, identification, segmentation, regression and many others.

With a focus on automotive industry, semantic segmentation plays a crucial role in autonomous driving capabilities. In order to get information about a type of objects around a vehicle, semantic segmentation of camera images is used. Resulting the so called segmentation maps then hold information what type of object (car, pedestrian, etc.) or surface (road, grass, pavement, etc.) is located on each pixel (see Fig 1.1). Information about a surface in front of a car may be used in large variety of practical applications. They include autonomous driving systems or driver assistance systems [BFP+20].

The aim of this work is:

Figure 1.1: Example of car-mounted camera image semantic segmentation reproduced from [BFC09]

1. To develop a reliable surface segmentation system relying on a front camera, to analyse results and to integrate the system into a multi-sensor sub-scale vehicle.

2. To implement simple path tracking algorithm and perform autonomous drive experiments in a real environment as a proof-of-concept.

The rest of the thesis is structured as follows: First, the TOMI platform is introduced in Chap. 2. All on-board sensors and computing devices are described and the architecture and functionality are covered. Chap. 3 reviews current state-of-the-art semantic segmentation algorithms and presents architecture selected for the purpose of this thesis. Then, data set used for training, validating and testing is presented in Chap. 4, including synthesised data. Neural network training process, including data preprocessing and augmentation, loss function proposal and accuracy evaluation, is covered in Chap. 5. In Chap. 6, all the modifications made to TOMI platform are described. Results of semantic segmentation model training are presented and analysed in Chap. 7. Chap. 8 presents autonomous drive experiments and evaluates them. And finally, results are concluded in Chap. 9.

Chapter 2

TOMI Platform

Experimental drives are performed on a Toyota Mini (TOMI) sub-scale vehicle platform. This multi-sensor platform built on an RC car model was developed in Toyota Lab at the CTU. The author of this thesis was one of developers. However, some modifications were necessary to be done for the purpose of this work. All those adjustments are listed and described in Sec. 6 and were done as a part of this work.

The TOMI platform was designed to be a multipurpose development subscale vehicle platform for various experiments. Data from all sensors (including RC throttle and steer signals) are logged. The on-board computer with integrated GPU provides high computational power for machine learning applications.

By the time of writing of this thesis, the TOMI platform was used for experiments in several other diploma theses. Detailed description of the platform can be found in [BFP+20].

2.1 Hardware

TOMI platform is based on an RC car model instrumented with several on-board computers, various sensors and data drives. Extra battery packs were mounted in order to provide independent and stable power supply for

Figure 2.1: Side view on the TOMI platform.

all on-board computers and sensors.

2.1.1 RC Car Model

TOMI platform builds on a commercial RC car model Losi Desert Buggy XL-E 1/5. This specific model has been selected since it satisfies following requirements established for TOMI project:

1. Electric motor
2. Load-capacity and space for on-board computers and sensors
3. Replacement parts availability

Main criterion was internal space and load-capacity, since several on-board computers (Jetson Xavier, Raspberry Pi and Arduino Mega), multiple sensors (IMU, GPS module, ...) and extra battery pack for on-board computers needed to be mounted inside of a model. Additional load to a platform due to the instrumentation (including support constructions, circuitry, etc.) is approximately 2.1 kg (original model weight is 13.8 kg).

2.1.2 On-board Computers

In early experiments, Jetson Nano was used as a central computation unit. Later on, it was replaced by Jetson Xavier due to higher performance. Jetson

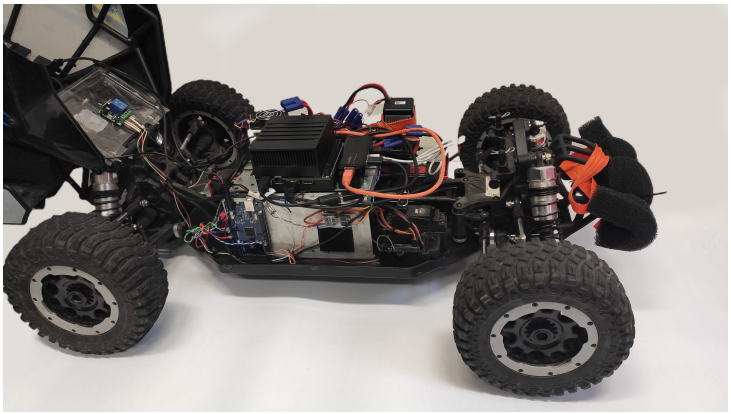


Figure 2.2: TOMI platform inside view

Xavier [Jet] is an embedded computing board with GPU running Ubuntu Linux. The latter is especially convenient for deep learning applications development, since machine learning frameworks (PyTorch, TensorFlow/Keras) with Python API can be used for both development and experiments.

For a PWM reference control and sensor data acquisition, Raspberry Pi with Navio shield was used. Raspberry Pi was chosen due to its Ubuntu-like OS (convinient for scripting based development), GPIO pins (providing PWM, USART) with drivers and LAN port (used for SSH connection from Jetson Xavier).

Arduino Mega provides decoding of PWM signals (throttle and steer) from remote control and providing numerical PWM characteristics to a Raspberry Pi. Since very low computation power is required and short code was expected for this application, embedded solution providing enough speed and low power consumption was selected.

Jetson Xavier can be operated using standard hardware inputs and outputs (external monitor, keyboard, mouse) or via SSH channel. This is especially useful for outdoor development and experiments. Since Raspberry Pi is in the same network, it can be accessed via SSH from Jetson Xavier.

Overall diagram of on-board computers can be found in Fig. 2.3.

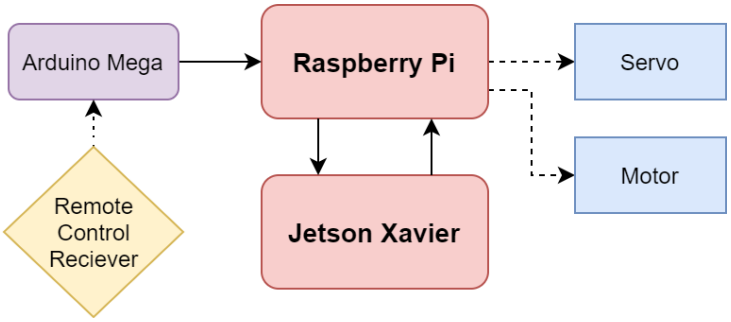


Figure 2.3: TOMI platform architecture block diagram. Full lines denote USART, dashed PWM signal.

2.1.3 Sensors

TOMI platform was mounted with variety of different sensors. Most relevant for this work is a ZED stereo-camera [ZED], but also IMU (with 6-axis accelerometer and gyroscope), GPS module and Hall sensor RPM gauge. All of sensor data logged in a file with a timestamp for synchronisation with image data.

As the main visual sensor, ZED stereo-camera has been chosen. Basic requirement was providing good image quality with real-time frequency. ZED camera is capable of capturing 1080p at 30 FPS and due to the low lens aperture ($f/2.0$), short exposition time can be used in order to minimise motion blur. It also provides universal and fast connectivity (USB3) and Python API. On top of that, ZED camera software Development Kit provides depth computation and basic visual localisation.

2.2 Communication and Computation Distribution Architecture

Distributed architecture was chosen for a TOMI platform in order to separate higher level computations and control (machine learning, autonomous control, data processing, etc.) and lower level control (motor and servo PWM control, data acquisition). Simplified architecture with indicated data flows is in Fig. 2.3.

Computation demanding operations like machine perception, navigation, planning, data processing and acquisition are done on Jetson Xavier. Com-

munication with Raspberry Pi is established via USART in both ways. PWM commands for motor and servo control can be sent from Jetson Xavier to Raspberry Pi. On the other side, Raspberry Pi collects data from all sensors and sends it to Jetson Xavier. Moreover, Raspberry Pi reads throttle and steer signals (decoded by Arduino Mega from remote control receiver) and forwards them to the motor and steering servo.

Chapter 3

Semantic Segmentation - State of the Art

Semantic segmentation is defined as a classification problem, where a class is to be predicted for each pixel of an input image. Precise semantic segmentation was hard to achieve with classical approaches. But thanks to recent advance of deep neural networks and convolution layers, precision and robustness improved up significantly. This development allowed semantic segmentation to be used in many practical applications, including but not limiting to autonomous driving or medical images analysis.

3.1 Fully Convolutional Networks

Fully convolutional networks (FCN) introduced a big step forward in the field of semantic segmentation. As stated in [UA19], fully convulutional networks were able to perform well with much less parameters, then previous deep networks, while input image is not limited to a fixed size.

FCNs were improved by introducing various new approaches. As an example, well-known and successful encoder-decoder architectures with skip connections are presented.

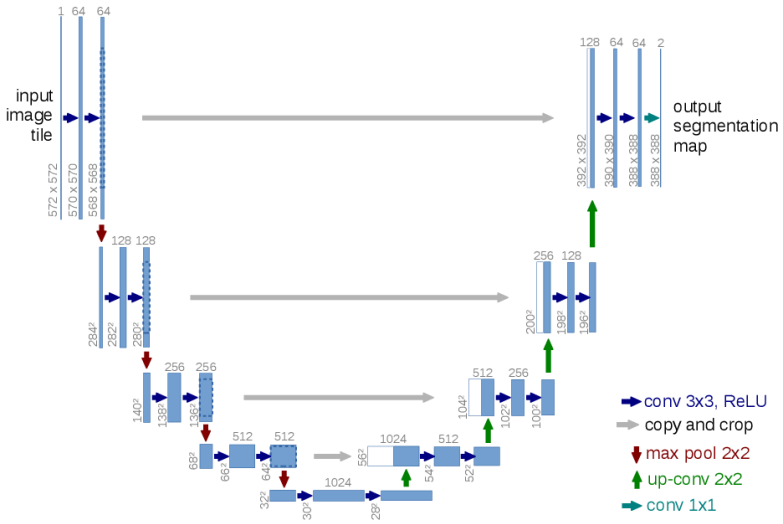


Figure 3.1: U-net architecture (example for 32x32 pixels in the lowest resolution) scheme reproduced from U-Net: Convolutional Networks for Biomedical Image Segmentation [RFB15a]

3.2 U-net and Encoder-Decoder Architectures

Encoder-decoder architecture is a widely used design pattern. Whole architecture is divided to two parts: encoder and decoder. First, input is mapped by encoder to a feature space and then decoded by decoder to an output space. Softmax layer is then used as the last layer to predict class probability scores for every pixel of an image and is defined as

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}, \tag{3.1}$$

where x_i (x_j) is i-th (j-th) input to a Softmax layer. Length of $\mathbf{x}$ is equal to the number of classes to be predicted. In other words, softmax layer normalises signals into a probability scores distribution proportional to signal strengths. All values are in range $[0, 1]$ and sum to 1.

A prominent example of a encoder-decoder architecture is a U-net, presented in [RFB15b]. It follows base encoder-decoder architecture, but introduces a skip connections from encoder to decoder (see Fig. 3.1). According to authors, this architecture is able to train and perform well even for a small training data set and it is capable of real time segmentation. Originally, it was developed for biomedical image classification, but it performed well in other segmentation tasks, such as satellite imagery segmentation [WZL+19, RDN18].

Another advantage of this architecture is the fact, that is it fully convo-

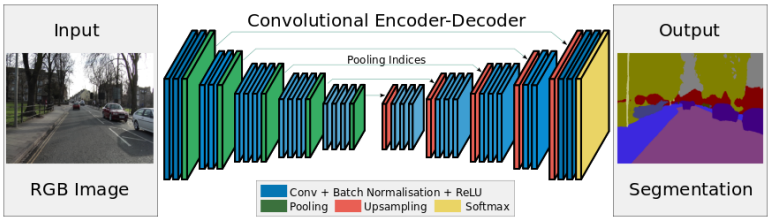


Figure 3.2: SegNet architecture scheme reproduced from SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation [BKC17]

lutional. Therefore, the network can be trained on images of arbitrary size without changing the architecture. Moreover, the trained network can be fed with images of different size than it was trained for. This proves to be useful, as described in Chap. 7.

Another example of encoder-decoder architecture with skip connections is SegNet, presented in [BKC17]. Architecture is similar to U-net, but instead of entire feature map, only polling indices are transferred in skip connections (see Fig. 3.2). This also results in smaller memory requirements in comparison with U-net.

Both U-net and SegNet are considered to be a good choice for a pixel-wise segmentation architecture. In a paper [DGHC19] authors did not find significant difference between performance of both architectures. Nevertheless, U-net performed slightly better for smaller training data set, which is also mentioned in original U-net paper [RFB15b].

Also, there is no suitable database of annotated surface images known to the author of this work. All data need to be annotated manually and thus, limited data set is available. Due to this fact, U-net architecture was preferred, since it performs well even on smaller data sets [RFB15b].

3.3 Other Architectures

Early FCN architectures suffered from several drawbacks, such as inability to process global context knowledge, or lack of multi-scaling processing abilities (as described in [UA19]). There are many recently introduced architectures aimed to improve early FCNs drawbacks. Some of those architectures worth mentioning are Spatial Pyramid Pooling Network (SPP-Net [HZRS14]), Recurrent Convolutional Networks (ReSeg [VRC+16]) and many others. Re-

cently many attention-based methods were presented with aim to improve ability of global corelations [NSD+20, HWH+19, MHZ19].

However, those architectures do not introduce extraordinary performance improvement and are not well established yet, so for the purpose of this work, proven architectures are considered.

Chapter 4

Data Set

Data set used for training, validation and testing consists of two parts: manually annotated real-world data and synthesised data. First, only manually annotated data were used for training. Later on, various amounts of synthesised images were used to extend the training data set.

4.1 Orthorectification

All images are rectified to the birds-eye view (orthographic rectification). This warping assures that images are not dependent on a specific camera point of view. This is especially useful, when camera position may change or inputs from different cameras are processed. Since images are transformed to the same ortho-plane, new homogpraphy mapping needs to be found, but neural network does not need to be trained again. Transformed images are also proportional to the metric plane map in font of the car. The full image scale is approximately $1\,\mathrm{cm\,px^{-1}}$.

The warp transformation was found by capturing a planar rectangular object in front of the car. Then, homography estimation H was computed from the known object's corner points positions in ortho-plane and object's corner points position in the image.

Figure 4.1: Image from one lens taken on a test ride on the CTU ground.

4.2 Manually annotated real-world data

Several test drives were performed with the TOMI platform in real environment conditions and distinct images taken by camera were selected as training data. Images were orthorectified and annotation maps were created manually using *PixelAnnotationTool* [Bré17]. As an example of this process, original picture form one lens is in Fig. 4.1. The same image after rectification and manually annotated map can be seen in Fig. 4.2.

Data set consists of 276 annotated images and contains 6 surface types (classes): Grass, interlocking pavement, cobblestone, gravel, asphalt and other. Image cutouts, objects (benches, persons, ...) and other (or unrecognisable) surfaces as labelled as other. Class distribution of manually annotated data is in Fig. 4.4.

Our early experiments showed that images taken by camera suffered from overexpose due to the presence of very bright areas (sky in upper part of the image). This problem was effectively eliminated by improvised sun shield mounted to the camera frame. This devalued upper part of the image (as seen in Fig. 4.2), but did not affect resulting orthorectified image, since relevant pixels are in lower part of the image.

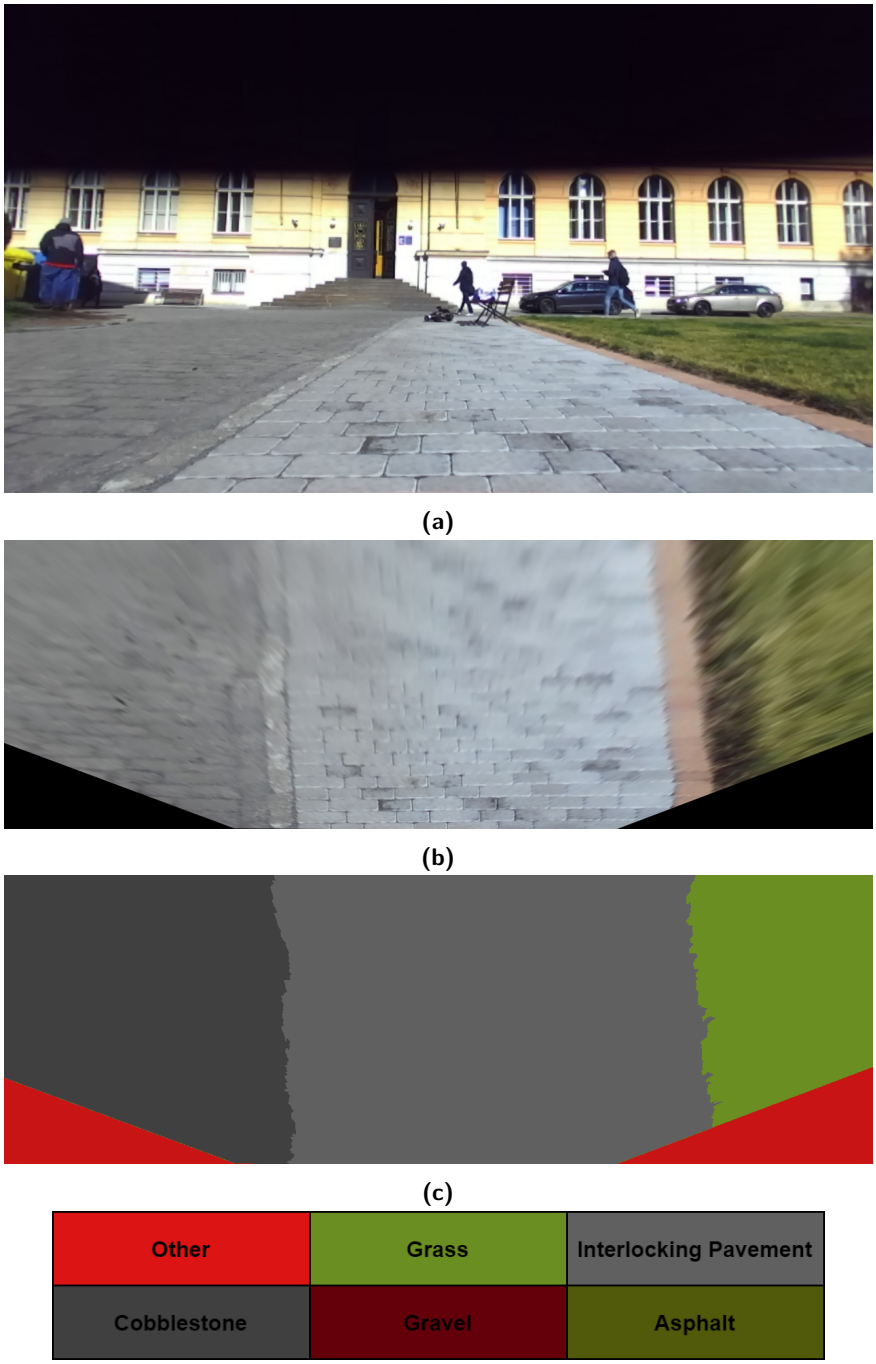


Figure 4.2: Image from camera (a) is rectified to birds-eye view (b) and then manually annotated (c). RGB representation of annotation map is done according to (d).

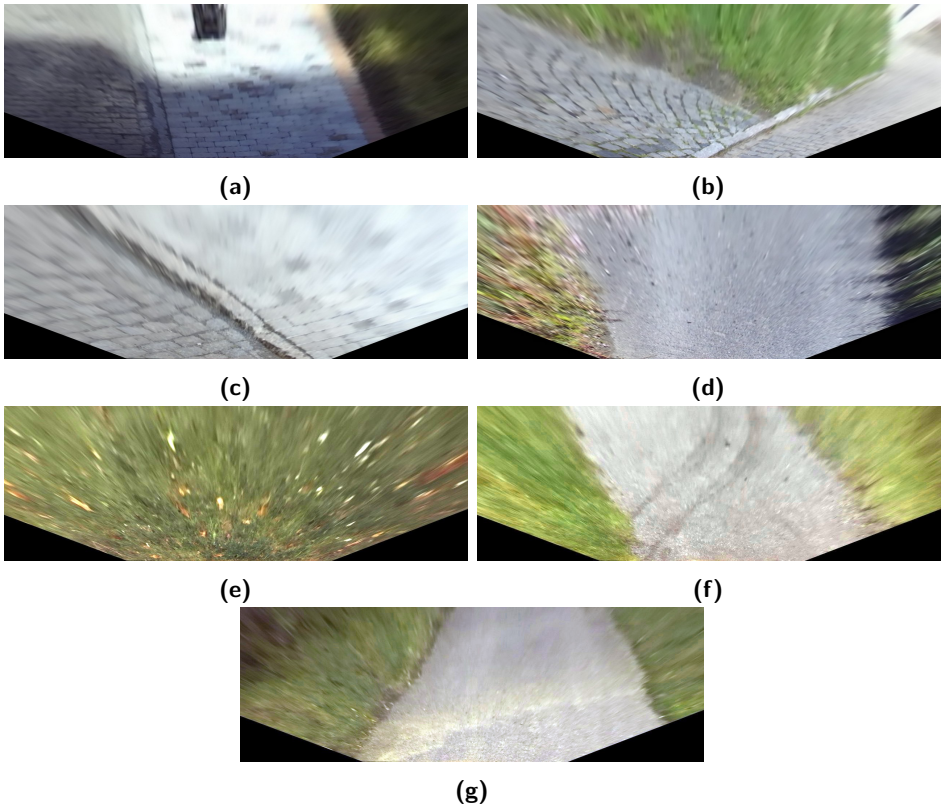


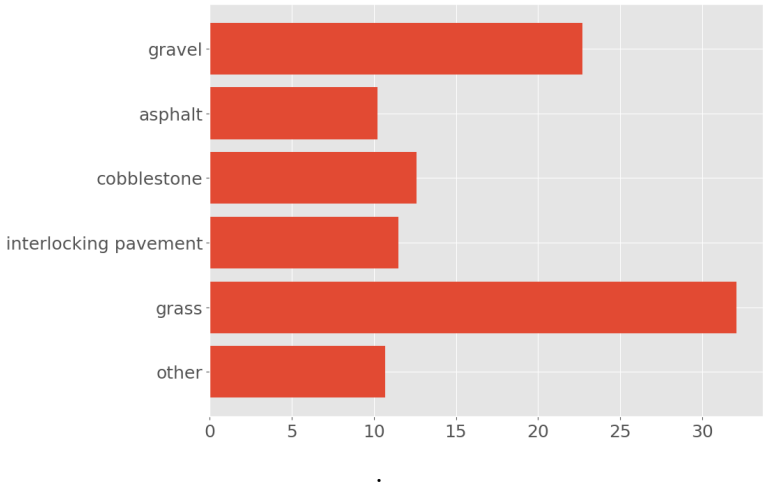
Figure 4.3: Examples of various manually annotated images from data set.

4.3 Synthesised data

Since number of manually annotated images in data set is relatively small (276 images), a data synthesis is proposed to increase data set size in order to achieve better results of segmentation. Since resulting images are to be in birds-eye view perspective, the synthesis is rather simple in comparison with raw camera imaged, because it does not take the perspective into an account.

4.3.1 Source data

As a source data, high resolution images of various surfaces are used. All images were unified in resolution and scale, so a pixel corresponds to same length in every image. Resulting images have approximately same scale as real-world rectified images. All source images are of 4000 px × 4000 px.



.

Figure 4.4: Class distribution of manually annotated data set: Other 10.7 %, grass 32.1 %, interlocking pavement 11.5 %, cobblestone 12.6 %, asphalt 10.5 % and gravel 22.7 %

These images are randomly rotated and other surface is placed on top of it (with random rotation and shift). Same operations are done with annotations, so a large number of synthetic images with ground-truth annotation maps is available.

To make synthesised data more authentic, resulting image is transformed from bird's eye view to camera perspective and back again. This generates image degradation similar to the one on rectified images. Example of a synthesised image is in Fig. 4.5.

(a)

(b)

(c)

(d)

Figure 4.5: Synthesised images for train data set extension (a), (c) and corresponding annotation maps (b), (d).

Chapter 5

Neural Network Training, Testing and Validation

5.1 Image preprocessing

After loading into a memory, all images are resized to fixed size. This size can be specified and the effect of selected size on training process is discussed in Chapter 7. Note that images are resized using bilinear interpolation, while annotation maps are resized using nearest-neighbour interpolation. This preserves annotation map consistency, but may introduce resampling artefacts.

5.2 Data augmentation

When data set is not large enough or when greater accuracy is needed, data augmentation may be performed to artificially inflate the data set. Common data augmentation techniques are affine or perspective transformations, contrast and colour adjustments, noise perturbation, etc.

Because the degradation caused by rectification is not centro-symmetric, images cannot be rotated, since training process expects the degradation in the upper part of the image (see Sec. 5.3.1). Nevertheless, (random) horizontal

flip is performed in order to augment the data set.

5.3 Loss function

As a default loss function, Negative Log-Likelihood (NLL) is used for training. Since used neural network architecture has a Log-Softmax layer as a output layer, training criterion is effectively Cross Entropy (see [Bis06]). However, for the purpose of this work, NLL shall be considered to be a loss function in the rest of the work.

NLL loss function for an image (one input sample) is defined as

$$L(\mathbf{p}, \mathbf{y}) = -w_s \frac{\sum_{n=0}^{N} \log(p_{n,y_n})}{N}, \tag{5.1}$$

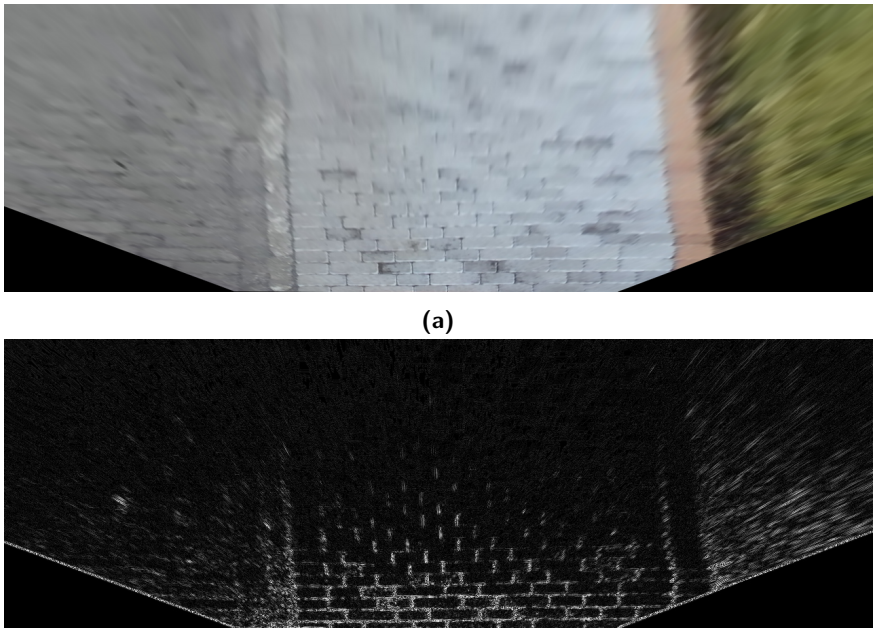
where $\mathbf{p}$ is a distribution of a probability score over classes, $\mathbf{y}$ is image labelling ground-truth annotation map, N is number of image elements (pixels), p_{n,y_n} is output value corresponding to n-th element for true class, w_s is a sample (image) weight. Note that for simplicity, images and annotation maps are treated as a 1D vector of length $N = width \cdot height$ instead of 2D matrices.

After image rectification to bird's eye view perspective, upper part of an image suffers from a significant information loss due to lack of resolution caused by camera viewpoint. Since the NLL loss function considers every pixel to be equally important, missclassified pixels in the very top of the image (where the image is strongly distorted) contributes to total loss the same way, as pixels in the bottom (where the image is less distorted). In other words, the network is learning on the heavily distorted parts of the image the same as on less distorted parts. This may lead to worse accuracy and capability of generalisation. Further, most of the time we are interested mainly in the immediate proximity of the vehicle. Therefore pixel weighting was proposed in order to favour pixels of the less distorted part of the image.

5.3.1 Pixel-wise area of interest weighting

In order to achieve more accurate segmentation in the important regions closer to the vehicle, pixel-wise weighted NLL loss function is proposed:

$$L(\mathbf{p}, \mathbf{y}) = -w_s \frac{\sum_{n=0}^{N} w_n \log(p_{n,y_n})}{\sum_{n=0}^{N} w_n}, \tag{5.2}$$



(a)

(b)

Figure 5.1: Laplacian operator was applied to rectified image (a) to illustrate the loss of resolution (b) due to camera perspective.

where w_n is n-th pixel weight.

Weight of every element lies in interval $[0; 1]$ and is computed using formula given by:

$$w = 2 - \frac{1}{e^{-kd}}, \tag{5.3}$$

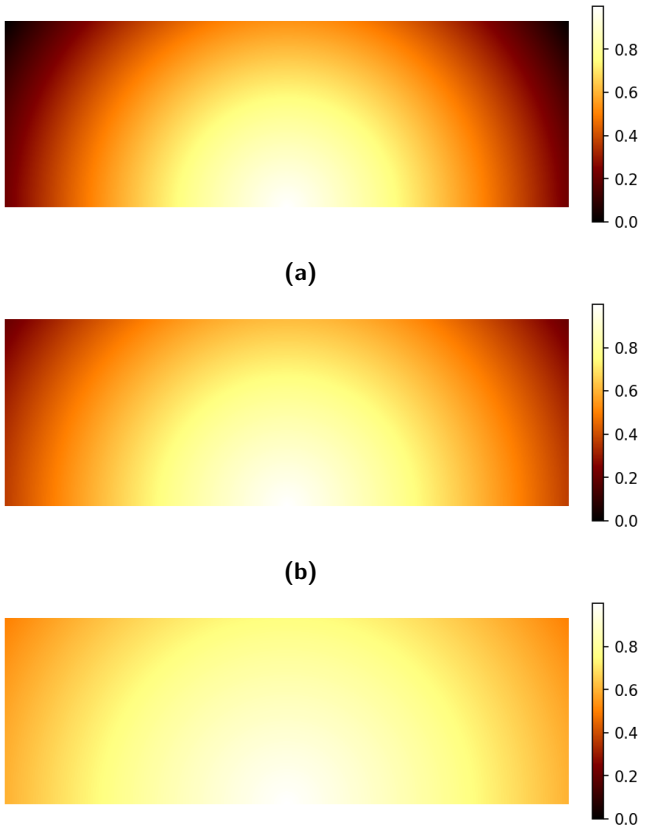
where w is a pixel weight, d is Euclidean distance of a pixel to the centre of the bottom side of the image and k is weight decay coefficient. It is computed as

$$k = \frac{\log(2 - m)}{h}, \tag{5.4}$$

where h is image height and m is desired minimal weight (in top left or right corner). in Figure 5.2c, weight maps are visualised for different values of m.

5.3.2 Focal loss

Focal Loss [LGG+17] is a recent approach to handle unbalanced training sets (classes are not distributed uniformely). It is a modification of NLL loss function introducing weight, which decreases exponentially with growing SoftMax layer output (predicted probability). Focal Loss for one pixel is defined as



(c): Visualisation of pixel weights for $m = 0$ (a), $m = 0.2$ (b) and $m = 0.5$ (c).

$$FL(p_n, y_n) = (1 - p_{n,y_n})^\gamma - \log(p_{n,y_n}). \tag{5.5}$$

Intuitively, Focal Loss focuses more on pixels, where the prediction is uncertain. This way it copes with unbalanced training set and improves training convergence. The effect of a focusing parameter γ is evaluated in Chapter 7.

Then, final loss function formula containing both Pixel-wise area of interest weighting and Focal Loss is

$$L(\mathbf{p}, \mathbf{y}) = -w_s \frac{\sum_{n=0}^{N} (1 - p_{n,y_n})^\gamma w_n \log(p_{n,y_n})}{\sum_{n=0}^{N} w_n}. \tag{5.6}$$

5.4 Optimiser

Adam [KB14] optimiser was used in all our experiments. It adapts learning rate dynamically by using the first and second gradient moments. Thus it is less sensitive to learning rate setting, than a vanilla Stochastic gradient descent (SGD).

5.5 Accuracy evaluation

In order to reflect the effect of pixel-wise weight loss training, accuracy function needs to be adjusted. To get a comparable accuracy metric, same weights w_n as proposed in Subsection 5.3.1 for loss function are used in accuracy

$$A(\hat{\mathbf{y}}, \mathbf{y}) = w_s \frac{\sum_{n=0}^{N} w_n [\hat{y}_n = y_n]}{\sum_{n=0}^{N} w_n}, \tag{5.7}$$

where A is a image accuracy, $\hat{\mathbf{y}}$ is a segmentation map, $\hat{y}_n$ is n-th pixel in an segmented map, $\mathbf{y}$ is a ground-truth annotation map and y_n is n-th pixel in annotation map. Note that [.] stands for Inverson bracket.

5.6 Training, validation and testing

Manually annotated data are randomly split to training (50 %), validation (25 %) and testing set (25 %). Testing data set is fixed for all experiments while the rest of the data is randomly split into training and validation data before every training.

Evaluation on validation data is performed after every training epoch and only model weights achieving the highest accuracy on the validation set are saved. The trained model is evaluated on test data set to get the resulting accuracy (see Chap. 7).

Additionally, synthesised data (see Section 4.3) were added to the training set in order to improve the training process. Different proportions of manually

annotated and synthesised data were used in the training process and the effect is evaluated later in Chapter 7.

Due to the model size (more than 2 millions parameters), batch size for training was limited by hardware capability. For original image size after bird's eye transformation with size 424 px × 1280 px, the maximum possible batch size is 2. In order to improve convergence speed, batch size was increased to 5 by training on smaller images (resolution of 320 px × 960 px). Effects on convergence and resulting accuracy using full size and half scale images are discussed in Chapter 7.

5.7 Training on GPU

Training of neural networks was performed on GPU servers at CTU. Servers are equipped with NVIDIA GeForce GTX 1080Ti GPUs. One training epoch on 139 images (non-extended train data set) took in average 26 sec, while validating on 71 images (validation or test data set) took in average 6 sec. Average number of epochs was approximately 1100, which results in almost 10 hours of a total time spent on one training process of the neural network.

Chapter 6

TOMI platform adjustments

In order to safely control steer and throttle of the TOMI platform from on-board computer (Jetson Xavier), some adjustments needs to be done.

6.1 Autopilot Interface

For any kind of autonomous driving, there is a need for direct control over the steering and throttle of the platform. An Autopilot interface was implemented to simply set throttle and steer from anywhere in the code. These steering and throttle set points are sent from Jetson Xavier to the Raspberry Pi, where PWM signals for servo and motor are generated (see Fig. 2.3). Nevertheless, remote control signals are still recieved and logged.

6.2 Fail-save systems

Failure of any crucial part of a control system may result in loss of control, which may lead to a serious collision of a TOMI platform with the surrounding environment. Since the platform weights almost 16 kg and maximum speed is about 80 $\mathrm{km\,h^{-1}}$, such a collision can cause substantial damage to a platform or property or injuries. Thus, fail-save systems must be implemented in order to minimise potential risk of a collision.

6.2.1 Autopilot Interruption

Since the platform is designed to receive remote control signals independently of a control mode (autopilot or manual control), the throttle signal was used as a basic level fail-save system.

For the fail-save system to be as robust as possible, remote control signal must be received. When the signal data are not available in the control loop for any reason, autopilot is turned off and a neutral PWM signal (50 % duty cycle) is fed to the motor and the steering servo controllers. Note that this does not mean, that car will actively break. Reverse torque generated by the motor is required for active breaking. However, reverse torque shall be applied only, when vehicle is moving forward, otherwise it would result in backward movement. Therefore after centring the throttle PWM signal, vehicle keeps moving by inertia, until dissipation forces stop the vehicle movement.

When the remote signal is available, throttle signal is continuously analysed. If breaking signal from the remote RC controller is detected, the autopilot is disengaged and the control mode is changed to manual control, so full control over the platform is granted back to the human pilot.

6.2.2 Watchdogs

In case of a failure of PWM generation and throttle control, watchdogs were implemented as an additional safety mean. If one of crucial processes is not alive, PWM signals are forced to neutral. When PWM control process itself is not alive, PWM signal generation is stopped.

6.2.3 Remote relay kill switch

In order to provide independent motor kill switch, a relay with 433 MHz remote control was inserted between Raspberry Pi and motor controller (see Fig. 6.1). When a signal from the remote control is received, the relay opens the motor controller PWM circuit and controller sets the current to the motor to zero. However, as already explained above, this does not cause active breaking. Moreover, a range of the remote control is limited to approximately 60 m, even with an external antenna.

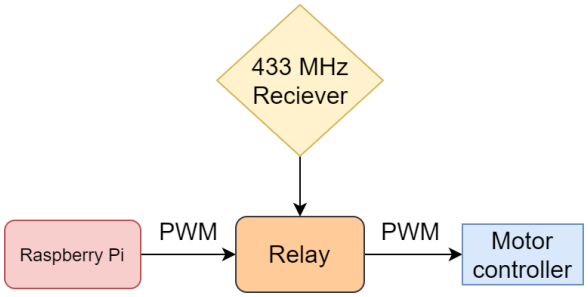


Figure 6.1: Remote relay was used to independently cut off PWM signal to the motor.

6.3 Drive control loop

Integrating semantic segmentation, segmentation map analysis and the control algorithm results in a closed-loop control shown in Fig. 6.2. When an image from the front camera (ZED) is available, it is rectified and forwarded to a neural network. Resulting segmentation map is then used for a surface tracking. Surface tracking algorithm including steering and throttle regulation is presented in the following section.

Figure 6.2: Block diagram of surface tracking control loop

6.3.1 Surface tracking

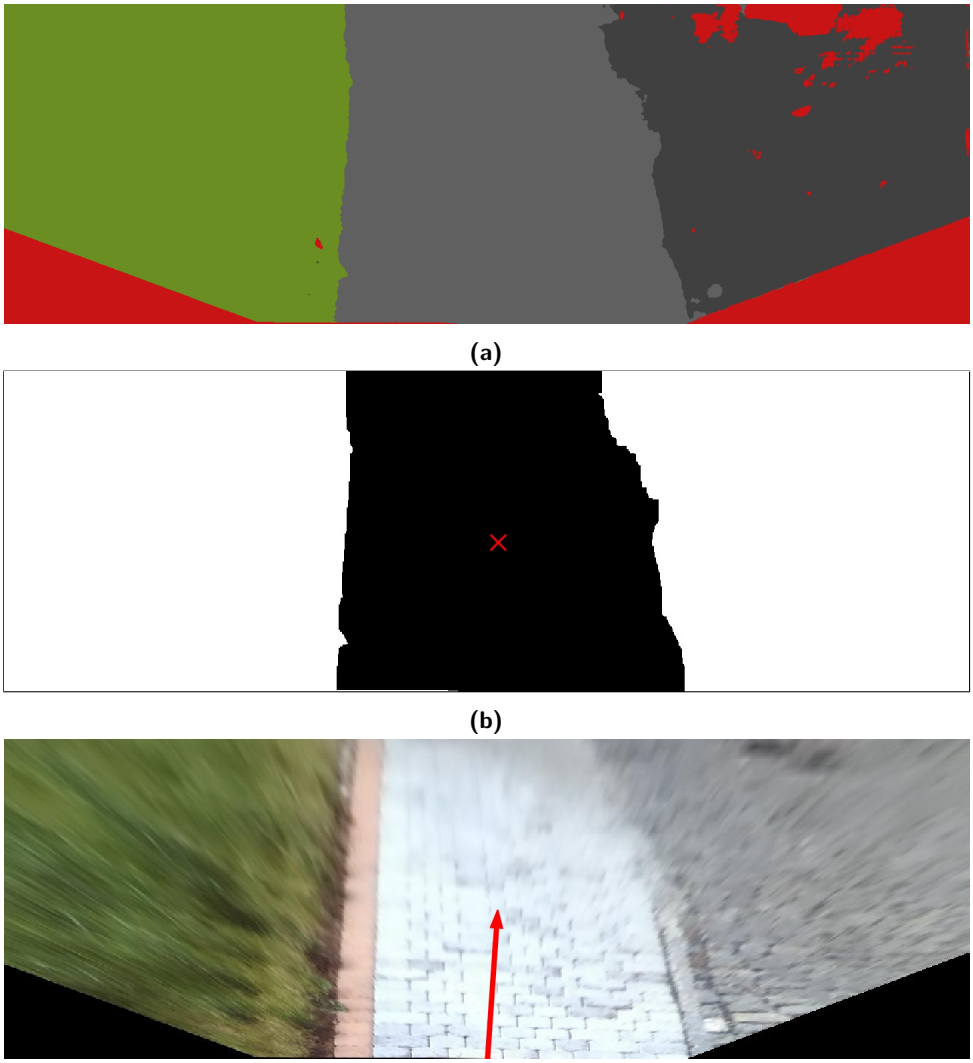
For a surface tracking, fast and simple algorithm based on geometrical centre of area was implemented. Surface tracking intermediate steps are illustrated on an example in Fig. 6.3 and the pseudo code is in Alg. 1.

First, the map is transformed into a binary image to separate tracked surface from all the other surfaces. Only the biggest connected component of a tracked surface is preserved. Then, morphological opening is applied to segmentation map in order to remove small areas of misclassified pixels. The result is used for computing geometrical centre of the area (in Fig.6.3b). Resulting coordinate vector is used as heading (in Fig. 6.3c), in which the platform should move. The difference (normalised to image size) from image centre line (in x-axis) and the difference from bottom of the image are used as

controller references. P controllers were designed for regulation and resulting control action is linearly mapped to PWM command.

Algorithm 1 Control loop

```
1: procedure getControlPWM
2:     segMap ← binaryOpening(segMap)
3:     segMap ← biggestConnectedComponent(segMap)
4:     CoM ← centreOfMass(segMap)
5:     headingX, headingY ← getHeading(CoM)
6:     ctrlSteer ← PID(headingX)
7:     ctrlThrottle ← PID(headingY)
8:     pwmSteer ← tooPWM(ctrlSteer)
9:     pwmThrottle ← toPWM(ctrlThrottle)
10:    return pwmSteer, pwmThrottle
```



(a) **(b)** **(c)**

Figure 6.3: Tracked surface area is separated from a segmentation map (a). Morphological opening is applied to a binary map of a tracked surface and the centre of mass is computed (b). Heading vector (c) is computed from the centre of mass and is forwarded to P regulators.

Chapter 7

Semantic Segmentation Results

In this chapter, results of neural network training and surface tracking drive experiments are shown and effects of proposed solutions on results are analysed.

Major criterion for evaluating trained models was accuracy according to Eq. (5.7). Models with different training settings were trained until convergence and the model with highest accuracy on validation set was chosen to be used in the drive experiment. Final model was also tested on the test set. Note that model training was considered converged after 200 epochs of no improvement of the validation data set accuracy. The training and validation sets were randomly split for every learning process, but the test set remained fixed for all experiments. Learning with each hyper-parameter set was repeated for $n = 5$ trials with constant hyper-parameters. All weights of convolutional layers were initialised randomly using uniform Kaiming (He) initialisation [HZRS15]. The repeated experiment was necessary for a solid result analysis, since the accuracy varied in some cases significantly. If not stated otherwise, default hyper-parameters (see Tab. 7.1) were used for all experiments.

Due to high computational time demands (see Sec. 5.7), exhaustive grid search over all hyper-parameters was not possible. Instead, all hyper-parameters were considered orthogonal and tuned independently.

In following sections, effects of different learning rate setting, batch size, input image size, pixel-wise weight m parameter, focal loss γ parameter and number of synthesised images added to training set are discussed. Note that

Learning rate	Batch size	Image size	m	γ	Synth. data
10^{-3}	5	960 px × 320 px	1	0	0

Table 7.1: Default hyperparameters set: Learning rate, batch size, input image size, pixel-wise weighting m parameter, focal loss γ parameter and number of synthesised images added to training set.

all figures showing accuracy scores are boxplots depicting following statistics: Box presents lower and upper quartile with a colour line at median. The whiskers extend from box by value of $Q_1 - 1.5(Q_3 - Q_1)$ for lower whisker and $Q_3 + 1.5(Q_3 - Q_1)$ fo upper whisker, where Q_i is an i-th quartile. All values outside whiskers range are plotted as outliers separately.

To provide further insight on the distribution of individual image accuracy, cumulative histograms are provided. Histograms are computed on all images from all trials ($n = 5$ repetitions) in the experiment trained with fixed hyper-parameters. In another words, cumulative histograms present fraction of images having lower or equal accuracy than given among all trials.

7.1 Learning rate

First, effect of the learning rate was evaluated. As Fig. 7.1 shows, values close to default value ($\mathrm{lr} = 10^{-3}$) perform similarly. On the other hand, too large or too low learning rates resulted in significant accuracy drop.

Since Adam optimiser adapts learning rate automatically based on data, it was expected to be less sensitive to learning rate setting. In this experiment, $\mathrm{lr} = 5 \cdot 10^{-2}$ performed slightly better. However, in following experiments the best results were achieved with the default value. Thus, default value is used in following experiments.

7.2 Batch size

Due to model memory requirements, available training batch sizes are limited, see Tab. 7.2. This experiment evaluates effect of a batch size (which is dependent on the input image size) to the validation set accuracy.

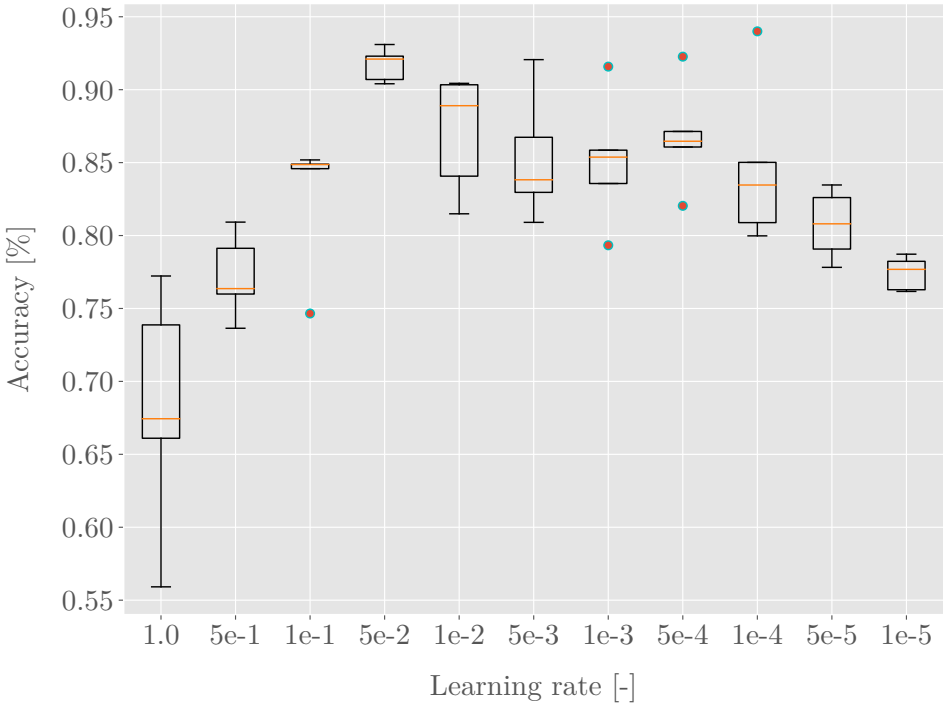


Figure 7.1: The effect of learning rate on validation set accuracy.

Input size	Maximum batch size
1280 px × 424 px	2
960 px × 320 px	5
640 px × 216 px	12

Table 7.2: Maximum available batch size

Experiment results in Fig. 7.2 clearly shows, that increasing batch size to 5, while sacrificing image resolution, improves accuracy. Since further batch size increase (and inevitably decreasing image resolution) causes higher variance, the resolution 960 px × 320 px with batch size 5 was chosen for further experiments and final hyper-parameter set.

This experiment showed, that learning on smaller input data can achieve higher accuracy due to larger batch size available, but only till some extent.

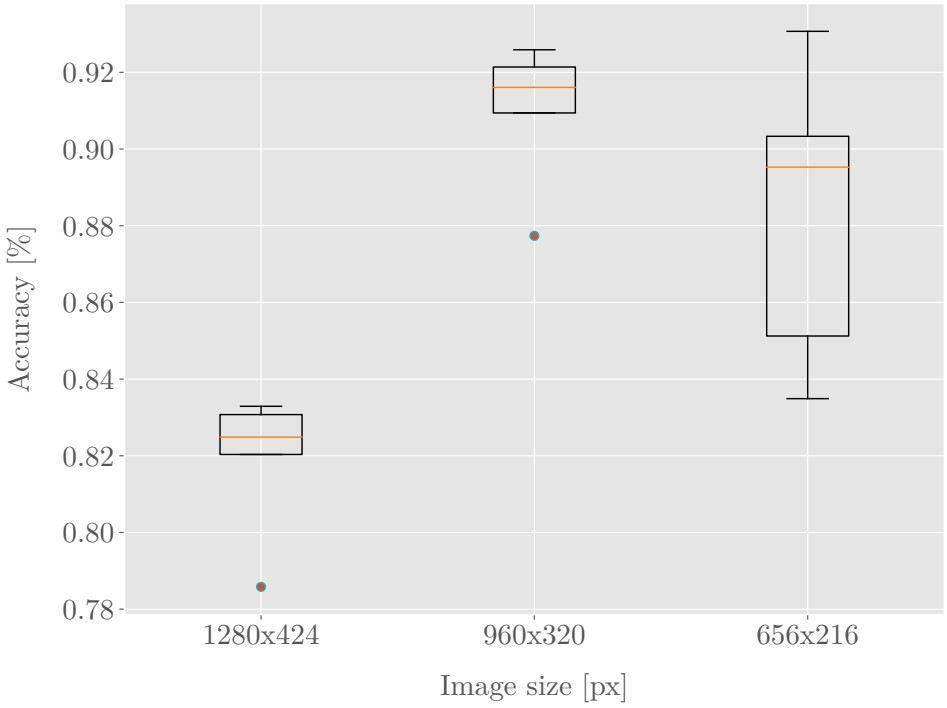


Figure 7.2: The effect of an image size and batch size and input image size on validation accuracy. Batch size mas set to maximum possible according to Tab. 7.2.

7.3 Pixel-wise Weighting

As explained in Sec. 5.3.1, pixel-wise weighted loss was implemented in order to improve segmentation in lower part of the image corresponding to the surface close to car. In Fig. 7.3 there is measurable decrease or variance with decreasing m parameter (defined in Eq. (5.4)), but no significant improvement in overall accuracy was observed.

To provide an insight to the spatial distribution of accuracy, error maps are presented. Error maps are the same size as validation (or testing) images and each pixel is an average error over the training set images over the whole batch, namely

$$e(\mathbf{Pred}, \mathbf{True}) = \frac{1}{M} \sum_{n=0}^{M} [\mathrm{True}_m = \mathrm{Pred}_m], \tag{7.1}$$

where e is error map pixel value, Pred is vector of predicted classes for pixel of all images (vector length is M), True is vector of true classes for pixel of all images (vector length is M) and M is number of validation (testing) images.

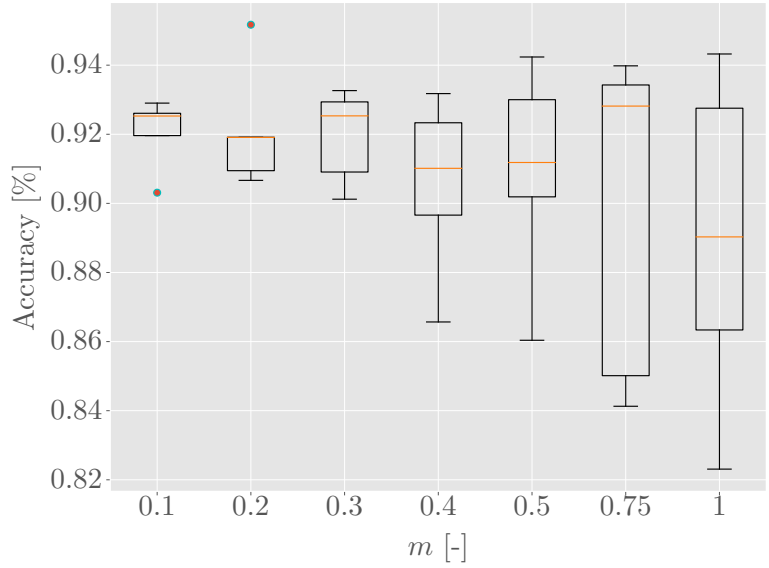
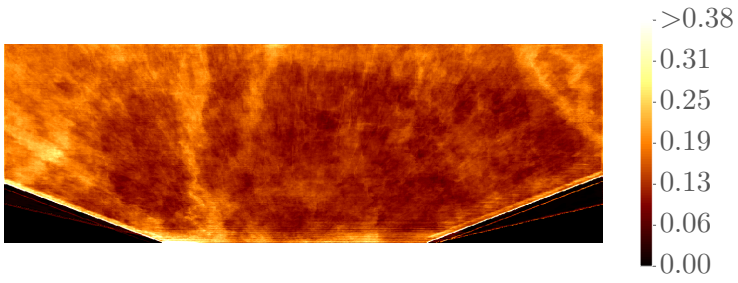


Figure 7.3: The effect of a pixel-wise weighting parameter m on validation accuracy. Lower m decreases the accuracy variance, however does not significantly increase mean.

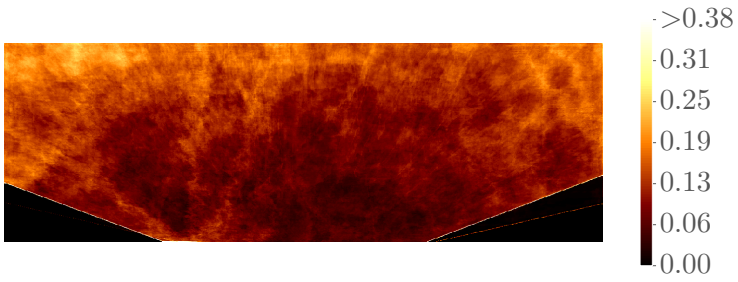
On error maps in Fig. 7.4, the effect is clearly visible. The accuracy is higher in the region closer to the vehicle with exponential weighting compared to uniform weights.

Cumulative histograms are presented in Fig. 7.6. It is seen that the model trained with uniform weights (corresponding to $m = 1$) has higher occurrence of images of very low accuracy compared to exponential weights. As an example, for model trained with uniform weights 6.5 % of images achieved 50 % or lower validation accuracy. On the other hand, there is only about 3 % of images with validation accuracy 50 % or lower for model trained with exponential weights ($m = 0.2$)

Since parameter $m = 0.2$ caused the lowest fraction of poorly classified images in comparison with other parameters (see detail in Fig. 7.6), it was used further in learning process.

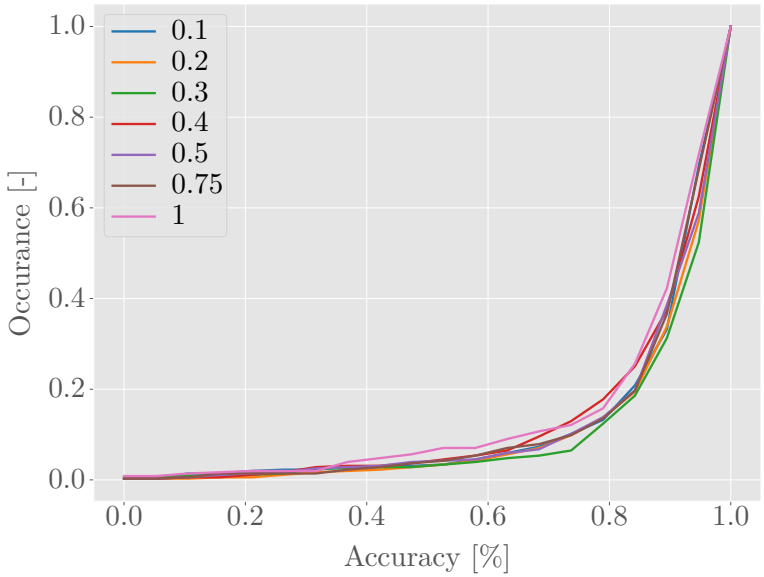


(a): Uniform weights ($m = 1.0$)

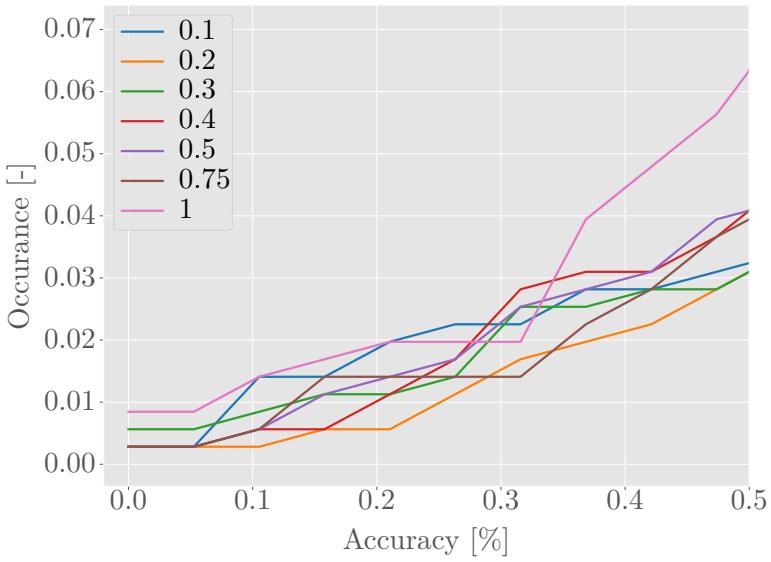


(b): Exponential weights ($m = 0.1$)

Figure 7.4: Error maps of two m parameters showing the effect of pixel-wise weighting.



(a)



(b)

Figure 7.6: Effect of pixel-wise weighting m parameter on poorly classified images. Comparing to uniform weights, exponential weights showed lower fraction of poorly classified images.

7.4 Synthesised data

Train set was extended with various number of synthesised images to improve learning process. Fig. 7.7 shows, that extending train set by a large number of synthesised images has negative effect on accuracy. Similarly, extending the train set has negative impact on number of poorly classified images (see Fig. 7.8).

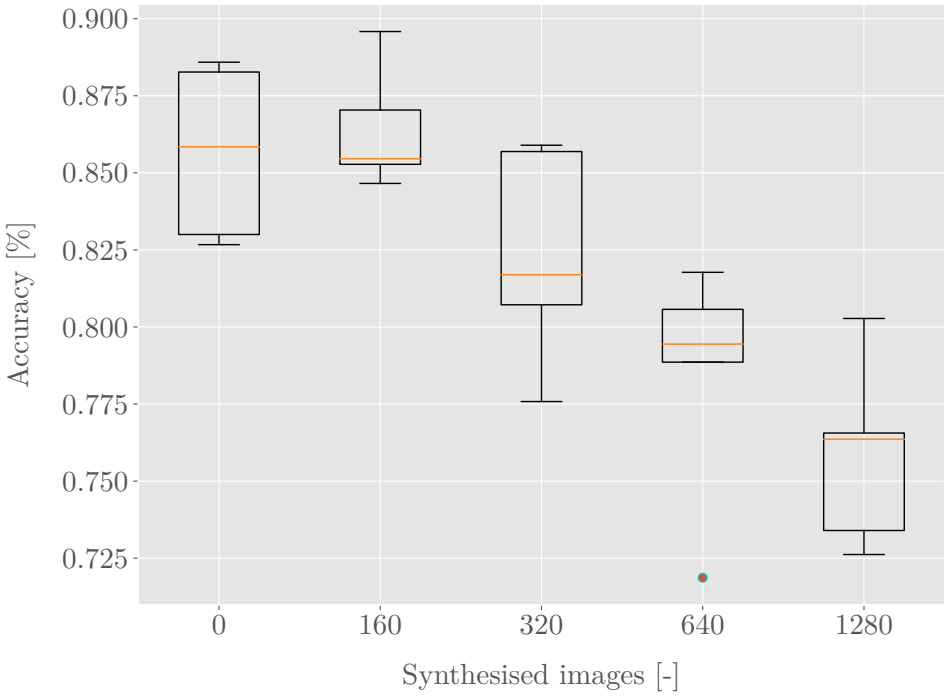


Figure 7.7: Validation accuracy of a models trained with train set extended by various number of synthesised images.

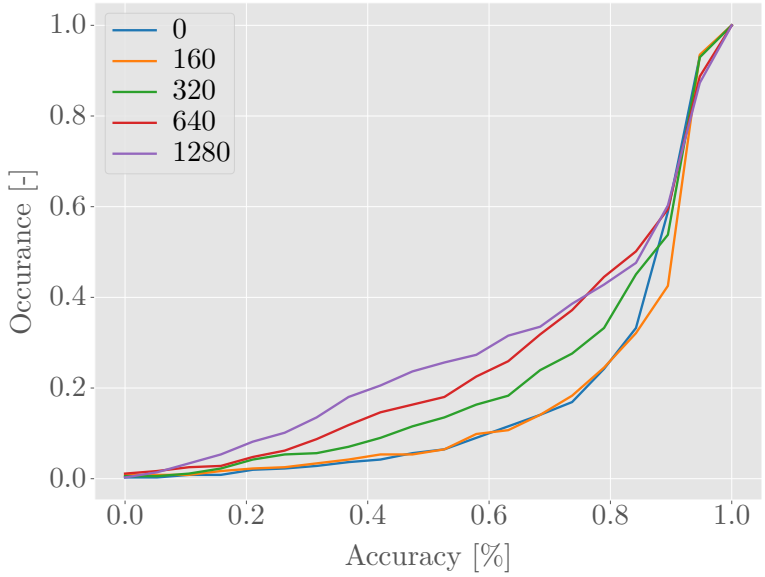


Figure 7.8: Accuracy cumulative histogram of a models trained with train set extended by various number of synthesised images. .

7.5 Focal Loss γ

As seen in Fig. 7.9, introduction of Focal Loss did not improve accuracy significantly on validation data, but there is noticeable increase of low accuracy outliers for larger γ.

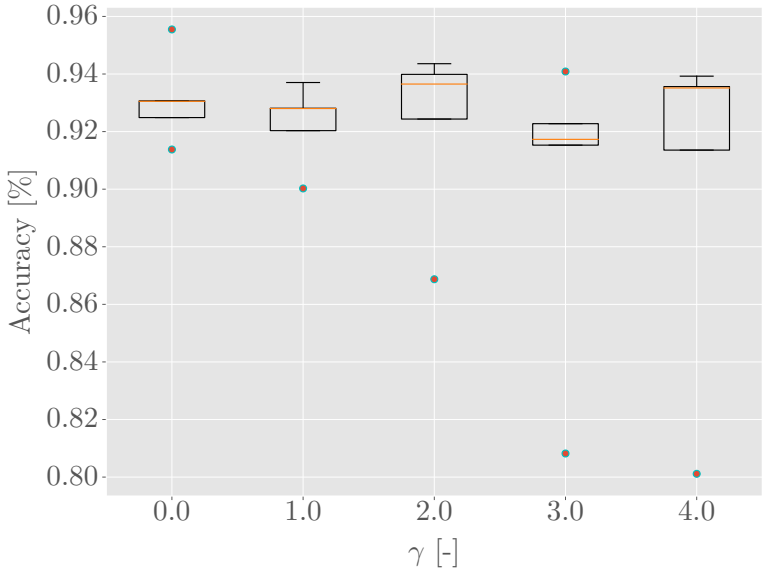
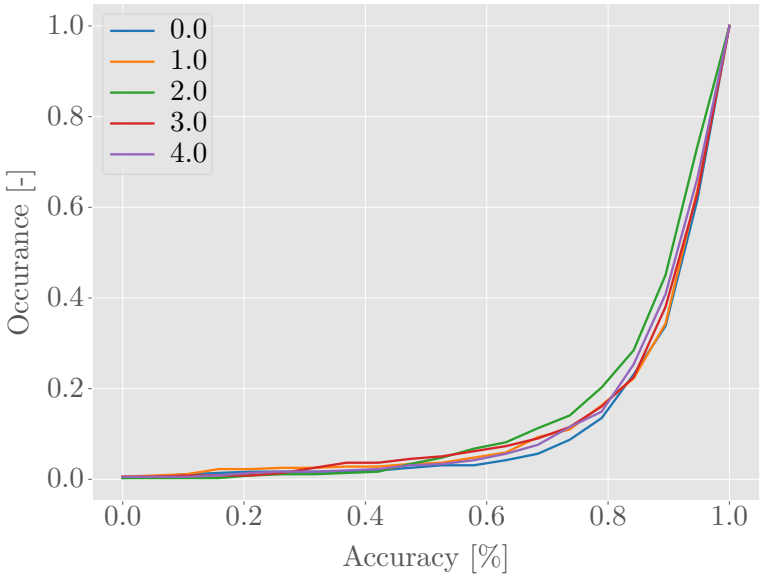
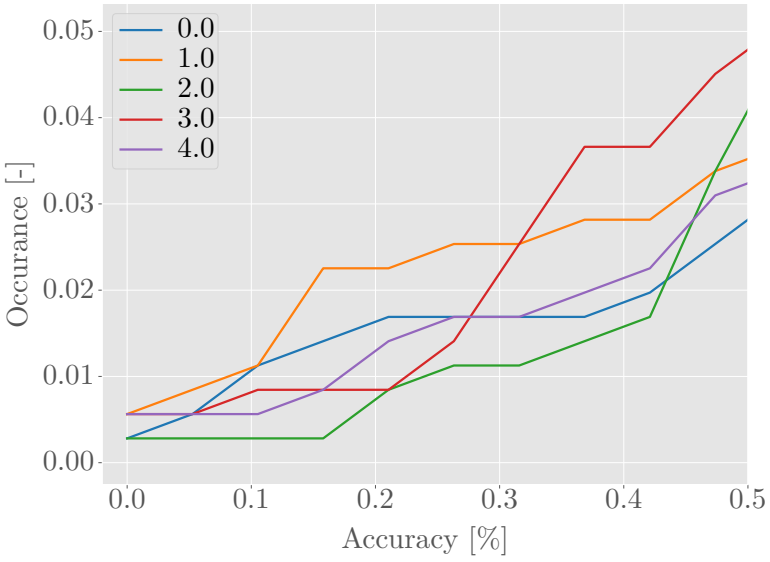


Figure 7.9: Validation accuracy boxplot for Focal Loss γ

On cumulative histogram of image accuracies (in Fig. 7.10) there is a slight effect of elimination of poorly classified images. Especially on detailed look, there is a noticeable decrease of images with classification accuracy under approximately 35 % for $\gamma = 2$. Interestingly, $\gamma = 2$ was also recommended in the original paper [LGG+17] for a different problem (object detection).



(a)



(b)

Figure 7.10: Cumulative histogram of validation accuracies for Focal Loss γ

7.6 Final model

Final model was selected among all models trained in experiments with respect to accuracy, but considering also performance on poorly classified images. The best model was trained with parameters listed in Tab. 7.3 and achieved 94.4 % accuracy on validation set and 92.7 % accuracy on the test set.

In order to show effect of pixel-wise weighting and focal loss, final model is compared with baseline model trained with default parameters (see Fig. 7.1). Both models were evaluated using pixel-wise weighted accuracy as defined in Eq. (5.7) with parameter $m = 0.2$ (same as the final model).

As shown in Fig. 7.11, the final model performed significantly better in terms of accuracy than model trained with default hyper-parameters.

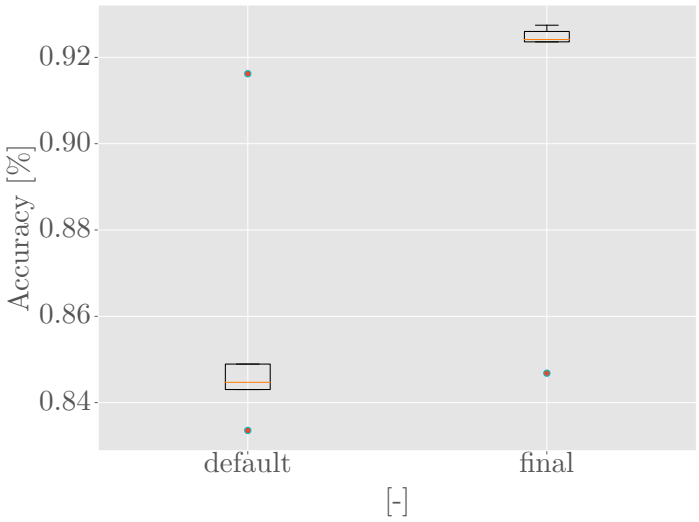


Figure 7.11: Test data set accuracy of base and final model

The final model also performed better on poorly segmented images. There is a significantly lower occurrence of poorly classified images in whole accuracy range, as can be seen in Fig. 7.12.

It is clear from error maps in Fig. 7.13, that final model makes less errors in classification of pixels corresponding to the front of the car, than base model. It illustrates well the effect of pixel-wise weighting loss function.

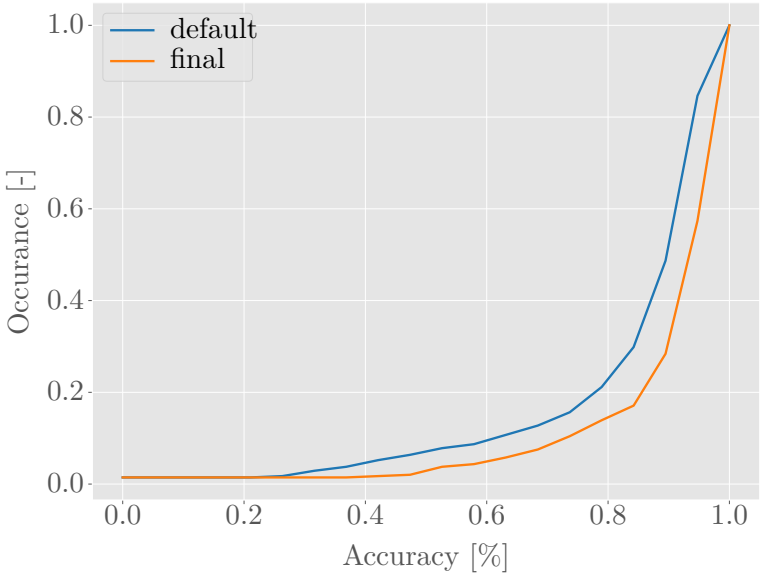


Figure 7.12: Test data set cumulative histogram of base and final model.

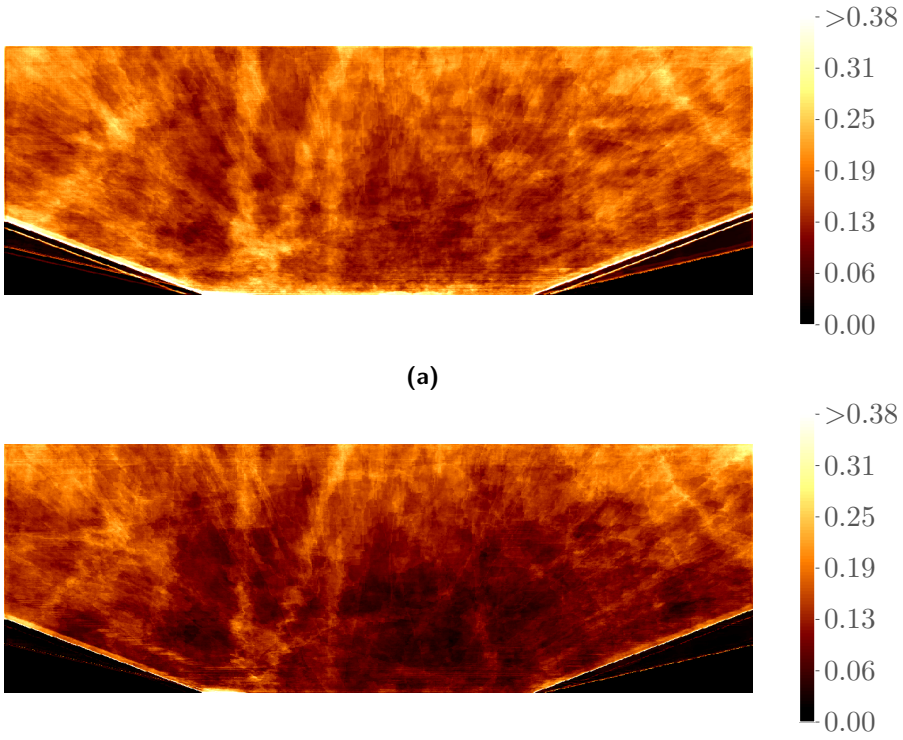


Figure 7.13: Test data set error maps of model with default hyperparameters (a) and final model (b).

Confusion map (see Fig. 7.14) was computed using the same spatial weights as used for training the final model ($m = 0.2$). There is some noticeable

Figure 7.14: Confusion matrix of test data computed for the final model. Matrix is weighted by a weight map for $m = 0.2$ and normalised to percents.

Learning rate	Batch size	Image size	m	γ	Synth. data
10^{-3}	5	960 px × 320 px	0.2	2.0	-

Table 7.3: Final hyperparameters set

correspondence between class distribution and class true positive rate (*grass* and *gravel* classes). However, *asphalt* class was the least represented and still reached above average true positive rate. This suggests, that dataset distribution disbalance was compensated for some classes (*asphalt*) by focal loss. However, *cobblestone* and *interlocking pavement* classes were frequently confused despite focal loss, probably due to their similar appearance.

On Fig. 7.15, there are examples of images and corresponding segmentation maps taken and segmented during experiment drives. The final model was used for segmentation in all experiments.

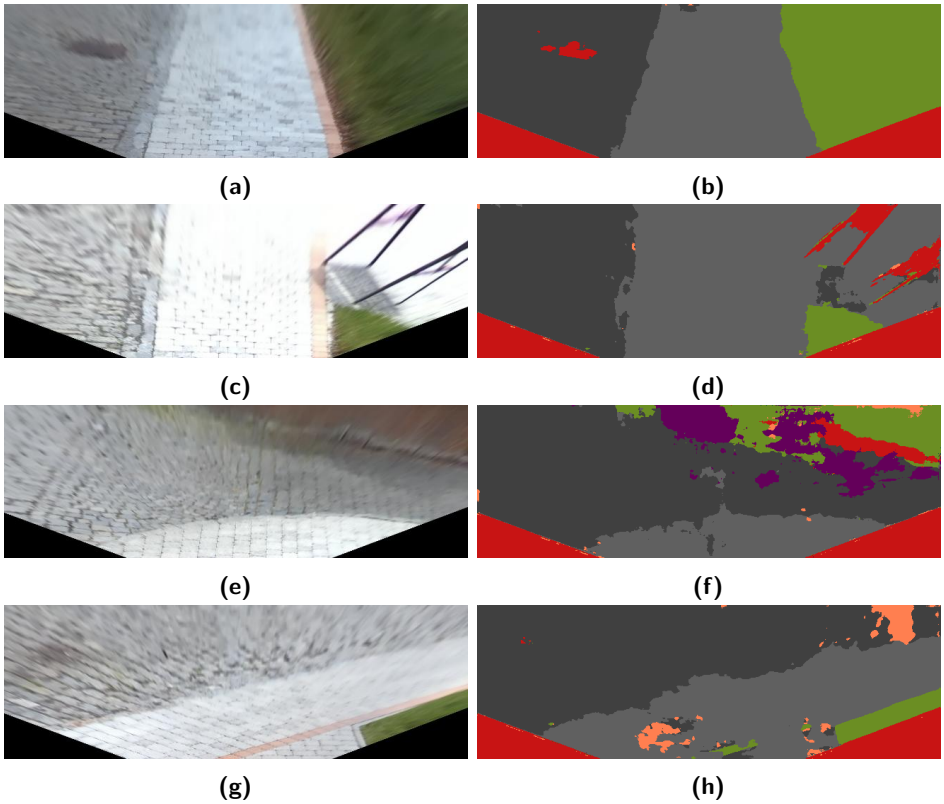


Figure 7.15: Example images corresponding annotation maps taken and segmented with the final model during experimental drives.

Chapter 8

Autonomous Driving Experiment

In order to present reliability and usability of the proposed semantic surface segmentation in a real-world application, autonomous surface tracking experiment was performed on TOMI platform. The experiment was designed to show near real-time capability of reliable surface semantic segmentation. All experiments took place at the yard of the CTU at Karlovo náměstí (see Fig. 8.1).

8.1 Description

Basic idea of the experiment is to follow a path of a given surface and avoid moving over other surfaces. Target surface is determined after the system starts according to the surface right in front of the platform. Several paths of a specific surface were chosen and obstacles (benches) were placed on paths.

Following goals were achieved:

1. Autopilot is disengaged, if fail safe initiated
2. Platform is not exceed maximum speed
3. Platform may slow down temporarily
4. Platform moves over selected surface only

Figure 8.1: Experiments were performed on the university yard. Pavement path was tracked in several route variants.

Image size	Average inference time	FPS
1280 px × 424 px	451 ms	2.2
960 px × 320 px	367 ms	2.7
640 px × 216 px	214 ms	4.7

Table 8.1: Average neural network inference and FPS on Jetson Xavier during the test drive.

5. Platform stops, if movement forward on a selected surface is not possible

8.2 Semantic segmentation and tracking on Jetson Xavier

Both semantic segmentation and surface tracking were performed on Jetson Xavier in all experiments. Average segmentation inference (see Tab. 8.2) did not satisfy real-time requirements for image segmentation. Image size 640 px × 216 px was selected due to its segmentation speed. Including also tracking time (20 ms) and core code, resulting control loop period was approximately 250 ms (4 Hz). That was enough for basic vehicle control at low speed. However, for control at higher speeds, more computational performance or simpler neural network architecture is necessary.

As stated before, segmentation itself causes considerable latency by itself. To estimate total latency from the time of image capture to the time of PWM

Image size	Total latency estimate
1280 px × 424 px	567 ms to 601 ms
960 px × 320 px	483 ms to 517 ms
640 px × 216 px	330 ms to 364 ms

Table 8.2: Total estimated latency from the image capture to PWM signal generation.

generation, following delays need to be considered: Tracking algorithm takes approximately 20 ms. According to the ZED API reference, communication time introduces additional 2 or 3 frame-time delay (for 30 FPS 66 to 100 ms). Additional delays from various sources (rectification, USART communication, process sleep time, ...) were empirically estimated to 30 ms. Note that image capturing runs in separate thread and thus delay time contributes to latency, but does not affect control loop period, which is shorter.

In our experiment, maximum speed was limited to approximately $1\,\mathrm{m\,s^{-1}}$. But for control at higher speeds, latency becomes more important factor to be considered.

8.3 Results

Unfortunately, mounted GPS modules do not provide sufficient accuracy for experimental drive evaluation. Therefore, experiment evaluation is limited only to recorded visual data and sensor data logs.

Example of an experimental drive is illustrated on superimposed motion picture in Fig. 8.2. Recording of selected experiment from different points of view is attached to this work (see Appendix A). Data logs on Fig. 8.3 show steer and throttle PWM pulse duration and heading γ estimated from gyroscope data.

All experiments were repeated several times. Fig. 8.4 contains logged data from three drives on the same route. Note that neutral PWM pulse duration is 1419 ns for steering and 1520 ns for throttle.

Fig.8.5 shows data logged during an experiment while performing 90-degree turn. There is clear throttle signal decrease, resulting in lower speed. However, during the experiment car stopped completely for a moment (as

Figure 8.2: Autonomous drive experiment. The TOMI platform captured every 5 s and composed into a picture from a video (by Jan Čech). See attached video of the experiment.

seen on attached video). Author believes, it was caused due to the motor control. During low speed steering, more torque is needed to maintain the same speed, in comparison with forward movement. But currently, there is no velocity regulator implemented, because RPM sensor does not provide reliable data in low speeds. This causes undesired velocity decrease while steering.

Note that the control frequency of about 4 Hz is visible, especially on the steer PWM reference graph.

Examples of computed centre of mass and heading vector from experimental drives are shown in Fig. 7.15.

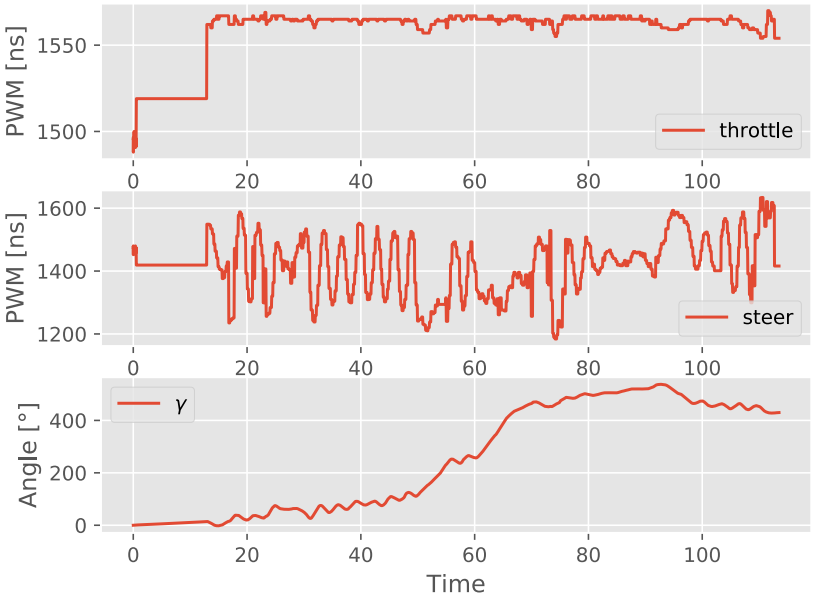
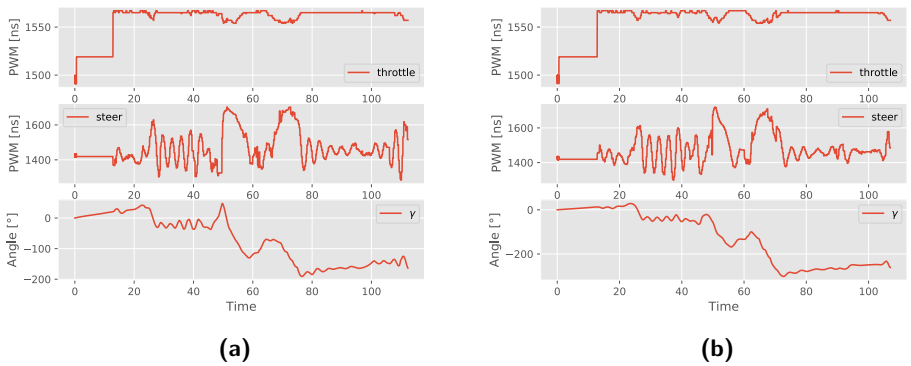


Figure 8.3: Example of data logged during experimental drive. Steer and throttle PWM reference signals and γ estimation is selected.



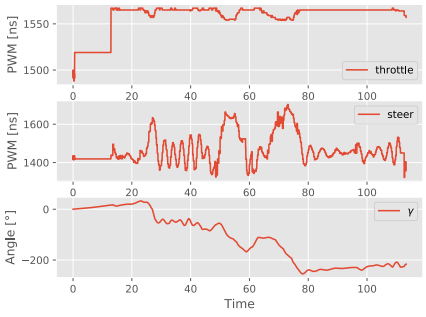


Figure 8.4: Data logged during repeated experiment on the same path.

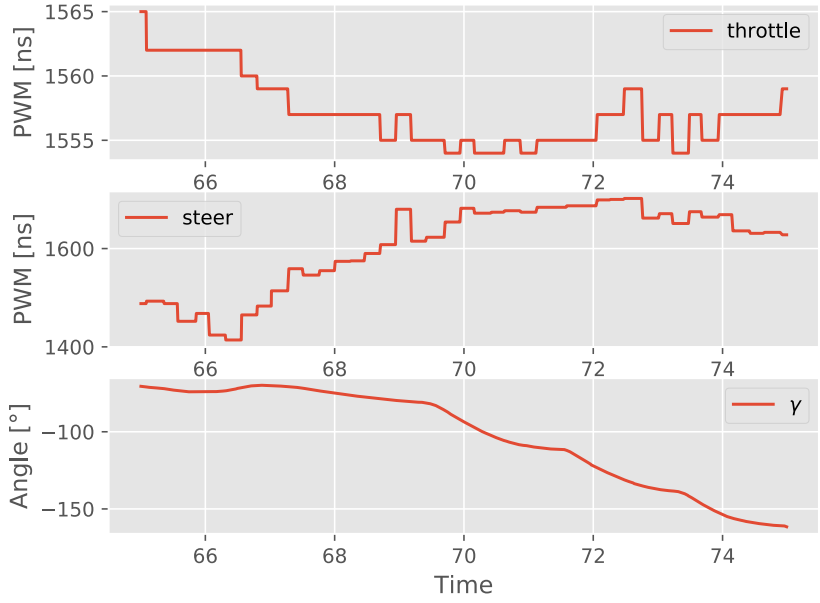


Figure 8.5: Data logged during 90 degree turn.

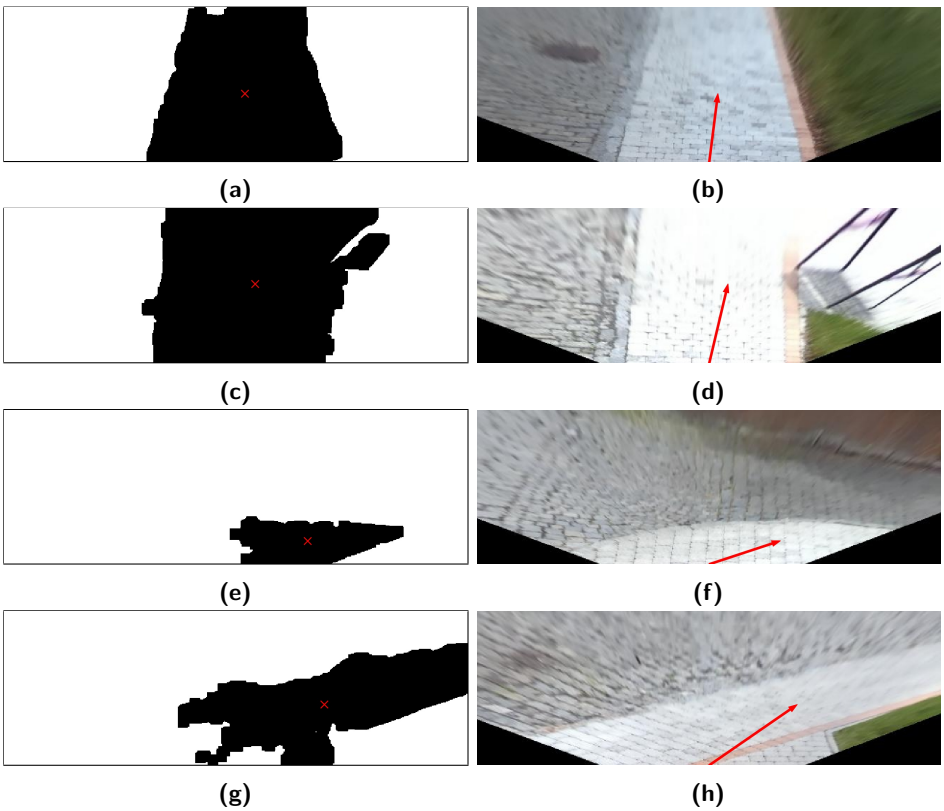


Figure 8.6: Examples of computed centre of mass and heading vector. Images and segmentation maps were captured and segmented using the final model during experimental drives and are identical to 7.15

Chapter 9

Conclusion/Summary

As presented in Chap. 7, reliable semantic segmentation system was proposed and tested in experimental autonomous drives. The experiment showed, that semantic segmentation performed well enough to enable segmentation-based navigation.

This was achieved mainly due to pixel-wise weighting, which improved system spatial accuracy on important areas (closer to the vehicle) while keeping overall accuracy. Secondly, poorly segmented images were partially eliminated by using the Focal loss. This is another feature increasing robustness. Altogether, proposed system showed its perspective in surface semantic segmentation applications. On the other hand, training with data set extended by synthesised data did not introduce noticeable accuracy improvement. Author of this work suspects that synthesised images were not authentic enough.

Surface tracking drive experiments show the segmentation system was reliable enough to provide data necessary for autonomous surface tracking on known surfaces in known environment. This holds true even for lower image resolution (for experiments 640 px × 216 px image size was used). Platform was able to follow a target surface with certain imperfections. Temporary crossings onto other surfaces were caused mainly by trivial planing algorithm and suboptimal control.

However, it was not analysed, how different weather conditions or different environments affect segmentation. Data set of manually annotated images is limited to a small number of locations and basically lacks variance in weather

or ambient conditions. This is a limitation, which may require additional work to be done on further development of the system.

Camera is fixed to the vehicle top cage. This may cause significant camera tilt during accelerating and decelerating, resulting in camera point of view change. Homography used for rectification is then inaccurate. This problem can be eliminated by using a gimbal.

Segmentation accuracy was showed high enough to be used for autonomous vehicle drive. However, inference speed is far from real-time. In order to speed up inference while keeping accuracy, other neural network architecture may be used. Promising option is also converting trained model to a format supporting NVIDIA TensorRT library [Ten].

As drive experiments showed, tracking algorithm lacks the ability to avoid obstacles. This causes algorithm to ignore small obstacles in the path, if they are not large enough to move the centre of mass. Better results may be accomplished by using advanced planing algorithm in combination with more advanced control technique (predictive regulators, ...). Necessary first step is to implement proper inner-loop speed regulator, as mentioned in Sec. 8.3.

As a future improvement, independent RC transceiver and receiver system is proposed as a kill switch to replace current 444 MHz short range remote control relay. To eliminate risk of out-of-range failure, kill switch should be also activated whenever signal is not being received for any reason.

Appendix A

Attachments

Following files are attached to this work:

- Source code (in one archive)
- Video from the experiment

Appendix B

Bibliography

[BFC09] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla, *Segmented video preview image*, 2009, [Online; accessed May 8, 2020].

[BFP+20] Michal Bahník, Dominik Filyo, David Pekárek, Martin Vlašimský, Jan Čech, Tomáš Haniš, and Martin Hromčík, *Visually assisted anti-lock braking system*, IEEE Intelligent Vehicles Symposium (IV), 2020, To Appear.

[Bis06] Christopher M. Bishop, *Pattern recognition and machine learning (information science and statistics)*, p. 229, Springer-Verlag, Berlin, Heidelberg, 2006.

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **39** (2017), no. 12, 2481–2495.

[Bré17] Amaury Bréhéret, *Pixel Annotation Tool*, `https://github.com/abreheret/PixelAnnotationTool`, 2017.

[DGHC19] Arjun Desai, Garry Gold, Brian Hargreaves, and Akshay Chaudhari, *Technical considerations for semantic segmentation in mri using convolutional neural networks*, 02 2019.

[HWH+19] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu, *Ccnet: Criss-cross attention for semantic segmentation*, 2019 IEEE/CVF International Conference on Computer Vision (ICCV) (2019).

[HZRS14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Spatial pyramid pooling in deep convolutional networks for visual recognition*, Lecture Notes in Computer Science (2014), 346–361.

[HZRS15] ———, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015 IEEE International Conference on Computer Vision (ICCV) (2015).

[Jet] *Jetson AGX Xavier Developer Kit | NVIDIA Developer*, `https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit`.

[KB14] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2014.

[LGG+17] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár, *Focal loss for dense object detection*, CoRR **abs/1708.02002** (2017).

[MHZ19] Lichao Mou, Yuansheng Hua, and Xiao Xiang Zhu, *A relation-augmented fully convolutional network for semantic segmentation in aerial scenes*, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019).

[MLS18] Andres Milioto, Philipp Lottes, and Cyrill Stachniss, *Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in cnns*, 2018 IEEE International Conference on Robotics and Automation (ICRA) (2018).

[NSD+20] Ruigang Niu, Xian Sun, Wenhui Diao, Kaiqiang Chen, and Kun Fu, *Hmanet: Hybrid multiple attention network for semantic segmentation in aerial images*, 2020.

[RDN18] Alexander Rakhlin, Alex Davydow, and Sergey Nikolenko, *Land cover classification from satellite imagery with u-net and lovász-softmax loss*, 06 2018, pp. 257–2574.

[RFB15a] O. Ronneberger, P. Fischer, and T. Brox, *U-net architecture (example for 32x32 pixels in the lowest resolution).*, Reproduced from [RFB15b], 2015.

[RFB15b] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, Medical Image Computing and Computer-Assisted Intervention (MICCAI), LNCS, vol. 9351, Springer, 2015, (available on arXiv:1505.04597 [cs.CV]), pp. 234–241.

[SGA+18] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, M. Jagersand, and H. Zhang, *A comparative study of real-time semantic*

segmentation for autonomous driving, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2018, pp. 700–70010.

[Ten] *NVIDIA TensorRT | NVIDIA Developer*, `https://developer.nvidia.com/tensorrt`.

[UA19] Irem Ulku and Erdem Akagunduz, *A survey on deep learning-based architectures for semantic segmentation on 2d images*, 2019.

[VRC+16] Francesco Visin, Adriana Romero, Kyunghyun Cho, Matteo Matteucci, Marco Ciccone, Kyle Kastner, Yoshua Bengio, and Aaron Courville, *Reseg: A recurrent neural network-based model for semantic segmentation*, 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (2016).

[WZL+19] M. Wu, C. Zhang, J. Liu, L. Zhou, and X. Li, *Towards accurate high resolution satellite image semantic segmentation*, IEEE Access **7** (2019), 55609–55619.

[YLCT20] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda, *A survey of autonomous driving: Common practices and emerging technologies*, IEEE Access **8** (2020), 58443–58469.

[ZED] *ZED Stereo Camera | Stereolabs*, `https://www.stereolabs.com/zed/`.