# CHARLES UNIVERSITY

## Faculty of Science

Department of Applied Geoinformatics and Cartography

Study program: Geography
Curriculum: Cartography and Geoinformatics




Bc. Jakub RŮŽIČKA


# Automatic Detection of Driving Lanes Geometry Based on Aerial Images and Existing Spatial Data

**Automatizované odvození geometrie jízdních pruhů na základě leteckých snímků a existujících prostorových dat**

*Master thesis*


Supervisor: RNDr. Lukáš Brůha, Ph.D.



Prague 2020

**Declaration**

Hereby I declare that I have written this thesis myself and that I quoted all references accordingly. Neither this thesis nor its part was used to obtain other academic degree of the same or other levels.

I hereby agree with using this thesis for academic purposes and I agree with its inclusion in the evidence of academic works.

In Prague on 4 May 2020 . . . . . .

Jakub Růžička

***Acknowledgment***

*Hereby I would like to thank the thesis supervisor, RNDr. Lukáš Brůha, Ph.D., for his support, helpful advice and shared knowledge without which it would not be possible for me to write the thesis.*

*I would not be able to test the proposed method without relevant data; therefore I thank the Prague Institute of Planning and Development for their kind provision of aerial images as well as to the whole OpenStreet Map community for their contribution to VGI.*

*The thesis would not have been written without Prof. Dr. Ioannis Giannopoulos, Head of Research Group, Department of Geodesy and Geoinformation of Technical University in Vienna, whose seminar ignited my interest in the field of autonomous driving and use of GIS in traffic infrastructure.*

*I would like to thank my friends from EGEA (the European Geography Association for students and young geographers) and especially to my colleagues from the Board of EGEA 2019/2020 (Ksenia Simonova, Jeroen Royer, Johanna Zempel and Merli Ilves) for their support and understanding in times when I was working on the thesis.*

*Finally, I would like to thank my family for the motivation and support they were giving me all the time throughout my studies and to my friends who advised me, supported me and kept my spirits high. I would not have finished without you.*

*This thesis was written in the difficult times of the worldwide COVID-19 pandemic. Thus, I would like to dedicate this thesis to everyone who took their part in fighting against the disease. This includes paramedics, medical staff, fire fighters, police and other people in the front line as well as everyone who volunteered and offered their help in order to overcome the pandemic.*

# Abstrakt

Cílem této práce je vytvoření metody odvození geometrie jízdních pruhů na základě leteckých snímků a existujících prostorových dat. Navržená metoda používá současně dostupná data, ve kterých identifikuje vodorovné dopravní značení (VDZ). Polygony, které jsou klasifikovány jako VDZ, jsou následně zpracovány jedním z navržených algoritmů, který vytvoří jejich liniovou reprezentaci (vektor), která je jedním z dílčích výsledků. Tyto linie jsou dále analyzovány a na jejich základě dochází k vytvoření linií symbolizujících hranice mezi jednotlivými jízdními pruhy, které představují druhý dílčí výsledek. Kromě toho je snaha o automatizované rozlišení mezi plnou a přerušovanou čárou, což přináší větší informační hodnotu vytvořeného datového souboru.

Navrhnuté algoritmy byly otestovány ve 20 zájmových územích a výsledky testování jsou uvedeny v této práci. Celková správnost a stejně tak i prostorová přesnost testovaných dat dokazuje, že navrhovaná metoda je efektivní. V průběhu testování byly identifikovány určité nedostatky navrhovaného procesu, které jsou v textu blíže popsány, stejně tak je v textu navrženo jejich eventuální řešení. Práce je doprovázena více než 70 obrázky, které ilustrují text a přinášejí jasnější pohled na probíraná témata.

Práce je rozdělena na následující kapitoly: nejprve Úvod a Přehled aktuální literatury, které přinášejí širší kontext analyzovaného problému, autorovu motivaci a současný stav řešení problému v odborné literatuře. Následně jsou popsány technické a právní normy definující pravidla značení VDZ a stejně tak jsou prezentovány i teoretické postupy použité dále v práci. Dále jsou prezentovány navržené algoritmy spolu s citacemi kódu, komentáři autora a obrázky. Problémy metody a její eventuální nedostatky jsou uvedeny a diskutovány v následné kapitole, stejně jako zhodnocení polohové přesnosti metody. Závěrečnou kapitolou je potom Diskuse, která slouží jako všeobecné shrnutí.

**Klíčová slova:** GIS, algoritmus, silnice, silniční infrastruktura, jízdní pruhy, vodorovné dopravní značení, detekce vodorovného dopravního značení

# Abstract

The aim of the thesis is to develop a method to identify driving lanes based on aerial images and existing spatial data. The proposed method uses up to date available data in which it identifies road surface marking (RSM). Polygons classified as RSM are further processed to obtain their vector line representation as the first partial result. While processing RSM vectors further, borders of driving lanes are modelled as the second partial result. Furthermore, attempts were done to be able to automatically distinguish between solid and broken lines for a higher amount of information contained in the resulting dataset.

Proposed algorithms were tested in 20 case study areas and results are presented further in this thesis. The overall correctness as well as the positional accuracy proves effectivity of the method. However, several shortcomings were identified and are discussed as well as possible solutions for them are suggested. The text is accompanied by more than 70 figures to offer a clear perspective on the topic.

The thesis is organised as follows: First, Introduction and Literature review are presented including the problem background, author's motivation, state of the art and contribution of the thesis. Secondly, technical and legal requirements of RSM are presented as well as theoretical concepts and methods used in the process. Furthermore, proposed algorithms are presented together with code samples, author's commentary and figures. Identified problems and potential shortcomings of the method are than presented together with positional accuracy assessment. Finally, the thesis is concluded with Discussion.

**Key words:** GIS, algorithm, roads, road infrastructure, driving lanes, road surface marking, road surface marking detection

# Table of contents

# List of abbreviations

| | |
|---|---|
| AD | autonomous driving |
| ADAS | advanced driver assistance system |
| ČÚZK | Český úřad zeměměřický a katastrální, Czech Office for Surveying, Mapping and Cadastre |
| ČR | Česká republika, the Czech Republic |
| ČSN | České státní normy, Czech national norms (translated by the author) |
| DN | digital number |
| FID | feature identifier |
| GIS | geographic information system |
| GNSS | global navigation satellite system |
| HW | hardware |
| LEGION | locally excitatory globally inhibitory oscillator networks |
| LIDAR | light detecting and ranging |
| OBIA | object-based image analysis |
| RANSAC | random sample consensus |
| RMSE | root mean square error |
| RTK | real-time kinematic |
| ŘSD | Ředitelství silnic a dálnic, Road and Motorway Directorate of the Czech Republic |
| RSM | road surface marking |
| SW | software |
| TP | Technické podmínky, Technical norms (translated by the author) |
| VGI | volunteered geographic information |
| ZABAGED® | Základní báze geografických dat, Fundamental Base of Geographic Data of the Czech Republic |
| ZM10 | Základní Mapa 1 : 10 000, Base Map 1 : 10 000 (translated by the author) |

# Figures and Tables

## 1.1. List of figures

*If not stated otherwise, all figures were created by the author and show data as referenced in References.*

## 1.2. List of tables

# 1. Introduction

High resolution spatial data are a key element of modern Geoinformatics. They allow us to do precise analyses up to sub-object level such as identifying a location within a building or determining the exact train location up to track level on a multi-track railway. Not only technical solutions with sub-meter accuracy in location measurement are needed to identify the exact location of things; precise spatial data with high resolution with similar scale are necessary to use the positional accuracy which modern methods can offer. There are several specific fields such as cadastre, safety, navigation or traffic management, where high resolution spatial data are needed. Especially in recent era and in the future, high precision navigation, automation of traffic control or autonomous driving will need to be able to determine the exact location of vehicles on the road up to single driving lanes or of trains on railway tracks.

Current methods of positioning using global navigation satellite systems (GNSS) such as real-time kinematic (RTK) method are able to determine the position up to centimetre-level accuracy. These positioning methods are getting widely available as their price lowers. Thus, it is not a problem anymore to measure accurate location; however, getting data with the same level of detail remains problematic. For instance, there is no openly accessible geospatial dataset on the Czech market which would offer sub-object accuracy of buildings or roads. No better result would one get if searching among international products. For institutions these products mean a fortune as their creation is an extensive process with high acquisition costs.

Focusing on road infrastructure, there is vast amount of existing spatial data in both Czech and international context; however, none of these open datasets have sub-object level of detail. In Czech national context the open public data are managed by the Czech Office for Surveying, Mapping and Cadastre (Český úřad zeměměřický a katastrální, ČÚZK) and the Road and Motorway Directorate of the Czech Republic (Ředitelství silnic a dálnic ČR, ŘSD). ČÚZK manages the Fundamental Base of Geographic Data of the Czech Republic (Základní báze geografických dat, ZABAGED®) which is the Czech national geospatial database and contains (among others) vector data of traffic infrastructure. Roads are characterised with a centreline (for divided multi-way roads such as motorways one centreline per roadway) with positional accuracy up to 5 meters (ČÚZK 2018). ŘSD manages its own database of road infrastructure. However, even though this is supposed to be a specialised road infrastructure database, it does not contain roads network with the scale of traffic lanes. Even worldwide projects based on volunteered geographic information (VGI) such as OpenStreet Map do not use sub-object level of detail.

To obtain the geospatial information of road infrastructure with a sub-object level of detail, existing road network data have to be combined with a data source in which traffic lanes can be separated from each other. One way to do so would be using GPS tracks and on board sensors of vehicles on the road to identify driving lanes; this process was used to create high definition maps for testing highly autonomous driving (Aeberhard et al. 2015). Yet, this mapping method would be very challenging and costly when large-scale areas should be mapped in this way. Furthermore, such data would not necessarily represent the correct driving lanes as marked on the road but habitual lanes which drivers are used to use (e.g. cutting sharp curves or crossing drawn lane separators in order to smoothen driven trajectory).

Strictly speaking, this is not a correct solution because it may contain habitual lanes, which can contradict with official driving lanes marked on the road surface.

Luckily, aerial imagery would serve well as source of information since traffic lanes can be easily identified due to road surface marking. Current segmentation and classification methods are capable of identifying the different spectral values of the marking and the surrounding road surface. Geometry analysis of identified road surface marking polygons is necessary to consequently obtain a consistent and logical network of driving lanes. Yet, only little research has been done so far in obtaining such data from aerial images.

High resolution road data with sub-object level of detail have multiple usages in the real world. In navigation for example, the data can be used to identify the correct driving lane which the driver should choose and display it to the driver. Traffic control is another field where such datasets can be used: automatic reactions to traffic situations, traffic modelling or measuring traffic intensity are only several examples. However, the most significant usage is autonomous driving. As Fischer et al. (2018) present, high resolution road data combined with a GNSS unit can be successfully used for localisation of autonomous vehicles and can even be used as one of the steering methods.

# 2. Literature review

A lot of research has been done in the field of satellite or aerial image analysis in order to obtain road infrastructure; an overview can be found in Wang et al. (2016). However, most of the authors have studied obtaining roads as a whole and they do not go beyond road level of detail. On the other hand, extensive research has been done in detecting road surface marking in computer vision (i.e. from the perspective of a car or a driver) and working technical solutions are being implemented in modern cars in a form of advanced driver assistance systems (ADASs). Yet, these solutions have different approach and their technical solutions would not have much use when applied to aerial images due to different characteristics (perspective, time frame, coverage etc.). To sum up, there are only few papers which describe detection of single driving lanes or detection of surface marking in aerial images, what is the aim of this paper; these are presented in the section 2.3.

## 2.1. Detection of roads in satellite/aerial images

A lot of research has been done to detect roads or road networks in high resolution aerial images, therefore, only a limited amount of work is presented here. Hormese and Saravanan (2016) suggest a method of road detection which consists of three fully automatic steps: image segmentation, decision making based on continuity and vectorisation of segments identified as roads. Cheng et al. (2016) come with a method consisting of multiscale collaborative representation and graph cuts to create homogeneous segments, tensor voting and non-maximum suppression algorithm to extract a smooth centreline and a fitting based centreline connection algorithm to connect the road centreline around its intersections. Leninisha and Vani (2015) use a geometric active deformable model based on flowing water (starting from a manually set seed point), Yuan et al. (2009) use Locally Excitatory Globally Inhibitory Oscillator Networks (LEGION) to extract road networks, Shahi et al. (2015) propose a new spectral index (Road Extraction Index, REI), or Xia et al. (2018) use deep convolution network to detect roads in high resolution satellite images. Bakhtiari et al. (2017) suggest using support vector machine together with mathematical morphology method to extract roads in satellite images. Zhang et al. (2018) combine high resolution aerial images with LiDAR data to increase effectiveness of road centreline detection in complex urban scenes. A step forward is done by Máttyus et al. (2017) who estimate road network topology using deep learning and shortest path method directly from aerial imagery.

## 2.2. Detection of road surface marking by computer vision

Several different methods are used in technical solutions to detect road surface marking even in real-time while driving on a road. Mostly they combine optical camera with image analysing software and differ in approaches towards identifying the road marking. They use thresholding (Gontran et al. 2006), frequency analysis (Kreucher and Lakshmanan 1999), structure analysis (Lai and Yung 2000) or multilevel image classification (Jeong and Nedevschi 2005). López et al. (2010) identifies ridges of prospective marking polygons instead of edges and applies Random Sample Consensus (RANSAC) to fit them to the model.

Lee et al. (2017) use Convolutional Neural Network to detect different kinds of road marking even in bad weather or during the night.

## 2.3. Detection of driving lanes in aerial images

Jin, Feng and Li (2009) first extract rural roads form aerial images using maximum likelihood clustering algorithm and then driving lanes are detected within extracted roads using texture enhancement and morphological operations. The identified lines of driving lanes are finalised using a thinning algorithm, the short dangling branches are removed; and Douglas-Peucker simplification method and Bezier interpolation simplify and smoothen the lines. However, the test area was not big or complex enough to satisfactorily prove the method.

Jin and Feng (2010) propose combining low resolution satellite images to detect road centreline first and then apply Anisotropic Gauss filtering in a high resolution aerial image to extract road surface marking. Extracted marking is then combined with the road which is derived from the centreline to form a correct resulting image. Their method has detection rate of about 97.8% and quality of 94.6%; yet, the testing area of approximately 8 $km^2$ is too small for proving real-life effectiveness. Furthermore, authors themselves state that this method largely depends on straight road sections.

Fischer et al. (2018) feel the need of such data for autonomous driving (AD) usage, specifically for ego positioning (determining the position of the vehicle) and scene understanding. They test a method of surface marking detection on a motorway which combines vector data (in this case OpenStreet Map) with aerial image to create a buffer along road centrelines in which the detection is done. To identify the marking they use Random Forest classifier and Gabor Filtering. Their results are promising; yet their testing area was limited to motorway environment with expected higher number of errors in non-motorway environments and the method would require image pre-processing if used on a country scale.

A method of Baumgartner and Hinz (2003) extracts road networks from high-resolution aerial imagery in complex urban areas. They integrate detailed knowledge about the context of roads using explicitly formulated scale-dependent models of road networks. Proposed method has according to the authors 75% completeness and 95% correctness; however, the method is highly dependent on the model(s) and cannot handle some specific cases (such as traffic jams or complicated junctions) in the image.

In his diploma thesis, Seo (2012) presents a new method combining aerial images with images depicting road vectors on a background. The method uses two levels of image features: low-level and mid-level (smaller scale). On a low-level, pixels of aerial images are parsed and grouped according to their quantized orientations. The grouping is iteratively repeated until the size of the groups (segments) is big enough to fulfil a predefined condition. These segments are combined with vectors depicting images to identify ridge, extremity and bifurcation points of road lanes. On a mid-level, first potential road-depicting regions are identified; and then driving directions are estimated. Finally, features from both levels are combined to define a sub-region of interest in which lane-marking detection results from previous steps are used to generate road lanes. As author states, 79% of the resulting road-lanes were correctly extracted yet in some specific cases (overpasses, railway bridges or undetected road sections) the method shows shortcomings or failures.

Jin et al. (2012) propose a complex two-step method of extracting road network with sub-object resolution on lane level from aerial images in rural regions. First, they extract road network from high resolution aerial images using homogeneity histogram thresholding algorithm, such data are thinned and vectorised to make a vector road network. Secondly, they classify the extracted road surface with support vector machine to eliminate other ground details and then use Gabor filters to detect road surface marking. Authors identify obstruction of visible surface markings by cars or shadows as the main shortcoming of the method.

The literature presented in Section 2.1 could serve as a knowledge-base for the first step of this paper – the image classification. The most appropriate method presented could be taken into account when performing image classification on input aerial imagery; however, image classification has its limitations in software equipment that is available. Therefore only those methods could be used which were offered in the available software (see Chapter 6 for details).

Due to the fact that high resolution imagery is broadly available for the whole Czechia as well as there is available spatial data describing existing road infrastructure, both data sources can be used in the process. Therefore, it is not necessary to use multiple-scale imagery to first identify roads themselves in the imagery like Jin, Feng and Li (2009) or Jin and Feng (2010).

## 2.4. Contribution of the thesis

This paper builds upon the listed literature in the following way. Fischer et al. (2018) offer a detailed overview of road surface marking classification and present an elaborate method with promising results. To further process and analyse data extracted from the classification, Seo (2012) presents in his diploma thesis a method of combining vector data with classified aerial images. However, he uses raster representation with background for vector data and the analysis is therefore performed in raster data model. These two works are the closest to the method presented in this thesis and the ones on which this thesis builds upon.

The main contribution of the thesis lies in analysing the classified RSM in a vector data model and therefore the contribution of this thesis starts there, where most of the previously presented papers end. Together with vector polylines of road infrastructure, it will be possible to combine information from vectorised classification output (the classification was exported and further used in the process as a polygon vector dataset) to get a relevant result of lines representing RSM on roads.

Furthermore, vector representation allows using geometry topology rules and thus repairing topological errors which arise during the classification, during the analysis or because of neglected maintenance of RSM (fading RSM, road patches etc.); or simply because of shades in the images (trees, cars, buildings). Moreover, the thesis brings up an easy method of connecting appropriate RSM lines in places where the lines are blocked from view in the original aerial image (locations under a bridge or in a shadow of a building).

Third contribution of this work is distinguishing between different types of RSM (broken and solid lines); none of the previously mentioned papers describes that. This information is

appended to line segments as attributes to be available for further usage. Several shortcomings of the proposed method were identified and thus offer space for further improvement.

## 2.5. Objectives

This diploma thesis has two main objectives. The first one is to propose a method that automates creation of data representing road infrastructure at the level of detail of single driving lanes. Vector data representation of RSM polygons will be used during the process and both geometric and semantic (attributes) description will be provided as a result. Furthermore, multiple real-life situations including non-trivial situations such as highway junctions or underpasses will be solved in order to create a holistic method which will be able to process complex traffic infrastructure. To conclude, the crucial contribution lies in geometry analysis of polygons classified as RSM, not in improving current classification methods or coming up with a new classification method.

The second objective is to test the proposed method in a case study area and discuss its effectiveness and correctness as well as adjust input data and (numerical) parameters in order to obtain as precise results as possible. It is expected that multiple datasets will be entering the analysis as input data and that several conditions will be stated which have to be fulfilled to obtain relevant (and correct) results.

# 3. Requirements and regulations of road surface marking

Road surface marking (RSM) is part of the road infrastructure and is necessary for a common usage of roads just like traffic lights of traffic signs. There is a common standard for RSM worldwide; yet there are differences in colours, breaking interval, width or in specific phenomena marked on the road surface. Every state should have legally binding regulations concerning RSM so that the marking has the same characteristics everywhere on public roads within the state's national borders. Thanks to the regulations it is possible to create a set of machine-readable rules for RSM so that an algorithm is able to identify different types of RSM and detect what they stand for. This is used in the proposed method as supplementary information stored in attributes of RSM lines and is used as additional source of information in the process of creating the resulting network of driving lanes.

## 3.1. Czech legislative framework

In Czech legislative framework, requirements and regulation of RSM are defined and described in documents called Technické podmínky (TP, the term does not have an official English translation). TP are issued by the Ministry of Traffic of the Czech Republic as binding requirements and regulations of planning, construction and management of traffic infrastructure in Czechia based on modern findings and trends in the field. Some of the documents are publically accessible (http://www.pjpk.cz/technicke-podminky-tp/).

Technické podmínky Nr. 133, Zásady pro vodorovné dopravní značení na pozemních komunikacích ("Principles of road surface marking on roadways", translated by the author) define details of use, placement and visual characteristics (meaning, size, colour, texture etc.) of RSM. Technical solution for RSM is defined in Czech Technical Standard Nr. 1436 (ČSN EN 1436), yet it is not important for the analysis.

There are four acceptable colours in TP 133; each for different kinds of marking – white, yellow, red and blue. Blue is allowed to be used only in case of a sign V 10g Omezené stání (Restricted parking) and red for marking of escape lanes and of areas for pedestrians or cyclists (bicycle lanes, highlighting of dangerous crossings etc.). Yellow (in detail further) is in general reserved for two different phenomena: parking/standing restrictions and temporary marking. White marking (in detail further) is the most common one which is used for general lane and area marking including arrows, text and other information on the road. The width of the line is always 0.125 m, unless it is not stated differently (TP 133).

### 3.1.1. Yellow RSM

Yellow colour is used in five situations, four of them describe different parking/standing situations (see below) and the fourth is temporary marking in areas of road reconstruction or temporary traffic marking changes; in that case the yellow marking is superior to the white one if they contradict.

- No-parking areas, i.e. areas in which the driver is allowed to stop the vehicle for a short period of time without leaving it (V 12a, V 12d).

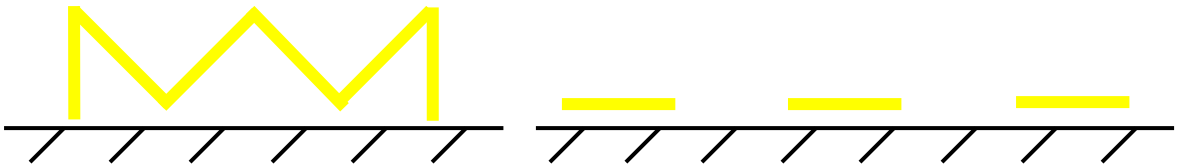

Figure 1: Surface road marking signs V 12a (left) and V 12d (right)

- No-standing areas, i.e. areas in which the driver is not allowed to stop the vehicle unless the traffic situation or safety requires it (V 12c).

Figure 2: Surface road marking sign V 12c

- Box junctions, i.e. areas in which it is not allowed under any circumstances to stop the vehicle and thus block the way for other vehicles (the middle of a junction, emergency vehicles exit etc.; V 12b).

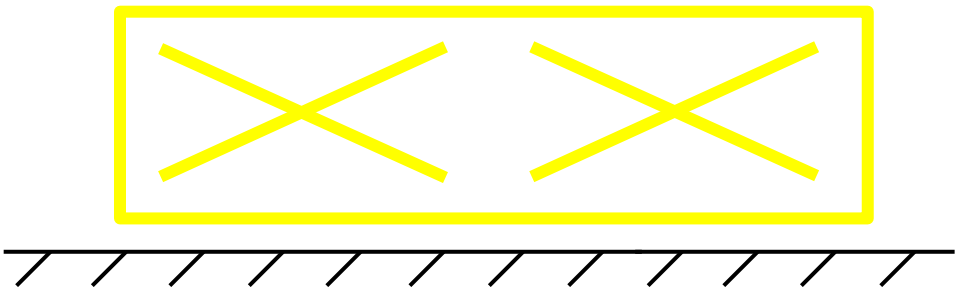

Figure 3: Box junction V 12b

- Areas of bus or tram stops (V 11a, V 11b) can be according to TP 133 marked either in yellow or in white.

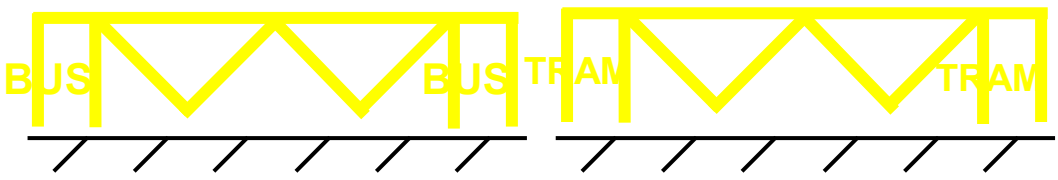


Figure 4: Surface road marking V 11a (left) and V 11b (right)

### 3.1.2. White RSM

White marking is used for any other information (besides those mentioned earlier) which is to be given to the driver on the road. It is used not only for lane and area marking (which is further described below as it carries the most valuable information for road lane detection) but also for pictograms, directional arrows, horizontal traffic signs, speed limits, text and any other information which is relevant for safe driving.

- An edge line (V 4) marks the edge of the road and thus separates a driving lane from a shoulder. An edge line is a solid white line with a width of 0.25 m (in specific cases also 0.125 m is allowed, TP 133). Theoretically, an edge line should be marked on both sides of all paved roads except roads in municipalities with a vertical curb. Yet, it is quite common not to have an edge line marked on minor country roads in Czechia.

  An edge line can be broken when there is acceleration or deceleration lane (V 2b) or a reserved lane (V 2b) or a parking lane (V 10d).

- A solid white line (V 1a) is used to separate driving lanes where it is not allowed to change lanes. It is usually used for separating driving lanes in opposite directions (when the road is wider than 6 m) or for separating driving lanes in the same direction before/after a junction or a manoeuvre. It is not to be crossed beside special circumstances. The width of the line is supposed to be 0.125 m (TP 133).

- A double solid line (V 1b) is used to emphasise the fact that it is not allowed to change lines. It is used in dangerous areas such as sharp curves, frequent traffic accident areas or to separate opposite directions with multiple driving lanes in one direction. The width of the lines and the space between the lines is 0.125 m (TP 133).

- A broken white line (V 2a, V 2b) separates driving lanes where it is allowed to change lanes. It is not marked on those roads whose width is less than 6 m. The width of the line is 0.125 m and 0.25 m (in specific cases) and the breaking interval changes under different circumstances (see Table 1).

| Line [m] | Break [m] | Width [m] | Use between |
|---|---|---|---|
| 6 | 12 | 0.125 | motorway lanes of the same direction |
| 3 | 6 | 0.125 | road lanes of the same direction (> 1 lane) |
| 3 | 6 | 0.125 | road lanes of the opposite direction (road width > 7 m) |
| 3 | 3 | 0.125 | a road lane and a lane for slow vehicles |
| 3 | 1.5 | 0.125 | road lanes of the opposite direction (road width 6–7 m) |
| 3 | 1.5 | 0.125 | road lanes before a junction/solid line |
| 3 | 1.5 | 0.25/0.125 | normal and reserved lanes (bus lane, taxi lane etc.) |
| 3 | 1.5 | 0.125 | road lanes and tram corridor in road level |
| 1.5 | 1.5 | 0.125 | road lanes within junction |
| 1.5 | 1.5 | 0.25 | acceleration/deceleration lane and main lane |
| 0.5 | 0.5 | 0.25 | normal and parking lane/(bus) stop lane |

Table 1: Different line and break intervals of broken white lines (TP 133)

- It is allowed to use a combination of a solid and a broken line when it is relevant from which side the line can be crossed. The solid line is in the axis of the road adjacent to neighbouring lines and the broken line is drawn to the side from which the crossing is allowed.

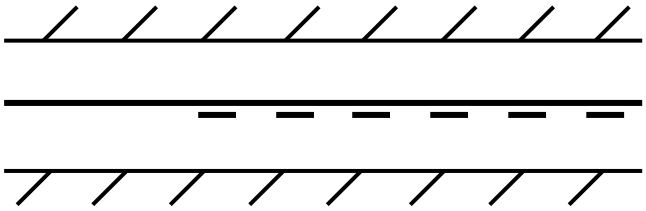

Figure 5: Combination of a solid and a broken line

- Diagonal hatching marks area which is prohibited to drive over because of fluent streamlining of traffic. Such areas serve as in-level separators within junctions; they highlight beginning and ending of acceleration/deceleration lanes, physical separators or barriers or highlight beginning and ending of a new parallel lane. The hatching lines are usually 0.5 m wide with offset of 0.5 m, 1 m or 1.5 m and angle to the edge of the adjacent driving lane of 45° in the direction of travel of the adjacent lane.

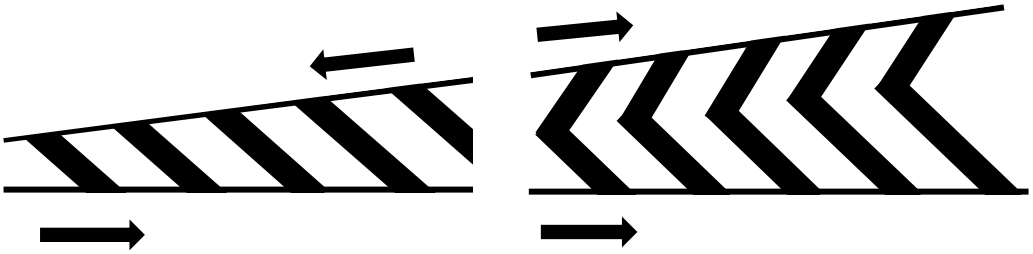

Figure 6: Diagonal hatching marks, arrows show direction of travel

- Transition from a broken white line to a solid white line can be signalised beforehand by either shortening the gap between lines of the broken line to 1.5 meter (see Table 1) or by adding warning arrows on the line. The size and shape has been changed in 2013; the old version still exists on Czech roads and that is the reason why both versions are being presented here.

Figure 7: Warning arrows signalising transition from a broken to a solid line, old (left) and new (right) versions

- In areas where special attention is necessary (such as areas where vehicles slow down suddenly and before dangerous pedestrian crossings) a white zigzag line can be drawn to increase drivers' attention. The lane is 0.125 m wide and the zigzag is drawn on the right side of the lane 0.5 m or 1 m into the lane.

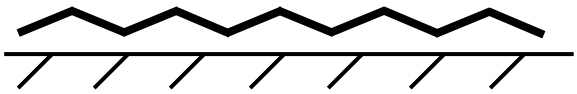

Figure 8: White zigzag line at the edge of a lane

# 4. Used concepts and methods

The process of obtaining driving lanes' centrelines from aerial images is complex and consists of multiple steps which use both aerial images and geospatial data (vector lines of current road infrastructure) as input. Following theoretical concepts and methods were used during the process described in this thesis; they are listed with a brief explanation in this chapter. Next chapter (5.) describes their sequence in practise, reasons for choosing them and what is their contribution on the way to final results.

## 4.1. Image classification

In order to identify road surface marking in aerial images image classification is used. From vast amount of classification methods an object-based image analysis (OBIA) was chosen. Using objects for image classification instead of pixels has become widely popular in scientific research in the new millennium due to growing resolution of available imagery and changing needs of the analysis (Blaschke 2009). Yet, one of the earliest use of OBIA date back to 1970s to the work of R. L. Kettig and D. A. Landgrebe (Kettig and Landgrebe 1976), who combined Rodd's conjunctive partitioning algorithm with a minimum distance sample classifier. Since then multiple methods of segmentation and segment classification were successfully developed for usage for image classification (Blaschke 2009).

### 4.1.1. Image Segmentation

The initial step of image classification is image segmentation. For image segmentation, bottom-up or top-down approaches can be used. The bottom-up approaches start with single pixels (or pixel groups) and group them consecutively into objects until the requested homogeneity criteria of objects are met. Top-down approach, on the other hand, starts with the whole image as initial object and divides it in smaller groups until the requested homogeneity criteria of objects are met. Various criteria can be used as homogeneity criteria, among others intensity, colour, texture or shape.

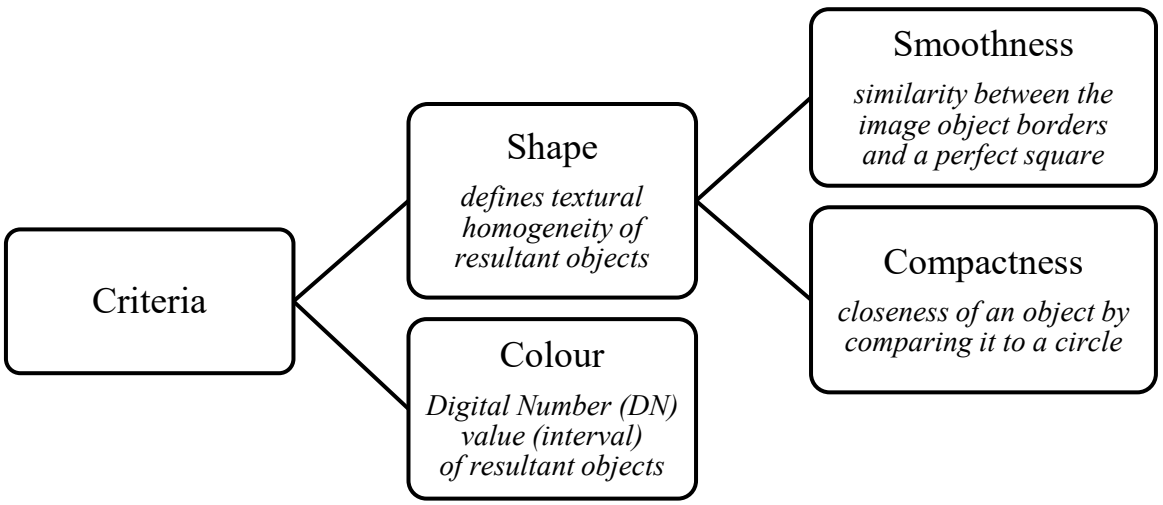


Figure 9: Schema of segmentation criteria

An iterative segmentation (a multi-resolution segmentation) can be used in order to divide the image iteratively into continuously smaller segments (i.e. segments are further segmented into smaller segments). With this approach it is guaranteed that the borders between segments which are created on higher levels (in previous phases) stay the same on this and all subsequent levels.

In this work, multi-resolution image segmentation based on colour and shape (smoothness and compactness) of objects was used, see details in Section 5.2.1

### 4.1.2. Classification of segments

The segments of an image are to be classified to classes according to criteria which are set by the user. There are multiple characteristics of segments which can be decisive criteria (band values, geometrical, positional or textural characteristics, hierarchy, thematic attributes etc.) and even segment related or linked and scene or region related features can (among others) be used as decisive criteria. However, relevant are those which determine the most the desired output (RSM in our case) from the background.

Due to the complexity of input images multiple criteria are expected to be needed to efficiently separate objects containing RSM. RSM is characteristic with its colour (compared to surrounding road surface) which leads to brightness being one of the most relevant criteria. Yet, other objects appear in the images such as vehicles, truck trailers, buildings or tiled pavements which have similar brightness values and thus other criteria have to be added to distinguish efficiently between RSM and these objects.

Geometric characteristics seem to be most useful when it comes to this problem because geometric extent of RSM is easily quantifiable and different enough in comparison to the previously mentioned (undesirable) objects to successfully distinguish between them and RSM. From the set of available geometric characters, width and length/width ratio seem to be the most relevant because RSM consists of narrow lines with specified width and length (yet with different length for different types of marking). Furthermore, the width of RSM lines can be easily recalculated to pixels when the pixel size of the image is known.

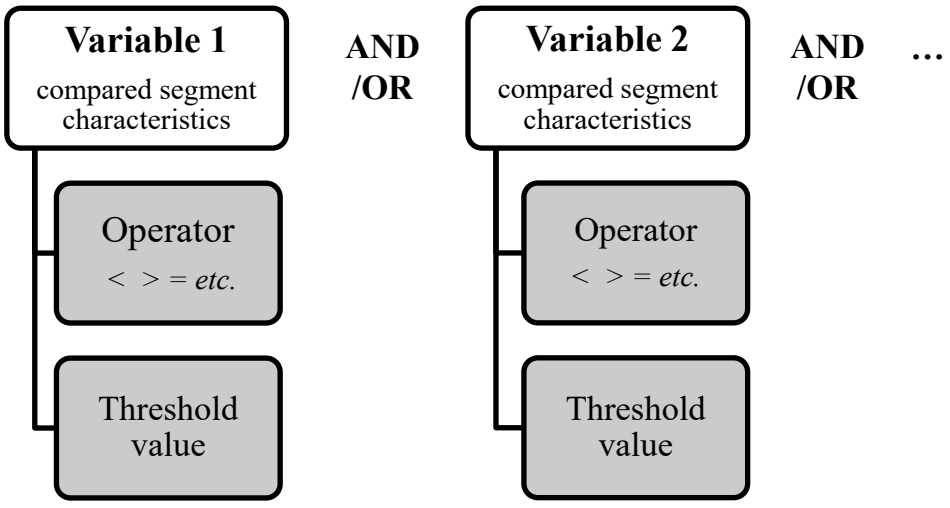


Figure 10: Schema of classification logic

The list of used criteria and further details regarding image classification are presented in Chapter 5.

## 4.2. Methods performed on raster

Several times during the process the data were converted to a raster to perform analysis or editing which is better to be performed in raster data model. The fact that the input (aerial image) of the whole process is a raster image means that, as long as the original cell size is preserved, there is no relevant resolution loss caused by conversion.

### 4.2.1. Thinning

Thinning is applied in order to regularise the shape of RSM and thus to minimise drawbacks of the segmentation and complex shape of classified RSM polygons. Mostly these drawbacks are adjacent objects which share a segment with RSM (because the part of RSM represents the majority of the segment yet the smaller object does not have a separate segment).

Thinning is performed on a binary raster (raster with two values: RSM and background). The method shrinks object boundaries closer together until the desired reduction (simplification) is performed; the desired reduction width is in this case equal to original image (raster) resolution to avoid presence of concave polygons.

Figure 11: Thinning of a raster: original raster (left) and thinned raster (right)

## 4.3. Methods performed on vectors

Vector representation of data is used mostly during the process with all three geometry types – points, lines and polygons. Vectors serve as a better representation of real life objects in geospatial databases because their costs (storage size, analysis difficulty, resolution etc.) are lower (better) than those of raster representation. The output driving lanes, for the same reasons, use vector representation as well.

### 4.3.1. Skeletonization

Skeletonization is used to convert polygons to lines. There are two geometry structures which are mostly used as skeletons of polygons – straight skeleton and medial axis. Straight skeleton "is defined as the union of the pieces of angular bisectors traced out by polygon vertices during the shrinking process." (Aichholzer et al. 1995, page 753). Compared to that, medial axis is defined as a set of points which have more than one closest point on the polygon's boundary (Blum 1967) resulting in a possibility of curved segments of medial axis in concave polygons. Straight skeleton, on the other hand, is only composed of straight line segments. Yet, straight skeleton and medial axis are identical for convex polygons (Aichholzer et al. 1995) and all skeletonised polygons are convex when being skeletonised in this work, therefore there is no difference between a straight skeleton and a medial axis of a polygon in this work.

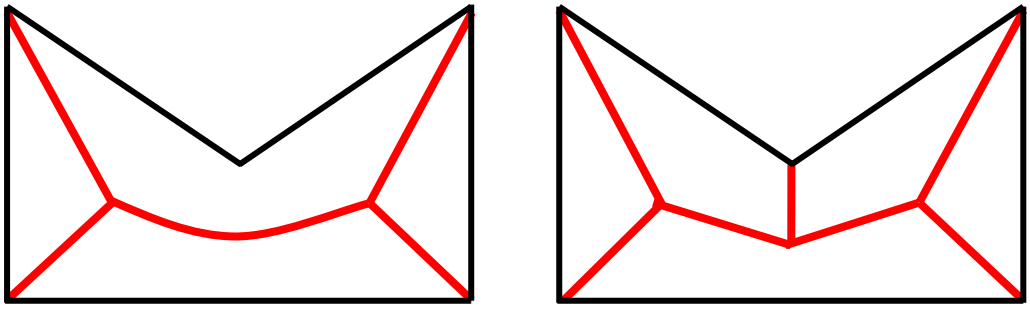

Figure 12: Comparison of medial axis (left) and straight skeleton (right) of a concave polygon

### 4.3.2. Buffer

Buffer in Geoinformatics is a synonym for a neighbourhood zone of an object. It encompasses the input object in all directions in a specified distance (buffer radius or buffer distance). Buffer of a point is a circle (with the original point in the centre), buffer of a line is a polygon (with the original line as skeleton without dangles) and buffer of a polygon is a polygon. Point buffer is constructed as a simple circle around the point, line and polygon buffers are constructed with line (border) expansion.

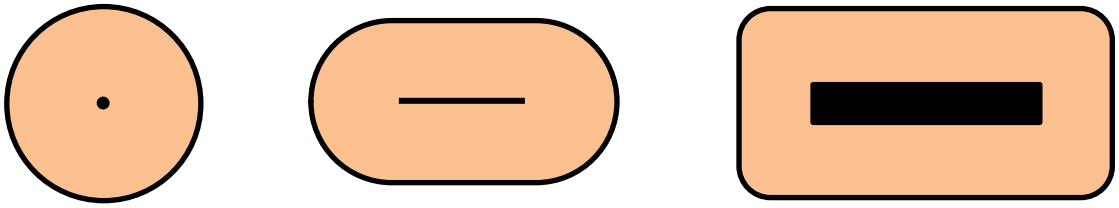

Figure 13: Buffer of a point (left), line (middle) and polygon (right)

### 4.3.3. Douglas–Peucker line simplification

As one of the most used simplification algorithms, the Douglas–Peucker algorithm is widely implemented in geographic information system (GIS) software. It is a recursive algorithm which starts with the simplest representation of a line (a connection between start and end

point) and consecutively keeps adding points into the line until no points are left to be added which fulfil a pre-set condition (simplification parameter); its full description can be found in (Douglas and Peucker 1973). Figure 14 illustrates run of the algorithm.

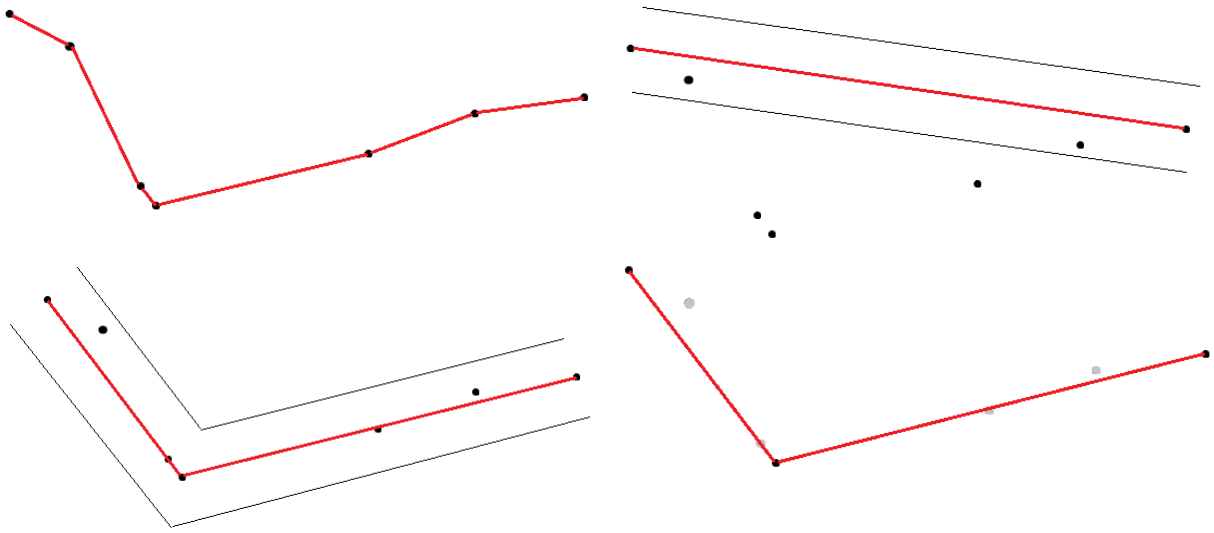

Figure 14: Sequences of Douglas–Peucker algorithm, simplification parameter is illustrated with thin grey lines: original line (top left), first recursion (top right), second recursion (bottom left) and resulting line (bottom right)

# 5. Proposed method description

In this section the new proposed method of detection of driving lanes geometry from aerial images and existing spatial data is described. The method combines the information contained in aerial imagery (brightness value of pixels) with information from an existing vector dataset containing polylines of roads. As well as these spatial data, relevant information describing parameters of and rules for road surface marking are another necessary input without which the method could not provide correct results.

The following section is organised as follows: first, the input data are described including conditions which have to be fulfilled in order to run the algorithm and obtain a correct result in the end. Secondly, the aerial image analysis (segmentation and classification) including its parameters is presented. Thirdly, the proposed algorithms of RSM geometry detection including all their steps and parts are described and finally, an algorithm classifying RSM lanes and assigning attributes is presented.

## 5.1. Data description

As mentioned before there are two different sources of input information: aerial imagery and vector layers. It would be possible to use only aerial imagery as input data and run a two-step image classification as Jin, Feng and Li (2009), Jin and Feng (2010) or Jin et al. (2012) suggest to first identify roads in the imagery, extract and vectorise them and then use identified segments of roads only for RSM detection. Yet, this method is highly dependent on the quality of input imagery and its classification and therefore it seems more convenient to rather use existing (and topologically correct) data such as Fischer et al. (2018) suggests in his work. Such data are available with almost worldwide coverage and despite bad quality in some regions they have more than sufficient quality and coverage in developed regions of the world.

### 5.1.1. Aerial imagery

When working with aerial imagery the most relevant parameters are spatial resolution (also called ground resolution), coverage, and spectral information. If these parameters do not coincide with the goal of the analysis, the result of the analysis will not be correct/will not bring correct and relevant information. Therefore, it is important to pay attention to all these parameters at the same time.

Spatial resolution of an aerial image is the amount of detail captured in the image. It is characterised by a number which describes the area on the ground (length of a rectangle edge) which is equal to one pixel of the image. For the purpose of extracting RSM lines in the images, the pixel size must be smaller than the width of the narrowest line of RSM, ideally at least two times smaller for an accurate analysis. Considering Czech national standards and regulations presented in Chapter 3 of this paper the minimal width of a RSM line is 0.125 m and therefore an acceptable spatial resolution of input imagery must be at least 0.125 m (12.5 cm). With modern state-of-the-art aerial scanners it is possible to have images with spatial resolution of up to 2.5 cm (Holmes 2012) and thus it is possible to meet the requirement of minimal spatial resolution as well as to have several times better resolution

to get a very accurate classification result. This is the main reason why satellite imagery cannot be used as input image source because despite the fact that its acquisition is faster and cheaper, its spatial resolution does not reach desired minima yet.

Figure 15: Aerial image with appropriate spatial resolution (10 cm); 1:1 view (left) and in detail (right)

Coverage of an aerial image means the size of the area which is covered by chosen aerial images. It is important to choose the source of aerial images according to desired result so that both the input images and the desired result have the same coverage. Many countries have such imagery which is provided by their national mapping agencies or private companies. It is possible though to combine different image sources but this will lead to bigger resource consumption during the image classification. More preparation steps foregoing image classification would have to be taken such as image alignment and adjustment of pixel values (so that pixels with same objects have same DN values) or the classification would have to be done for each images source separately and only the results of classification joined afterwards.

To be able to perform image segmentation and image classification, spectral information contained in the image must be relevant. There are multiple spectral bands available in visible, thermal or infrared spectra. For the purpose of identifying RSM (which is marked in white or yellow on a grey or a black surface), the visible spectrum (Red, Green and Blue spectral bands) is sufficient for the analysis.

| **Spectral band** | Red | Green | Blue |
|---|---|---|---|
| **Wavelength [nm]** | 650-700 | 500-550 | 450-500 |

Table 2: Red, Green and Blue bands and intervals of their wavelengths

### 5.1.2. Vector files

The vector files representing up to date road infrastructure are besides aerial images the other important information source. Polylines depicting centrelines of roads carry the information about the position of roads as well as can carry other relevant information as attributes such as number of driving lanes, information about the direction of traffic, width or category of the road etc. Parameters which have to be taken into account when choosing an appropriate data source are scale, completeness and correctness.

Scale, completeness and correctness can be described in this section together as 'geometry parameters' as they relate to geometry rather than attributes. Scale or minimum mapping unit represents (similarly as in aerial images) the spatial resolution of a dataset. Connected with vector spatial data, it is the size of the smallest object (length of the shortest polyline) from the specific dataset. Practical meaning of this parameter portrays positional accuracy of the dataset expressed in coordinates compared to the position of the object in the real life. It is desirable that the scale of input vector data is similar to the one of input aerial images; using data with too small scale would lead in errors in the result (missing parts of identified RSM), data with too big scale can always be generalized to appropriate scale. When it comes down to completeness and correctness, it is necessary to make sure that the chosen data are topologically correct and comprise all roads on which RSM is being detected.

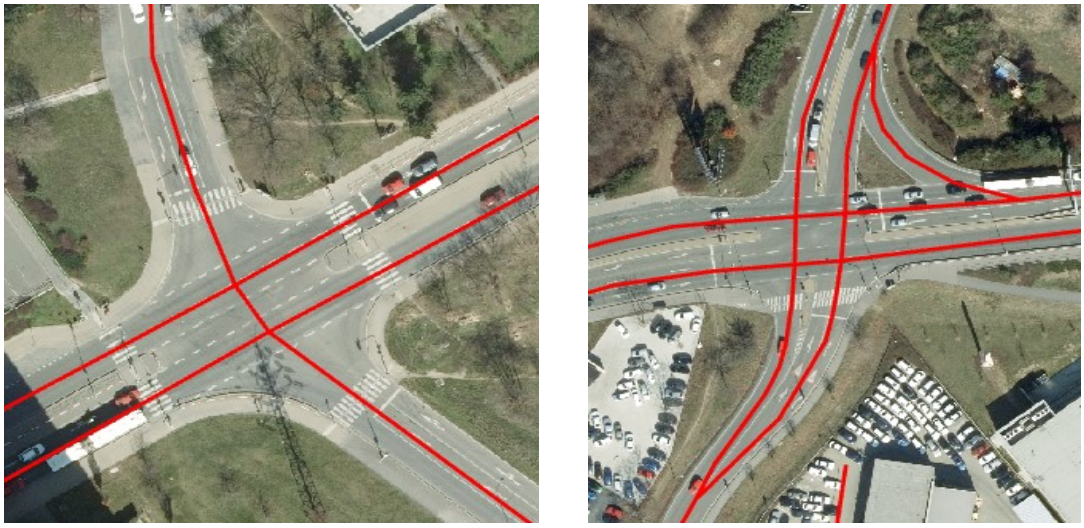

Figure 16: Vector line with adequate geometry parameters

The method was designed so that no attributes are necessary to be connected to the input vector file. However, having several attributes such as number of driving lanes, information about the direction of traffic, width or category of the road would make the analysis faster and would lower necessary manual inputs; this leaves an open space for future improvements of the method.

## 5.2. Image analysis

The aim of image analysis is to identify polygons representing RSM in an aerial image. As stated in Chapter 4, an object-based image analysis (OBIA) was used. Both image segmentation and classification of segments were performed in existing software specialised

on analysis of aerial or satellite imagery. It was necessary though to examine the imagery and to find out fitting criteria which will lead to desired result (polygons representing RSM).

### 5.2.1. Image segmentation

To successfully perform the process of image segmentation, it was necessary to find acceptable values/weights of the four criteria which are used during segmentation (scale, colour, smoothness and compactness). Since the segmentation is a bottom-up process, the scale parameter defines size of resulting segments, i.e. how much (into how big segments) will the pixels be grouped. A multi-phase segmentation is used in order to be dividing the image iteratively into continuously smaller segments (yet, as stated in Section 4.1.1. the boarders between segments on higher levels stay the same).

The segmentation was run on input images multiple times with different settings and the results were compared in order to identify the setting with the best segmentation result. The best segmentation result is the one which creates segments from RSM polygons which do not include surrounding road surface in the same segments, i.e. isolates the best pixels with RSM from the rest of pixels. Two iterations (with scale factors 100 and 20) were found out to contain enough detail for classification of RSM; all parameters of the segmentation can be found in Table 3.

| First iteration | | Second iteration | |
|---|---|---|---|
| Number of cycles | 1 | Number of cycles | 1 |
| Image Layer Weights | R: 1, G: 1, B: 1 | Image Layer Weights | R: 1, G: 1, B: 1 |
| Scale Parameter | 20 | Scale Parameter | 100 |
| Shape | 0.1 | Shape | 0.1 |
| Compactness | 0.5 | Compactness | 0.5 |

Table 3: Parameters of image segmentation

### 5.2.2. Classification of segments

The second step of image classification is the classification of segments into two final classes: road surface marking (RSM) and the rest (background). Due to a high complexity of input images it was found out as almost impossible to create just one class for RSM and therefore multiple subclasses (all of them representing the class RSM) with varying characteristics were created. From a vast amount of available decisive criteria, combinations of three of them (one combination per subclass) were chosen to contain the most relevant information (mean brightness, width and length/width ratio).

For each of the chosen combination variables, threshold values and operators need to be set in order to complete the classification successfully. The most effective way of finding the combinations is to manually choose several sample segments and examine their respective values. After the variables, threshold values and operators are found, they are set as input parameters of the classification process and all segments in the image are classified based on the threshold values and operators into either the RSM class or the background class.

Criteria and threshold values as well as operators used for the classification are presented in Table 4.

| Variable | Operator | Threshold value | Operator | Variable | Operator | Threshold value |
|---|---|---|---|---|---|---|
| brightness | > | 175 | AND | length/width | > | 3 |
| brightness | > | 145 | AND | length/width | > | 4 |
| brightness | > | 150 | AND | width | < | 7 |

Table 4: Classification criteria used in the classification

Figure 17: Segmentation result (on the left) and classification result (on the right)

## 5.3. RSM geometry detection

Polygons representing RSM (output of image classification) serve as the main information carrier for detection of geometry of driving lanes as they separate individual driving lanes. In specific cases it is not required to draw RSM on roads, in such cases the method would not withstand.

Premise for successful geometry detection is to inspect and prepare data from the classification and to resolve errors of misclassification. The amount of errors depends o input data as well as on classification settings and therefore will not be further discussed in this chapter; discussion of specific input data and specific classification settings which were used as case study for the purpose of this paper can be found in Chapter 6.

Proposed algorithms are described in the following way. First, the accompanying text states why and with which effect the respective step is used and how does it influence the input data. Each step is documented with accompanying code in Python 3. The important steps are illustrated graphically as well.

### 5.3.1. Creating RSM lines

#### Step 1: Merging neighbouring polygons

Due to the fact that the classification output is split into vast amount of polygons, it is necessary as a first step to organise them better for vector analysis. Therefore, neighbouring polygons are merged first into one aggregated polygon with the same outer boundaries as the small partial polygons had. This does not change location of the input data; it only reduces their amount and simplifies further work.

```
>>> # merge neighbouring input polygons into one polygon if they touch (share
part of borders)
>>> arcpy.Dissolve_management(input, dissolved, "", "", "SINGLE_PART",
"DISSOLVE_LINES")
```

Figure 18: Merging neighbouring polygons: before (left) and after (right)

#### Step 2: Selecting suitable polygons

Because of the fact that skeletonization is used in the following steps, it is necessary to select only those features which have appropriate shape for the operation. Appropriate shape means that the features have evident main direction; i.e. there is a significant difference between their height and length. Objects with shape not fulfilling this criterion will have their skeleton created in various directions which would create a zigzagging line in the result with a significant positional accuracy error. As the most appropriate condition to limit such small polygons, a condition of border length (attribute Shape_Length) being higher than 4 meters was set. Another algorithm (Section 5.4.) was created to analyse polygons which are removed from the process by this selection.

```
>>> # select only those polygons which are big enough to represent polygons
of road surface marking (the value (min. border length 4 m) is based on the
data examination)
>>> arcpy.Select_analysis(dissolved, selected, "\"Shape_Length\" > 4")
```

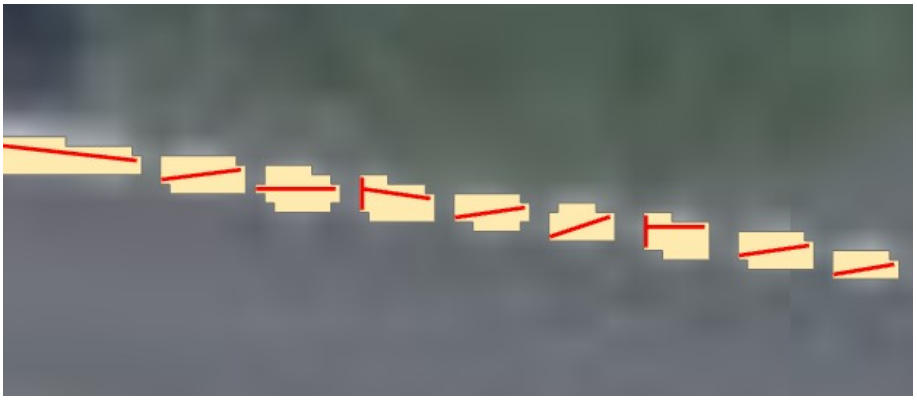

Figure 19: Inappropriately shaped polygons and created lines (zoomed in)

### Step 3: Thinning

Width of the polygons varies and this would negatively influence resulting shape of the skeleton. Therefore, all polygons representing RSM are first converted to raster and consecutively thinned up to the resulting width of one cell. With this step, the RSM representing shapes are simplified as their outer border is smoothed.

```
>>> # conversion to raster with same resolution as original raster, therefore
no information loss or generalisation
>>> arcpy.PolygonToRaster_conversion(selected, "OBJECTID", raster,
"CELL_CENTER", "NONE", "0,1")
>>> # thin rasterised lines of road surface marking to the width of 1 pixel
>>> thinned = arcpy.sa.Thin(raster, "NODATA", "NO_FILTER", "SHARP", "1")
```

Figure 20: RSM representation after thinning

### Step 4: Skeletonization

As the next step skeletonization is used to convert RSM to lines. RSM is currently represented with a raster and an in-built tool or Arcpy module which uses skeletonization is used in order to convert the input raster to line. The output line does not contain all parts of skeletons

of input polygons (to which is the raster internally converted) but filters them according to the overall shape and continuity of the raster (Figure 21).

```
>>> # raster values conversion (necessary for the following tools)
>>> integer = arcpy.sa.Int(thinned)
>>> # conversion of raster representing road surface marking to polylines
representing road surface marking
>>> arcpy.RasterToPolyline_conversion(integer, polyline, "ZERO", "0",
"SIMPLIFY", "VALUE")
```

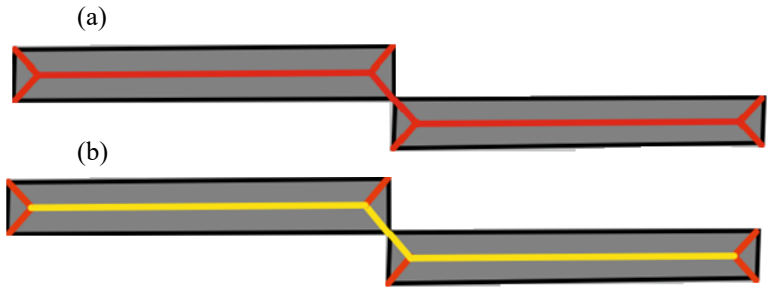


Figure 21: Thinned polygon (solid black outline and dark grey fill) and its skeleton (red line) in (a). Selected skeleton lines (the result of step 4) are shown in yellow in (b)

### Step 5: Simplifying generated line

Automatically generated polyline does not have a regular simplified shape in places where the input classified polygons representing RSM had irregular shape (dangling bays which disrupt regular rectangular shape of the polygon) therefore it is simplified with an inbuilt tool (Douglas–Peucker approach). The simplification tolerance used was set to 0.3 meters because this is the maximum distance which together with the width of RSM lines does not cause undesired positional error resulting from oversimplification.

```
>>> # simplify generated lines
>>> arcpy.SimplifyLine_cartography(polyline, outsm, "POINT_REMOVE", "0,3",
"RESOLVE_ERRORS", "NO_KEEP", "CHECK", "")
```

Figure 22: Lines before simplification (left) and simplified lines (right)

### 5.3.2. Interconnecting RSM lines

The second proposed algorithm forms the next step of the analysis. In this step, the lines which represent detected RSM are connected and transformed into a continuous network of polylines to represent borders of driving lanes. Such polylines no longer copy the exact location of RSM lines but form a linked network of boundaries of driving lanes instead.

#### Step 1: Calculating orientation of RSM line segments

In the first step, orientation of each line segment, which represents RSM (result of the algorithm described in 5.3.1.), is calculated. The input dataset is splitted into single line segments first and orientation is consecutively counted for each segment with an inbuilt tool. The orientation is saved as a new attribute called DirMean. Orientation is an important attribute for further use in Steps 3 and 4.

```
>>> # split lines representing road surface marking in vertices so that
following tools can be performed
>>> arcpy.SplitLine_management(input, splitted)
>>> # copy ID into a field accessible for Directional Mean tool
>>> arcpy.CalculateField_management(splitted, "arcid", "!OBJECTID!", "", "")
>>> # count orientation of line segments
>>> arcpy.DirectionalMean_stats(splitted, lindir, "ORIENTATION_ONLY",
"arcid")
```

#### Step 2: Calculating orientation of road line segments

The second step is the same as Step 1 with one significant difference – the input data. Instead of lines representing RSM, Step 2 splits and calculates orientation of polylines representing roads. Vector road representation is the second (and the only vector one) input dataset besides aerial imagery. No attributes are required for this step, see Section 5.1.2. for details.

```
>>> # split roads at vertices so that following tools can be performed
>>> arcpy.SplitLine_management(insil, splittedrds)
>>> # copy ID into a field accessible for Directional Mean tool
>>> arcpy.CalculateField_management(splittedrds, "arcid", "!OBJECTID!", "",
"")
>>> # count orientation of roads polyline segments
>>> arcpy.DirectionalMean_stats(splittedrds, lindirrds, "ORIENTATION_ONLY",
"arcid")
```

#### Step 3: Calculating orientation difference

The difference in orientations (calculated in the previous steps) is used to delete redundant line segments of RSM which are not relevant for the geometry of driving lanes such as stop lines, directional arrows, diagonal hatching lines and others. It is calculated as an absolute value of differences of orientations counted in Steps 1 and 2 and saved as an attribute (overwriting previous values in existing column CompassA).

```
>>> # join road segments and road surface marking segments based on proximity
(both with calculated orientation to be able to count their angle)
>>> arcpy.SpatialJoin_analysis(lindir, lindirrds, joined, "JOIN_ONE_TO_MANY",
"KEEP_ALL", "", "CLOSEST", "10", "")
```

```
>>> # calculate angles between road surface marking segments and road
segments to find out which segments don't have similar direction as roads (to
be deleted later)
>>> arcpy.CalculateField_management(joined, "CompassA", "abs(!DirMean! -
!DirMean_1!)", "PYTHON3", "", "")
```

| OBJECTID | Shape | Shape_Length | CompassA | DirMean |
|---|---|---|---|---|
| 1 | Polyline | 2,343075 | 129,805571 | 320,194429 |
| 2 | Polyline | 0,360555 | 33,690067 | 56,309933 |
| 3 | Polyline | 0,848528 | 135 | 315 |
| 4 | Polyline | 1,615549 | 111,80141 | 338,19859 |
| 5 | Polyline | 1,17047 | 109,983106 | 340,016894 |
| 6 | Polyline | 1,655295 | 115,016894 | 334,983106 |
| 7 | Polyline | 1,746425 | 113,629378 | 336,370622 |
| 8 | Polyline | 1,581139 | 108,434949 | 341,565051 |
| 9 | Polyline | 1,615549 | 111,80141 | 338,19859 |
| 10 | Polyline | 1,081665 | 123,690068 | 326,309932 |

Figure 23: Attributes of first 10 line segments with their orientation counted in the column DirMean

### Step 4: Deleting RSM segments with different orientation

Based on the difference of orientation counted in previous step RSM line segments with high difference in orientation are removed from the dataset. As a threshold value an angle of 35° is used. This may seem too high as a threshold but due to two following reasons it was proven necessary to use such a high number.

Figure 24: Examples of RSM line segments with significant (yet smaller than 35°) difference in orientation (road polylines in red); area of a junction (left) and changing organisation of driving lanes (right)

The first reason is inaccuracy of the input roads dataset in junction areas where the adjoining segments have almost orthogonal angle yet the RSM segments form a continuous curve. The second reason is the requirement of not deleting those segments of RSM which are not parallel to road axes where organisation of driving lanes changes (see Figure 24 for illustrations of example situations). It came up during data examination that some line

segments have reverse direction leading to two threshold values (35° and 360°-35 °) on both sides of 360° spectra.

```
>>> # initialise the field relevant for the cursor
>>> field = ["CompassA", "Shape_Length"]
>>> # use cursor to delete specific rows (segments with different (<35°) angle
between road surface marking and joined road segments)
>>> with arcpy.da.UpdateCursor(joined, field) as cursor:
>>>     for row in cursor:
>>>         # difference between orientation of RSM segments and road segments
            is > 35° (counted for both directions, therefore < 325°)
>>>         if (row[0] > 35 and row[0] < 145) or (row[0] > 215 and row[0]
            < 325):
>>>             cursor.deleteRow()
>>>         # to delete short redundant lines
>>>         elif row[1] < 0.5:
             cursor.deleteRow()
```

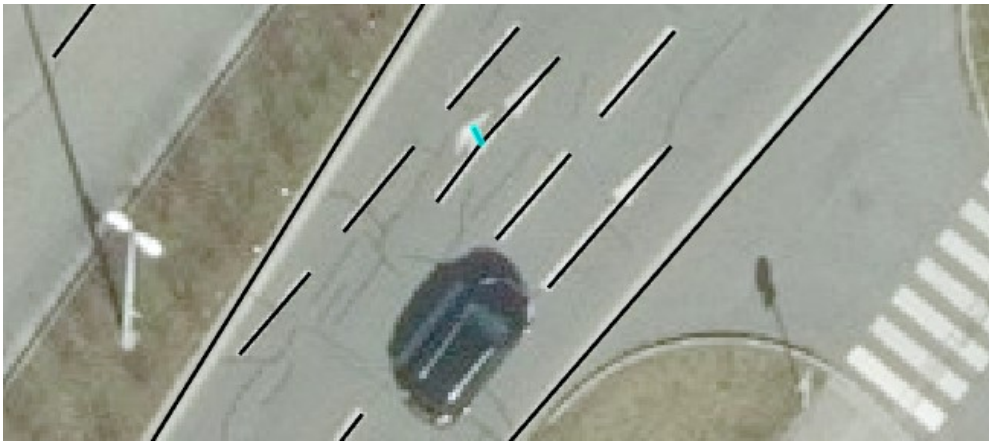

Figure 25: A segment to be deleted (light blue) because of difference in orientation

### Step 5: Connecting single lines of RSM into driving lanes boundaries

The crucial step of this algorithm is connecting self-standing lines of RSM into continuous lines representing boundaries of driving lanes; in other words, bridging gaps between broken lines or gaps in solid lines (results of imperfect image classification). This step is solved with an inbuilt buffering tool which is applied twice in two different directions with two different buffering distances. Intersecting buffers are merged into one polygon afterwards and this polygon is skeletonised in order to obtain a line.

```
>>> # dissolve segments to be able to identify gaps between road surface
marking lines
>>> arcpy.Dissolve_management(joined, dissolved, "", "", "SINGLE_PART",
"UNSPLIT_LINES")
>>> # create transects (short parallel lines) at the end of dissolved
segments
>>> arcpy.GenerateTransectsAlongLines_management(dissolved, transects,
"1000", "0,5", "END_POINTS")
>>> # delete transects which are in the same location
>>> arcpy.DeleteIdentical_management(transects, "Shape", "1", "")
>>> # create buffer around transects to connect road marking segments with
gap < 7 m in between
>>> arcpy.Buffer_analysis(transects, buffer, "3,5", "FULL", "FLAT", "ALL",
"", "PLANAR")
```

```
>>> # create narrow buffer along road marking segments
>>> arcpy.Buffer_analysis(joined, buffer2, "0,2", "FULL", "FLAT", "ALL", "",
"PLANAR")
>>> # union of buffers (created above) into one polygon which will be
skeletonised
>>> arcpy.Union_analysis([buffer, buffer2], union, "", "", "NO_GAPS")
>>> # dissolve boundaries of neighbouring buffers
>>> arcpy.Dissolve_management(union, dissolved2, "", "", "SINGLE_PART", "")
>>> # densify line to prevent errors in the next step
>>> arcpy.Densify_edit(dissolved2, "DISTANCE", "1", "", "")
>>> # create centre line of buffers (convert polygons to (centre-) lines)
>>> arcpy.PolygonToCenterline_topographic(dissolved2, centreline, joined)
```

Figure 26: Illustration of Step 5: input RSM (thick grey lines), merged buffers (light yellow polygons), generated centreline (red)

### Step 6: Deleting redundant line segments

There are lines representing not only driving lanes borders in the dataset now but also lines representing other objects marked on the road surface such as directional arrows, pedestrian crossings etc. which are to be removed in this step. The criterion which decides whether lines should/should not be deleted is their length – the lines which are not part of driving lanes borders are short and can be easily identified. Dangles are removed in this step too.

```
>>> # removing dangles (shorter than 5 m) from the skeleton
>>> arcpy.TrimLine_edit(centreline, "5", "DELETE_SHORT")
>>> # dissolve the skeleton so that short redundant lines (e.g. arrows on the
road surface) can be deleted
>>> arcpy.Dissolve_management(centreline, lines, "", "", "SINGLE_PART", "")
>>> # using cursor to delete short redundant lines (e.g. arrows drawn on the
road surface)
>>> with arcpy.da.UpdateCursor(lines, "Shape_Length") as cursor:
>>>     for row in cursor:
>>>         # deletes redundant lines in the middle of lanes which are not RSM
but directional arrows for example; these have length < 13 meters
>>>         if row[0] < 13:
>>>             cursor.deleteRow()
```

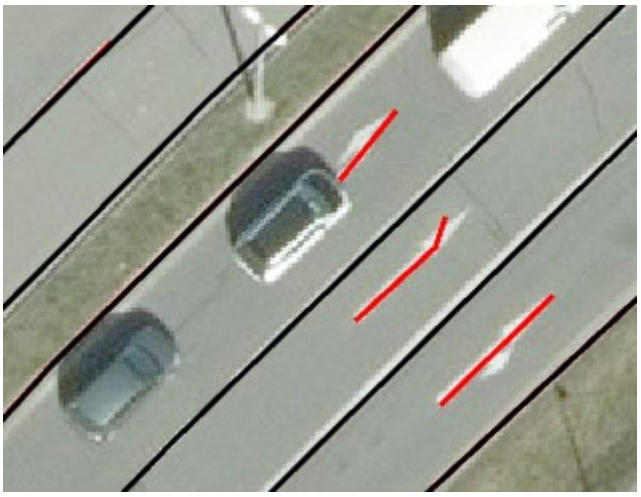

Figure 27: Directional arrows (red) to be deleted based on length criterion

Two different types of gap errors occur in lines which were created by the currently described algorithm. Ones of them are short gaps with length of several meters (condition in this work is shorter than 3 meters) which were created because the buffers of neighbouring RSM line segments (Step 5) were not intersecting. This can be caused by a slight rotation of the line segments or by other objects (for example shadows or cars) covering the original RSM line in the aerial image. The other types of gaps are long gaps with multiple meters or even completely unclassified RSM lines. Such gaps are caused either by a bad visibility of the actual RSM lines on roads (neglected road cleaning or fading marking) or by being covered or shadowed by other objects (long trucks or busses, building shadows, bridges etc.). Bridging such long gaps is more complicated than bridging short gaps; therefore there are two different approaches for bridging.

First, the short gaps are bridged as follows (Steps 7, 8 and 9 of the currently described algorithm). Then, the extensive process of bridging long gaps is performed which consists of three components (5.3.3.).

**Step 7: Identifying gap vertices**

Dead ends of lines (result of Step 6) are converted to points and the nearest points (from the same point layer) are identified as attributes for the created point layer. There is a distance condition of 3 meters – that means that points which do not have a neighbour within 3 meters do not have a joined neighbour. Such points are not vertices of a short gap and will be skipped in the next step.

```
>>> # convert dead ends of RSM lines into points
>>> arcpy.FeatureVerticesToPoints_management(lines, points, "DANGLE")
>>> # connect start points with their nearest located endpoints within 3
meters (maximum bridging distance)
>>> # NOTE: Lines around central separators which are narrower than this
distance will be connected as well if they have closely located dead ends.
>>> arcpy.Near_analysis(points, points, "3", "LOCATION", "NO_ANGLE",
"PLANAR")
```

Figure 28: A short gap caused by a shadow of a directional sign stand

### Step 8: Connecting gap vertices

Vertices created in Step 7 are now interconnected with their nearest neighbour and a line is created to interconnect them. Such line has only two vertices (start and end point) which are vertices of the same (short) gap. Created line is added to the output dataset.

```
>>> # define attributes used in cursor
>>> fields = ['SHAPE@X', 'SHAPE@Y', 'NEAR_FID', 'NEAR_X', 'NEAR_Y']
>>> # initialise index i
>>> i = 1
>>> # loop through all gaps
>>> with arcpy.da.SearchCursor(points, fields) as cursor:
>>>     for row in cursor:
>>>         # if the start vertex does have a closely located end vertex
>>>         if row[2] > 0:
>>>             # initialise empty list of points
>>>             pointGeometryList = []
>>>             # initialise empty point object
>>>             point = arcpy.Point()
>>>             # create point object with the coordinates of gap start point
>>>             point.X = row[0]
>>>             point.Y = row[1]
>>>             pointGeometry = arcpy.PointGeometry(point)
>>>             # add the point to the list
>>>             pointGeometryList.append(pointGeometry)
>>>             # create point object with the coordinates of gap end point
>>>             point2 = arcpy.Point()
>>>             point2.X = row[3]
>>>             point2.Y = row[4]
>>>             pointGeometry = arcpy.PointGeometry(point2)
>>>             # add the point to the list
>>>             pointGeometryList.append(pointGeometry)
>>>             # name the line properly according to its index
>>>             name = "aa" + str(i)
>>>             # create line with a proper name of the points in the list
                (it is the line which bridges gap between start and endpoint)
>>>             arcpy.PointsToLine_management(pointGeometryList, name, "",
                 "", "NO_CLOSE")
>>>             # append the line to existing dataset of original lines
>>>             arcpy.Append_management(name, lines, "NO_TEST", "", "", "")
```

```
>>>             arcpy.Delete_management(name)
>>>             # increase index
>>>             i = i + 1
>>>         # if there's no closely located end vertex for a start vertex,
            just increase index and go on
>>>         else:
>>>             i = i + 1
```

**Step 9: Simplifying resulting lines**

The last step consists of two operations – dissolving created lines so that the gaps are not separate line segments and simplifying created lines with a Douglas–Peucker algorithm.

```
>>> # dissolve the output line
>>> arcpy.Dissolve_management(lines, out, "", "", "SINGLE_PART", "")
>>> # simplify resulting lines
>>> arcpy.SimplifyLine_cartography(out, output, "POINT_REMOVE", "0.3",
    "RESOLVE_ERRORS", "NO_KEEP", "CHECK", "")
```

Figure 29: A short gap before (on the left in red) and after (on the right in black) bridging

## 5.3.3. Bridging long gaps

Long gaps are gaps which are longer that 3 meters and thus were not bridged in 5.3.2. Their bridging is more complex because the newly created line has to reflect the shape of the road. In other words, if the gap is in a curve, then the line has to be curved as well. To observe this condition, input data layer is needed which will include information about the shape of roads.

The vector dataset representing road infrastructure (described in 5.1.2.) can fully serve the purpose and can be used for bridging long gaps. It will, however, need special attention because the resulting points need to be ordered according to their appearance along the line.

### Identifying long gaps

Similarly as when identifying short gaps in Step 7 of 5.3.2, long gaps are identified using dead ends of RSM lines which are converted to points. However, due to the complexity of RSM

on roads and due to the complexity of road junctions, manual input is necessary in here to decide which points are vertices of which gaps. Furthermore, if no line was created at all (the whole line was in shadows and thus not classified), then the points have to be created manually as well.

An algorithm was created to create dead end vertices and add appropriate attribute fields to the newly created vertices layer. This algorithm first converts dead ends of RSM lines into points, then creates a selection of those points which are within the case study area (to remove the points which mark ends of lines at the borders of the case study area) and creates an attribute called GapID as the last step. GapID is then filled in manually during the editing process. The result of the editing process is shown in Figure 30.

```
>>> # convert dead ends of RSM lines into points
>>> arcpy.FeatureVerticesToPoints_management(inlines, linepoints, "DANGLE")
>>> # select only those dead ends which are within test area (not even on
the border)
>>> select = arcpy.SelectLayerByLocation_management(linepoints,
    "COMPLETELY_WITHIN", clip, "", "NEW_SELECTION", "NOT_INVERT")
>>> # export selected features as feature class
>>> arcpy.CopyFeatures_management(select, output,"")
>>> # add attribute field to the output, values have to be added manually
after running the script
>>> arcpy.AddField_management(output, "GapID", "INTEGER", "", "", "", "",
    "NULLABLE", "REQUIRED", "")
```

## Editing road data

Input polyline road dataset is converted into points which are then ordered with the following script. The resulting point dataset should be checked manually because eventual missing points or incorrect ordering would cause the following tool (Bridging long gaps in the following section) to fail or to produce incorrect results.

**Step 1: Converting road polylines to points**

First, the input polyline dataset representing roads is cropped according to the borders of the case study area. Secondly, the cropped road segments which touch are dissolved into single objects and are cut in the location of RSM dead ends (vertices). Thirdly, vertices of the line segments are converted into a point layer and attributes of dissolved lines (FID) are joined to them.

```
>>> # clip roads with the boarder of the test area
>>> arcpy.Clip_analysis(inrds, clip, clipped)
>>> # dissolve them into single elements if they touch
>>> arcpy.Dissolve_management(clipped, dissolved, "", "", "SINGLE_PART", "")
>>> # add road vertices where gap vertices are
>>> arcpy.SplitLineAtPoint_management(dissolved, vertices, splitted, "5")
>>> # convert road lines to points
>>> arcpy.FeatureVerticesToPoints_management(splitted, points,"ALL")
>>> #join road attribtues to points
>>> arcpy.SpatialJoin_analysis(points, dissolved, joined, "JOIN_ONE_TO_MANY",
 "KEEP_ALL","", "INTERSECT", "", "")
```

### Step 2: Counting order of points along lines

The second step is the crucial one which determined order of points in which they appear along road lines. This is decided by a distance of the point to the start point of the respective line. To count such distance, two loops are nested into each other, the outer one over (dissolved) road lines and the inner one over road vertices. The first point of each line (start point) is copied to a list which is appended to resulting points at a later stage (Step 3).

The inner loop contains a condition which decides whether the currently accessed point does or does not belong to the currently accessed line; in a positive case, the point is used to cut the dissolved polyline to be able to get the length of its segment between this point and start point.

```
>>> # create feature class for data created in loops
>>> out = arcpy.CreateFeatureclass_management("C:\\...\\Default.gdb", "out",
"POLYLINE", "", "DISABLED", "DISABLED", "5514", "")
>>> # initialise variables for loops
>>> line = 0
>>> pointList = []
>>> # outer loop over road lines
>>> with arcpy.da.SearchCursor(dissolved, ["OBJECTID", "SHAPE@"]) as cursor:
>>>     for row in cursor:
>>>             # assign variables
>>>             line = row[0]
>>>             geom = row[1]
>>>             # add start points of the line into list
>>>             pt1 = arcpy.Point()
>>>             pt1.X = geom.firstPoint.X
>>>             pt1.Y = geom.firstPoint.Y
>>>             pt1Geometry = arcpy.PointGeometry(pt1)
>>>             pointList.append(pt1Geometry)
>>>             # inner loop over road points
>>>             with arcpy.da.SearchCursor(joined, ["JOIN_FID", "Shape@X",
             "Shape@Y"]) as cursor:
>>>                 for row in cursor:
>>>                     # if road point is on road line
>>>                     if row [0] == line:
>>>                     # create point object and assign coordinates
>>>                     point = arcpy.Point()
>>>                     point.X = row[1]
>>>                     point.Y = row[2]
>>>                     pointGeometry = arcpy.PointGeometry(point)
>>>                     # split original dissolved line with created point
>>>                     arcpy.SplitLineAtPoint_management(geom, pointGeometry,
                        clipline, "1")
>>>                     # select the line between start point and split point
>>>                     select = arcpy.SelectLayerByAttribute_management
                        (clipline, "NEW_SELECTION", """ OBJECTID = 1 """, "")
>>>                     # append selected line into created dataset
>>>                     arcpy.Append_management(select, out, "NO_TEST", "",
                        "", "")
>>>                     # delete temporal splitted line
>>>                     arcpy.Delete_management(clipline)
```

### Step 3: Creating points with appropriate attributes

A new attribute is created and counted for the lines which were created in the previous step for storing their length. The lines are then converted to points (only their endpoints are converted) and line starting points (isolated in a list in the outer loop of the previous step) are appended to them.

```
>>> # add field to dataset
>>> arcpy.AddField_management(out, "LineLength", "DOUBLE", "", "", "",
"NULLABLE", "REQUIRED", "")
>>> # count line length (distance of points along lines) in the added field
>>> arcpy.CalculateField_management(out, "LineLength", "!Shape_Length!",
"PYTHON3", "", "")
>>> # convert line end points to points
>>> arcpy.FeatureVerticesToPoints_management(out, points2, "END")
>>> # copy points from the list (first points of road lines) into resulting
points
>>> for point in pointList:
>>>     arcpy.Append_management(point, points2, "NO_TEST", "", "", "")
```

### Step 4: Sorting

In the last step possible duplicate points are deleted first. Then, road line FID is joined to all points and finally, the points are sorted according to two attributes: road line FID first (ascending) and their distance (along line) from the line start point (ascending as well).

```
>>> # delete identical points (with the same position)
>>> arcpy.DeleteIdentical_management(points2, "SHAPE", "0.1", "")
>>> # join road line ID attribute to resulting points
>>> arcpy.SpatialJoin_analysis(points2, dissolved, joined2,
"JOIN_ONE_TO_MANY", "KEEP_ALL", "", "INTERSECT", "", "")
>>> # sort resulting points according to joined road polyline ID and distance
along the line
>>> arcpy.Sort_management(joined2, output, [["JOIN_FID", "ASCENDING"],
["LineLength", "ASCENDING"]], "")
```

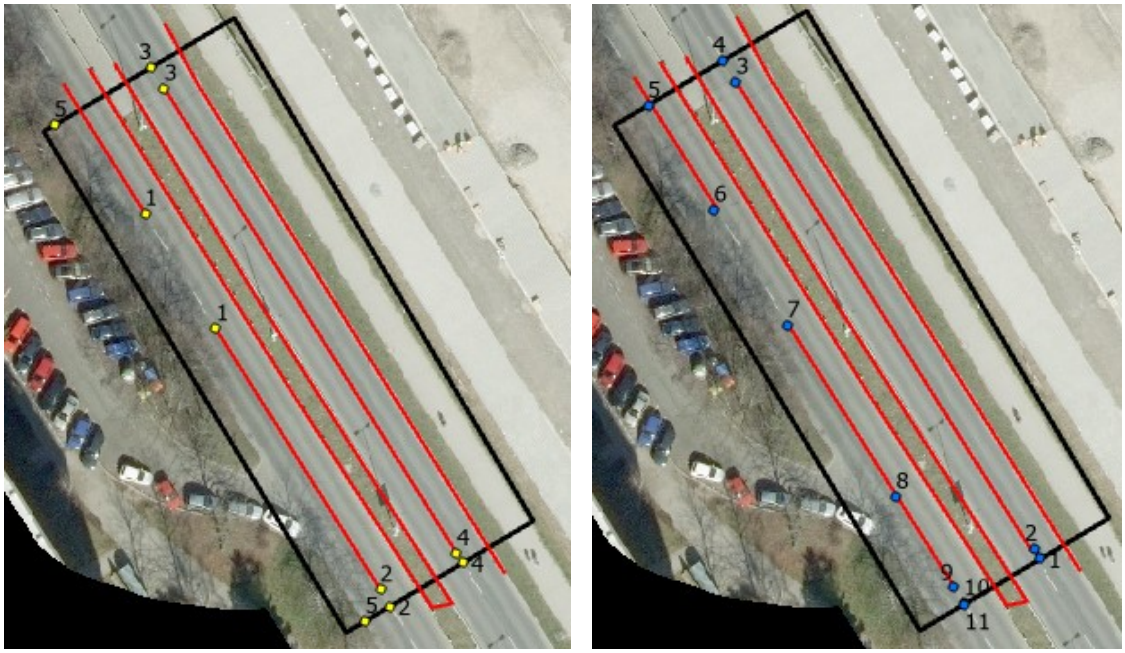


Figure 30: Gap vertices (on the left in yellow) with GapID and road points (on the right in blue) with FieldNr after correct manual editing.

## Bridging long gaps

Both gap vertex dataset and road point dataset are the only input data for the main bridging algorithm. The algorithm consists of several steps as follows and creates lines which cover identified gaps.

### Step 1: Counting coordinates of vertices

Firstly, input vertices are copied so that the input data remain unchanged. After that, X and Y coordinates of vertices are counted and added as attributes to the layer.

```
>>> # copy input vertices so that the input file remains unchanged
>>> arcpy.CopyFeatures_management(vertices, vertices2,"")
>>> # add attributes (X and Y coordinates of a vertex)
>>> arcpy.AddField_management(vertices2, "VertX", "DOUBLE", "", "", "", "",
    "NULLABLE", "REQUIRED", "")
>>> arcpy.AddField_management(vertices2, "VertY", "DOUBLE", "", "", "", "",
    "NULLABLE", "REQUIRED", "")
>>> # calculate attributes (X and Y coordinates of a vertex)
>>> arcpy.CalculateGeometryAttributes_management(vertices2, [["VertX",
   "POINT_X"], ["VertY", "POINT_Y"]], "METERS", "", "5514", "")
```

### Step 2: Counting coordinates of road points

Secondly, the input road points are sorted according to their logical order on the road (attribute FieldNr) and their X and Y coordinates are counted and added as attributes as well.

```
>>> # sort rows in the table according to their logical order along RSM lines
>>> arcpy.Sort_management(roadpoints, sorted, "FieldNr", "")
>>> # calculate X and Y coordinates of the points
>>> arcpy.AddGeometryAttributes_management(sorted, "POINT_X_Y_Z_M", "METERS",
"", "5514")
```

### Step 3: Identifying neighbourhood

After calculating X and Y coordinates in Steps 1 and 2 both point datasets are ready to be interconnected according to their spatial proximity. For each gap vertex the nearest road point is identified and then the relevant information of vertices (ID, X and Y coordinates) are joined to road points according to the identified neighbouring relationship (which is now stored as NEAR_FID attribute of the vertex dataset).

```
>>> # identify the nearest neighbour of each RSM dead end (gap vertex)
>>> arcpy.Near_analysis(vertices2, sorted, "", "", "", "PLANAR")
>>> # join the calculated information (nearest neighbour) to the points which
are the closest to the vertices
>>> joined = arcpy.AddJoin_management(sorted, "OBJECTID", vertices2,
"NEAR_FID", "KEEP_ALL")
>>> # copy the joined table for usage in next steps
>>> arcpy.CopyFeatures_management(joined, joined2, "", "", "", "")
```

### Step 4: Initialising variables

This step consist of several simple operations and serves as preparation for the main part of the algorithm which consist of two nested loops (Step 5). First, the input road points are sorted according to their logical order along roads (attribute FieldNr) and a stop condition

value (imax) is calculated; this number describes number of gaps in the dataset. As the last preparation step an empty output feature class is created.

```
>>> # sort rows in table according to their logical order along RSM lines
>>> arcpy.Sort_management(joined2, sorted2, "_sorted_FieldNr", "")
>>> # calculate number of gaps in the test area (imax)
>>> imax = int(int(arcpy.GetCount_management(vertices2).getOutput(0))/2)
>>> #initialise index i
>>> i = 1
>>> # create output feature class
>>> out = arcpy.CreateFeatureclass_management("C:\\Users… ", "out",
"POLYLINE", "", "DISABLED", "DISABLED", "5514", "")
```

### Step 5: Bridging long gaps

The main part of the algorithm consists of two nested loops: an outer while loop which iterates over all gaps with index i and an inner for loop which iterates over all road points with a search cursor. Necessary variables are initialised before the first iterations of the loops. The output of this step is a polyline dataset with lines filling all gaps in the current case study area. A decision tree of Step 5 is illustrated in Figure 31 for a clearer explanation.

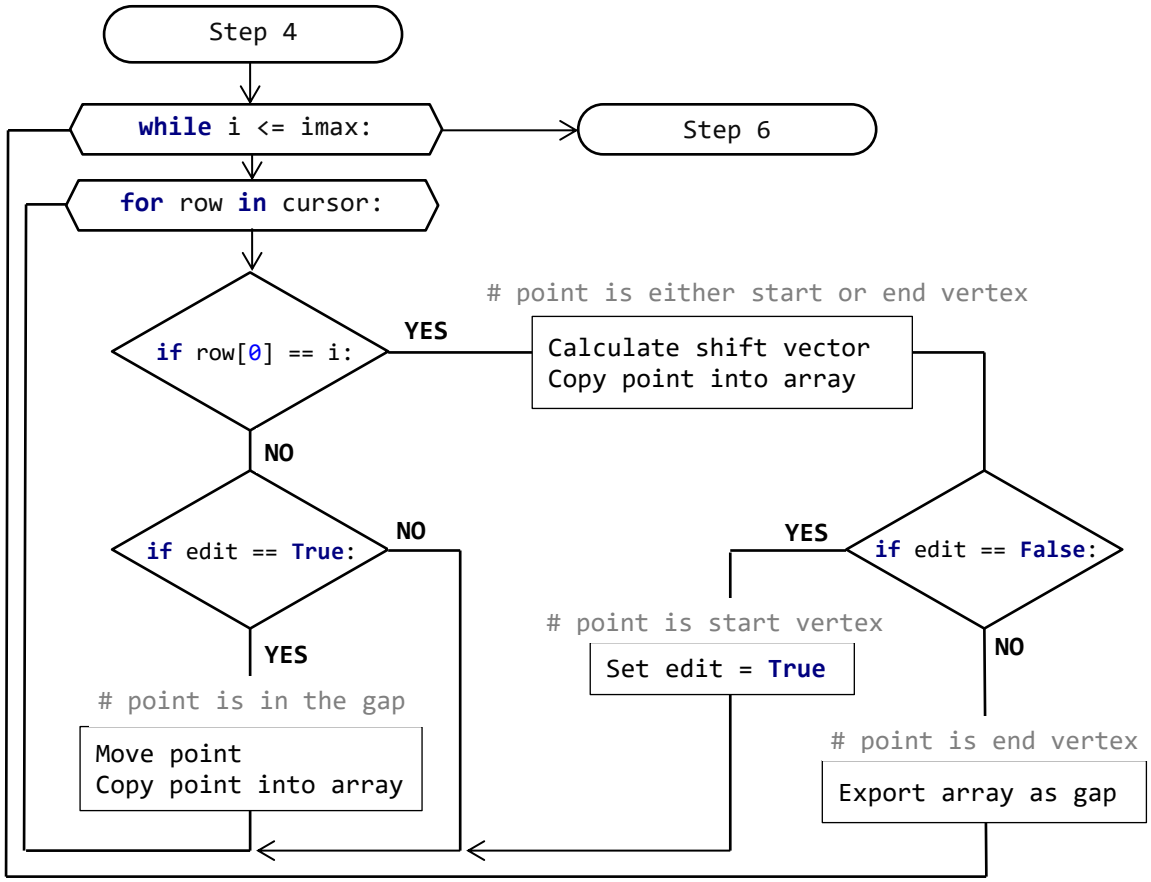


Figure 31: Decision tree of Step 5

```
>>> # define attributes used in cursor
>>> fields = ['_vertices2_GapID', '_sorted_POINT_X', '_sorted_POINT_Y',
    '_vertices2_VertX', '_vertices2_VertY']
>>> # loop over all gaps
```

```
>>> while i <= imax:
>>>     # initialise variables
>>>     edit = False
>>>     pointGeometryList = []
>>>     vector = [0, 0]
>>>     # loop over sorted road points
>>>     with arcpy.da.SearchCursor(sorted2, fields) as cursor:
>>>         for row in cursor:
>>>             # the point is a vertex (start od end point) of the gap that
                 is currently being edited
>>>             if row[0] == i:
>>>                 # calculate the shift vector
>>>                 vector = [row[3] - row[1], row[4] - row[2]]
>>>                 # initialise empty point object and assign coordinates
>>>                 point = arcpy.Point()
>>>                 point.X = row[3]
>>>                 point.Y = row[4]
>>>                 # create point object with coordinates of gap start point
>>>                 pointGeometry = arcpy.PointGeometry(point)
>>>                 # add the point to the list
>>>                 pointGeometryList.append(pointGeometry)
>>>                 # if the point is start point (first vertex) of currently
                     edited gap, change editing parameter to True (to allow
                     copying of following points in gap)
>>>                 if edit == False:
>>>                     edit = True
>>>                 # if the point is end point (a second vertex) of currently
                     edited gap
>>>                 else:
>>>                     # name line properly according to its index
>>>                     name = "gap" + str(i)
>>>                     # create line of the points in the list
>>>                     arcpy.PointsToLine_management(pointGeometryList, name,
                        "","", "NO_CLOSE")
>>>                     # append line to previously created dataset
                        of resulting gap lines
>>>                     arcpy.Append_management(name, out,"NO_TEST","", "", "")
>>>                     # delete temporal feature class
>>>                     arcpy.Delete_management(name)
>>>                     # increase index
>>>                     i = i + 1
>>>                     # no need to continue scrolling through rest of road
                        points after the gap is finished
>>>                     break
>>>             else:
>>>                 # editing parameter is True, therefore gap editing is on
>>>                 (the  point is between start and end point of the gap)
>>>                 if edit == True:
>>>                     # initialise empty point object and assign coordinates
>>>                     point = arcpy.Point()
>>>                     point.X = row[1] + vector[0]
>>>                     point.Y = row[2] + vector[1]
>>>                     # create point object with coordinates of the point
>>>                     pointGeometry = arcpy.PointGeometry(point)
>>>                     # add the point to the list
>>>                     pointGeometryList.append(pointGeometry)
>>>                 # no action as the point is not in the currently edited gap
>>>                 else:
>>>                     pass
```

**Step 6: Simplifying resulting line and adding attributes**

In the final step the resulting line is simplified with Douglas–Peucker algorithm and an attribute (LineType) is added to the output dataset. This attribute was set to "unknown" for all gap lines because it cannot be automatically decided whether the original unclassified RSM line was broken or full. The attribute needs to be filled in manually for a correct result.

```
>>> # simplify resulting lines
>>> arcpy.SimplifyLine_cartography(out, output, "POINT_REMOVE", "0.3",
"RESOLVE_ERRORS", "NO_KEEP", "CHECK", "")
>>> # add attribute field to the output
>>> arcpy.AddField_management(output, "TypeLine", "TEXT", "", "", "20", "",
"NULLABLE", "REQUIRED", "")
>>> # fill the attribute field with line type "unknown"
>>> with arcpy.da.UpdateCursor(output, "TypeLine") as cursor:
>>>     for row in cursor:
>>>         row[0] = "unknown"
>>>         cursor.updateRow(row)
```

Figure 32: Result of the algorithm: RSM lines with gaps (black) and bridging lines (red)

## 5.4. Small RSM polygons

'Small' polygons are those polygons which were removed from the process of RSM geometry detection in the second step (Selecting suitable polygons) of the algorithm presented in 5.3.1. The reason is as follows: their shape is too square and their length/width ratio is close to 1. Therefore the algorithm (especially in Steps 3 (thinning) and 4 (skeletonization) does not provide correct results – resulting skeletons are rotated from their supposed direction and this leads to undesired zigzags (see Figure 19). Therefore, a different approach was used to obtain such RSM lines; the approach is described in this section.

**Step 1: Merging neighbouring polygons**

First step is the same as in the previously described algorithm of RSM geometry detection. Neighbouring polygons are merged together in order to reduce the number of further steps

and to avoid duplicates (see Step 1 in 5.3.1 for details).

```
>>> # dissolve neighbouring input polygons into one polygon if they touch
(share part of borders)
>>> arcpy.Dissolve_management(input, dissolved, "", "", "SINGLE_PART",
"DISSOLVE_LINES")
```

**Step 2: Selecting 'small' polygons**

Step 2 is the exact opposite of Step 2 of the previously described algorithm. Here, only 'small' polygons (i.e. those which were not selected 5.3.1.) are selected for further analysis.

```
>>> # select only those polygons which are too small (and square shaped) and
their skeleton does not create a straight line which causes the tool to fail
(max. border length 4 m)
>>> arcpy.Select_analysis(dissolved, small, "\"Shape_Length\" <= 4")
```

**Step 3: Converting 'small' polygons to points**

Selected polygons are converted from polygons to an easier representation. Because conversion into a line caused problems on the previous algorithm, conversion to a point (centroid) is used.

```
>>> # convert dissolved polygons representing single lines of road surface
marking to centroids (points)
>>> arcpy.FeatureToPoint_management(small, points, "CENTROID")
```

**Step 4: Deleting isolated points**

Because of the fact that there may not only be polygons representing RSM in the classification dataset but other redundant (small) polygons as well, a deleting mechanism is implemented. It uses a distance to the nearest neighbour as a decision parameter whether to delete or not to delete a respective point. 'Small' polygons of RSM are always closely located to each other when representing a broken line (and according to Czech regulations (TP 133) it is not acceptable to mark a broken line with only one line segment); therefore all points which are isolated (have no nearest neighbour within 1.5 meters) are deleted.

```
>>> # find nearest neighbour of centroids within 1,5 m radius (to identify distant
points with no neighbours which are not part of road surface marking)
>>> arcpy.Near_analysis(points, points, "1.5", "NO_LOCATION", "ANGLE", "PLANAR")
>>> # initialise a field for cursor
>>> fields = ["NEAR_DIST"]
>>> # loop through data with a cursor and delete rows which have no neighbour
within 1.5 meters (to delete distant points with no neighbours which are not part
of road surface marking)
>>> with arcpy.da.UpdateCursor(points, fields) as cursor:
    for row in cursor:
        if row[0] == -1:
            cursor.deleteRow()
```

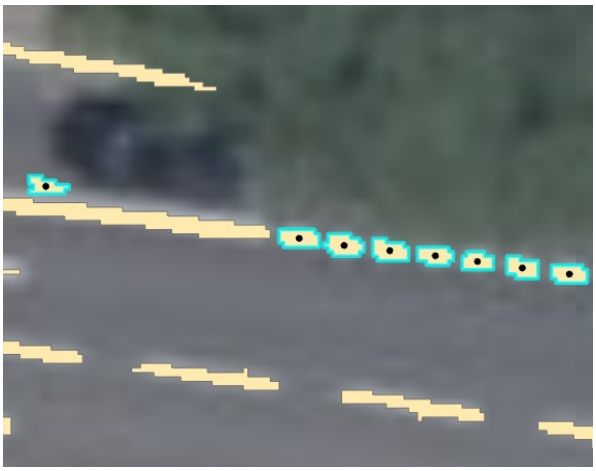

Figure 33: Isolated object (polygon with blue border and point in black) to be deleted

### Step 5: Connecting points into lines

Points created in Step 4 are to be connected into lines. However, one more operation is needed to differentiate among different lines. If there are more broken lines in the analysed image, it is needed to differentiate which point belongs to which line. Otherwise, all points would be transformed into one line and that would obviously be an incorrect solution.

The determination is performed using a buffer zone around the points with a radius of 0.75 meters. Such buffer zones are merged into one polygon (no matter its shape) in case that they overlap and each of such merged polygons has its own unique attribute identifier (FID). This unique identifier is copied to points which lie within the respective polygon and serves as separation criterion for creating lines.

```
>>> # make dissolved buffer around centroids of road surface marking (with
radius 0.75 m)
>>> arcpy.Buffer_analysis(points, buffer, "0.75", "FULL", "ROUND", "ALL")
>>> # conversion from dissolved multipart buffer to single part buffer
features
>>> arcpy.MultipartToSinglepart_management(buffer, singlepart)
>>> # divide centroids according to which buffer they belong to (makes
difference if there are more lines which are not supposed to be connected)
>>> arcpy.SpatialJoin_analysis(points, singlepart, points2,
"JOIN_ONE_TO_MANY", "", "", "WITHIN", "", "")
>>> # create lines from centroids (separate output lines for separate marking
lines based on previous spatial join)
>>> arcpy.PointsToLine_management(points2, line, "JOIN_FID", "", "")
```

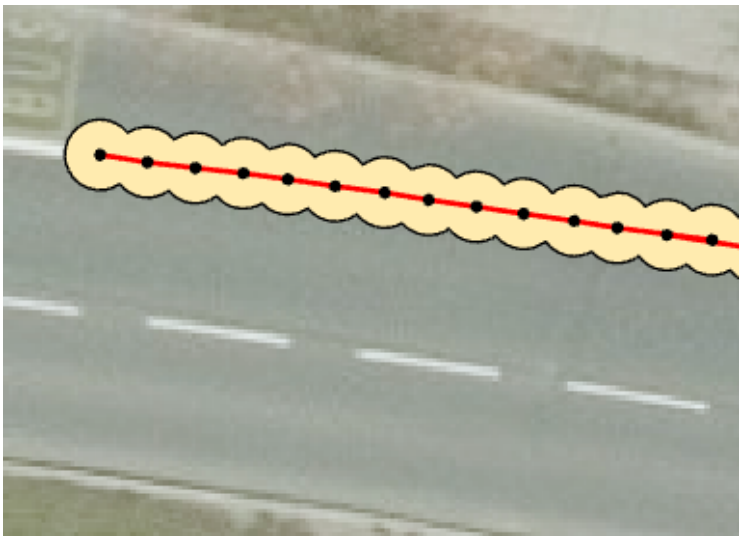

Figure 34: Connecting points into lines: original points (black dots), merged buffer zones (orange polygon with a black border) and resulting line (red)

### Step 6: Bridging short gaps

It can happen due to classification shortcomings that the created lines will have short gaps in between which would not create a topologically correct solution. Therefore the same approach of short gaps bridging as in Steps 7 and 8 of 5.3.1. is used here. First, gap vertices are converted to points and their neighbourhood points within three meters (maximal allowed gap length) are identified. After that, vertices are interconnected with their nearest neighbour with a line which is added to the output dataset.

```
>>> # convert dead ends of RSM lines into points
>>> arcpy.FeatureVerticesToPoints_management(line, vertices, "DANGLE")
>>> # connect edit points with their nearest located endpoints within 3 m
   (maximum bridging distance)
>>> arcpy.Near_analysis(vertices, vertices, "3", "LOCATION", "NO_ANGLE",
    "PLANAR")
>>> # define attributes used in cursor
>>> fields = ['SHAPE@X', 'SHAPE@Y', 'NEAR_FID', 'NEAR_X', 'NEAR_Y']
>>> #initialise index i
>>> i = 1
>>> # loop through all gaps
>>> with arcpy.da.SearchCursor(vertices, fields) as cursor:
>>>      for row in cursor:
>>>          # if the edit vertex does have a closely located end vertex
>>>           if row[2] > 0:
>>>              # initialise empty list of points
>>>              pointGeometryList = []
>>>              # initialise empty point object
>>>              point = arcpy.Point()
>>>              # create point object with the coordinates of gap edit point
>>>              point.X = row[0]
>>>              point.Y = row[1]
>>>              pointGeometry = arcpy.PointGeometry(point)
>>>              # add the point to the list
>>>              pointGeometryList.append(pointGeometry)
```

```
>>>                 # initialise empty point object
>>>                 point2 = arcpy.Point()
>>>                 # create point object with the coordinates of gap end point
>>>                 point2.X = row[3]
>>>                 point2.Y = row[4]
>>>                 pointGeometry = arcpy.PointGeometry(point2)
>>>                 # add the point to the list
>>>                 pointGeometryList.append(pointGeometry)
>>>                 # name the line properly according to its index
>>>                 name = "aa" + str(i)
>>>                 # create line with proper name of the points in the list
>>>                 arcpy.PointsToLine_management(pointGeometryList, name, "",
                    "", "NO_CLOSE")
>>>                 # append the line to existing dataset of original lines
>>>                 arcpy.Append_management(name, line, "NO_TEST", "", "", "")
>>>                 arcpy.Delete_management(name)
>>>                # increase index
>>>                 i = i + 1
>>>             # if there's no closely located end, increase index and go on
>>>             else:
>>>                 i = i + 1
>>> # dissolve the output line
>>> arcpy.Dissolve_management(line, out, "", "", "SINGLE_PART", "")
```

**Step 7: Simplifying generated line and attributes**

Generated lines suffer from the fact that they consist of single points which might not form a smooth curve and need to be simplified. Simplification is performed with the same inbuilt simplification tool using Douglas–Peucker algorithm with the simplification tolerance of 0.3 meters. After that, a new attribute of the dataset is created (TypeLine) and set for each object determining the type of lines.

```
>>> # simplify resulting lines
>>> arcpy.SimplifyLine_cartography(out, output, "POINT_REMOVE", "0.2",
"RESOLVE_ERRORS", "NO_KEEP", "CHECK", "")
>>> # add attribute field to the output
>>> arcpy.AddField_management(output, "TypeLine", "TEXT", "", "", "20", "",
"NULLABLE", "REQUIRED", "")
>>> # fill the attribute field with line type "broken"
>>> with arcpy.da.UpdateCursor(output, "TypeLine") as cursor:
>>>     for row in cursor:
>>>         row[0] = "broken"
>>>         cursor.updateRow(row)
```

## 5.5. RSM attributes detection

To distinguish between broken and solid lines, following algorithm was created and incorporated in the process. As input data it uses two datasets created with previously described algorithms and it creates a line representing only those segments of RSM where a broken line is drawn on the road surface. The algorithm does not distinguish different types of broken line in the way if it separates two lines in the same direction/different direction or a normal line/acceleration lane etc.

### Step 1: Identifying sampling breakpoints on RSM lines

Sampling breakpoints are those points in which RSM changes from broken to solid or vice versa. Such breakpoints are crucial for the algorithm because they will form a boundary between line segments representing a broken line on one side of the point and line segments representing a solid line on the other side. To find these start/end points, the RSM lines (created in 5.3.1.) are converted to vertices and driving lanes boundaries (created in 5.3.2.) are split in these points into new line segments. For the purpose of connecting correct points into lines later in the process (Step 3 of this chapter), the unique identifier of input RSM lines is joined to the points.

```
>>> # create vertices (points) at start and end points of road marking lines
>>> arcpy.FeatureVerticesToPoints_management(mline, points, "BOTH_ENDS")
>>> # split polylines of borders of driving lanes at start/end points of road
marking lines (the boarders represent result geometry but it is needed to
have vertices in the locations of start/end points of road marking lines)
>>> arcpy.SplitLineAtPoint_management(rsm, points, splittedrsm, "0.5")
>>> # join attribute information to start/end points about to which line they
belong (to be used in PointsToLine_management)
>>> arcpy.SpatialJoin_analysis(points, rsm, ptsjoined, "JOIN_ONE_TO_MANY",
"KEEP_ALL", "", "CLOSEST", "0.3", "")
```

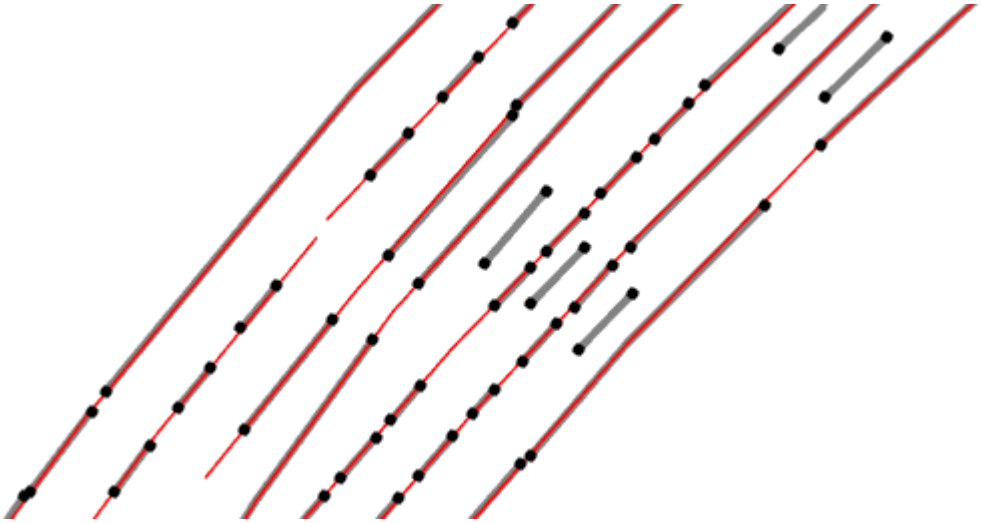

Figure 35: Illustration of step 1: RSM lines (grey), boundaries of driving lanes (red) and vertices of RSM lines (black points).

### Step 2: Selecting RSM line segments

The first selection selects only those segments which represent RSM lines drawn on the road and were classified as RSM polygons. The second selection selects from the selected segments (first selection of this step) only those segments with a proper length (according to Table 1).

```
>>> # select only segments of borders of driving lanes representing
identified lines of road surface marking and not gaps
>>> rsmselect = arcpy.SelectLayerByLocation_management(splittedrsm,
"HAVE_THEIR_CENTER_IN", mline, "0.3", "NEW_SELECTION", "NOT_INVERT")
>>> # select only segments of borders of driving lanes representing broken
lines and not parts of solid lines (therefore length parameters)
>>> rsmselect2 = arcpy.SelectLayerByAttribute_management(rsmselect,
"SUBSET_SELECTION", """ SHAPE_Length > 2.5 AND SHAPE_Length < 3.5 OR
SHAPE_Length > 1.2 AND SHAPE_Length < 1.8 OR SHAPE_Length > 5.5 AND
SHAPE_Length < 6.5""", "NON_INVERT")
```

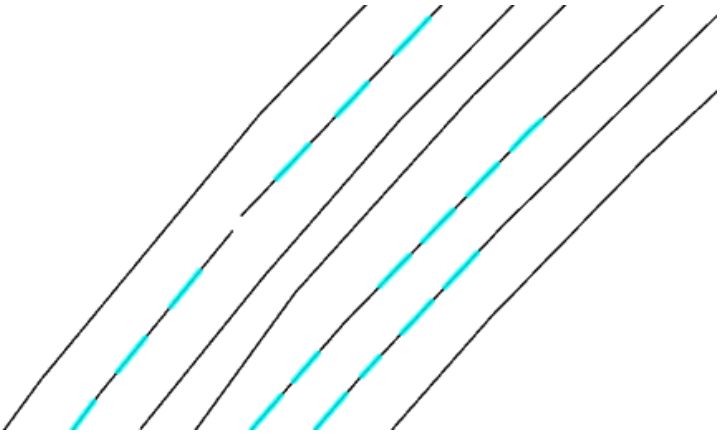

Figure 36: Selected RSM line segments (blue) compared to the rest of the dataset (black)

### Step 3: Creating a single broken line

Based on the location of the line segments selected in the previous step, breakpoints created in Step 1 are selected. From them a polyline is created which represents a boundary between driving lanes (an actual broken line). For the selection of breakpoints, a "within a distance" criterion is used with a distance of 0.3 meters from the selected lines (result of Step 2). Finally, selected points are converted to a line.

```
>>> # select only those start/end points of road marking lines which are
close to selected segments of borders of driving lanes from previous step
>>> ptsselect = arcpy.SelectLayerByLocation_management(ptsjoined,
"WITHIN_A_DISTANCE", rsmselect2, "0.3", "NEW_SELECTION", "NOT_INVERT")
>>> # create a line from selected points
>>> arcpy.PointsToLine_management(ptsselect, brokenline, "JOIN_FID", "",
"NO_CLOSE")
```

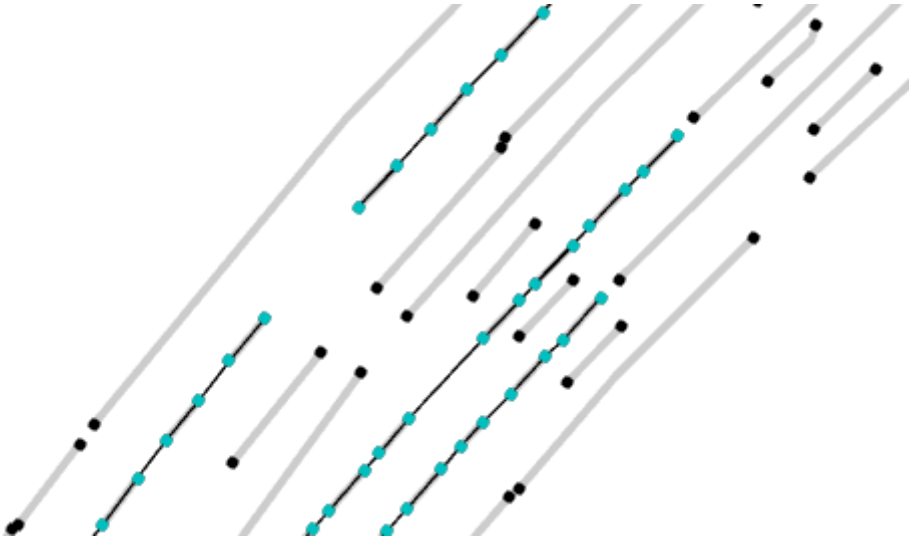

Figure 37: Illustration of Step 3: selected breakpoints (blue), other breakpoints (black), original RSM lines (grey) and created broken lines (black).

### Step 4: Editing generated lines

The generated line is first simplified with an inbuilt tool using Douglas–Peucker algorithm and then cut into segments in vertices selected in Step 3.

```
>>> # simplify resulting lines
>>> arcpy.SimplifyLine_cartography(brokenline, simplified, "POINT_REMOVE",
"0.3", "RESOLVE_ERRORS", "NO_KEEP", "CHECK", "")
>>> # cut line in selected vertices of RSM
>>> arcpy.SplitLineAtPoint_management(simplified, ptsselect, splitted, "0,3")
```

### Step 5: Creating resulting line

First, an empty feature class is created to store the result and then using a loop over all line segments only those with appropriate length are copied to the resulting feature class. The length criterion is set to "> 1 m" to remove short gaps with no connection to broken lines and "< 10 m) to remove long segments of solid lines.

```
>>> # create empty feature class to store the result
>>> out = Arcpy.CreateFeatureclass_management("C:\\... ", "out", "POLYLINE",
"", "DISABLED", "DISABLED", "5514", "")
>>> # initialise fields for cursor
>>> fields = ['SHAPE_Length', 'SHAPE@']
>>> # copy only those line segments with appropriate length (either spaces
between broken lines (> 1 m) or broken lines (< 10 m)
>>> with arcpy.da.SearchCursor(splitted, fields) as cursor:
>>>     for row in cursor:
>>>         if row[0] > 1 and row[0] < 15:
>>>            arcpy.Append_management(row[1], out, "NO_TEST", "", "", "")
>>> # dissolve resulting lines
>>> arcpy.Dissolve_management(out, output, "", "", "MULTI_PART",
"UNSPLIT_LINES")
```

### Step 6: Deleting redundant lines and assigning attributes

Finally, an attribute field is created in the table and filled in for all lines as well as redundant isolated lines which were created because of acceptable length of their segments are deleted.

```
>>> # add attribute field to output
>>> arcpy.AddField_management(output, "TypeLine", "TEXT", "", "", "20", "",
"NULLABLE", "REQUIRED", "")
>>> # final edits (attributes + deleteting too short features)
>>> with arcpy.da.UpdateCursor(output, ["TypeLine", "Shape_Length"]) as
cursor:
>>>     for row in cursor:
>>>         # delete lines which are too short (< 15 meters)
>>>         if row[1] < 15:
>>>             cursor.deleteRow()
>>>         # fill the attribute field with line type "broken"
>>>         else:
>>>            row[0] = "broken"
>>>            cursor.updateRow(row)
```

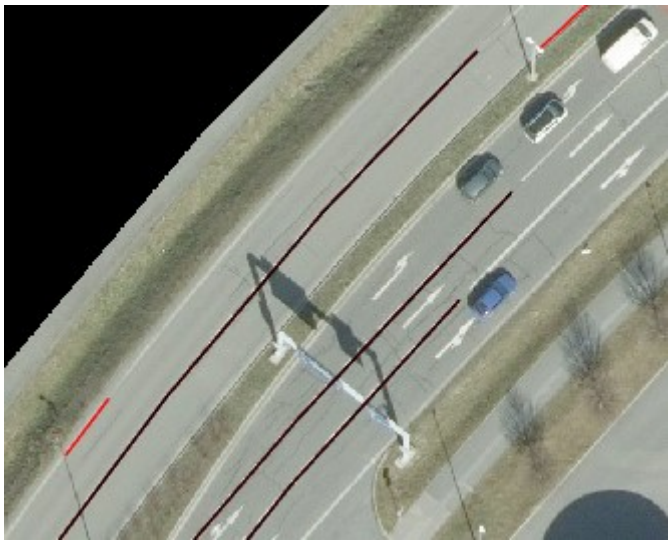

Figure 38: Resulting broken line (black) and redundant line segments which were deleted in Step 6 (red)

# 6. Results

In this chapter results of developed algorithms described in Chapter 5 are presented together with a discussion of eventual shortages and limitations of the process as well as possible solutions to them are suggested. All examples presented in this chapter are visualised results of the algorithms with input data described in 5.1.

The chapter is organised as follows: first, it is described which datasets and what case study area were used for the analysis (and for which reasons); secondly, used software (and relevant technical circumstances) are presented. After that, the successive partial results of single algorithms are presented (in the same logical order as the algorithms are presented in Chapters 5.2 to 5.5). Finally, in the end of the chapter, positional accuracy of the result is summarised and described.

## 6.1. Input data

Two input datasets were necessary to evaluate the proposed process successfully – aerial imagery and a vector dataset of road infrastructure. The main aim when choosing specific data sources was to choose the most universal dataset with no specific characteristics. Similar data should be available with as wide coverage as possible and thus should allow wide application of the proposed method. The only limitation would then be different rules and norms for RSM lines shape, width, texture and colour.

First input dataset is aerial imagery with one necessary requirement – minimum spatial resolution. For a successful identification of RSM on the road surface it is demanded to have a maximal pixel size of 0.125 m (12.5 cm). There are several datasets which are country-wide, fulfil the spatial resolution criterion and are publically presented online on geoportals or mapping platforms such as data from TopGIS, s. r. o. published on mapy.cz (https://mapy.cz/letecka?), maps by Google, Inc. (https://www.google.cz/maps/) or imagery from the Czech Office for Surveying, Mapping and Cadastre (https://geoportal.cuzk.cz/geoprohlizec/?wmcid=22524).

In the end none of the nation-wide imagery sources presented above were chosen. Instead, data from Prague Institute of Planning and Development was chosen. The data is available for student assignments free of charge and can be downloaded and saved to a local repository. Furthermore, the Institute offers a dataset which was created during non-vegetative season and therefore limits shadows of trees. The spatial resolution of such dataset is 0.1 m (10 cm) with the data coverage for the area of Prague.

The second dataset does not need to meet many special requirements either. The main requirements are spatial resolution and accuracy, plus useful information about a type of road (motorway/normal road) and overpasses/underpasses; however, such information can be added manually if necessary.

Such information is contained in OpenStreet Map data which have worldwide coverage (with changing quality characteristics such as completeness and correctness though). However, for areas in Central Europe, the quality characteristics are sufficient and therefore, the OSM roads dataset was used.

Figure 39: Difference between a vegetative season imagery (left) and non-vegetative season imagery (right)

## 6.2. Case study area

There were several conditions for selecting the case study area which cover:

(1) sufficient (yet not too high) complexity of roads and driving lanes including junctions, curves, multiple lanes in one direction, changing types of RSM lines, overpasses and underpasses etc.

(2) there must be RSM lines marked on the roads

(3) sufficient quality of RSM in the images (RSM lines need to be visible enough to be classified successfully)

(4) sufficient coverage of RSM in the image (RSM lines must not be shaded too much by buildings, trees or cars for most of the algorithms)

(5) there must not be bright (mainly concrete) surfaces since these would cause a lot of misclassified polygons (their reflectance and shape might be too close to white RSM polygons' reflectance)

An area of Střížkov, Prosek, Letňany and Ďáblice city quarters in the north-east of Prague, Czechia, was chosen for which appropriate input datasets were available. The area contains a motorway as well as major roads and minor roads with multiple driving lanes within both urban area and non-urban landscape. Furthermore, the author knows the chosen locality well (and could eventually easily visit it for field inspections). Multiple (20) areas with average size of 5,083 m² were chosen which cover various different types of RSM and road infrastructure including a motorway, over- and underpasses and multiple (changing) driving lanes. Chosen localities are visualised in Figure 40 and an overview of all case study areas is presented in Table 5.

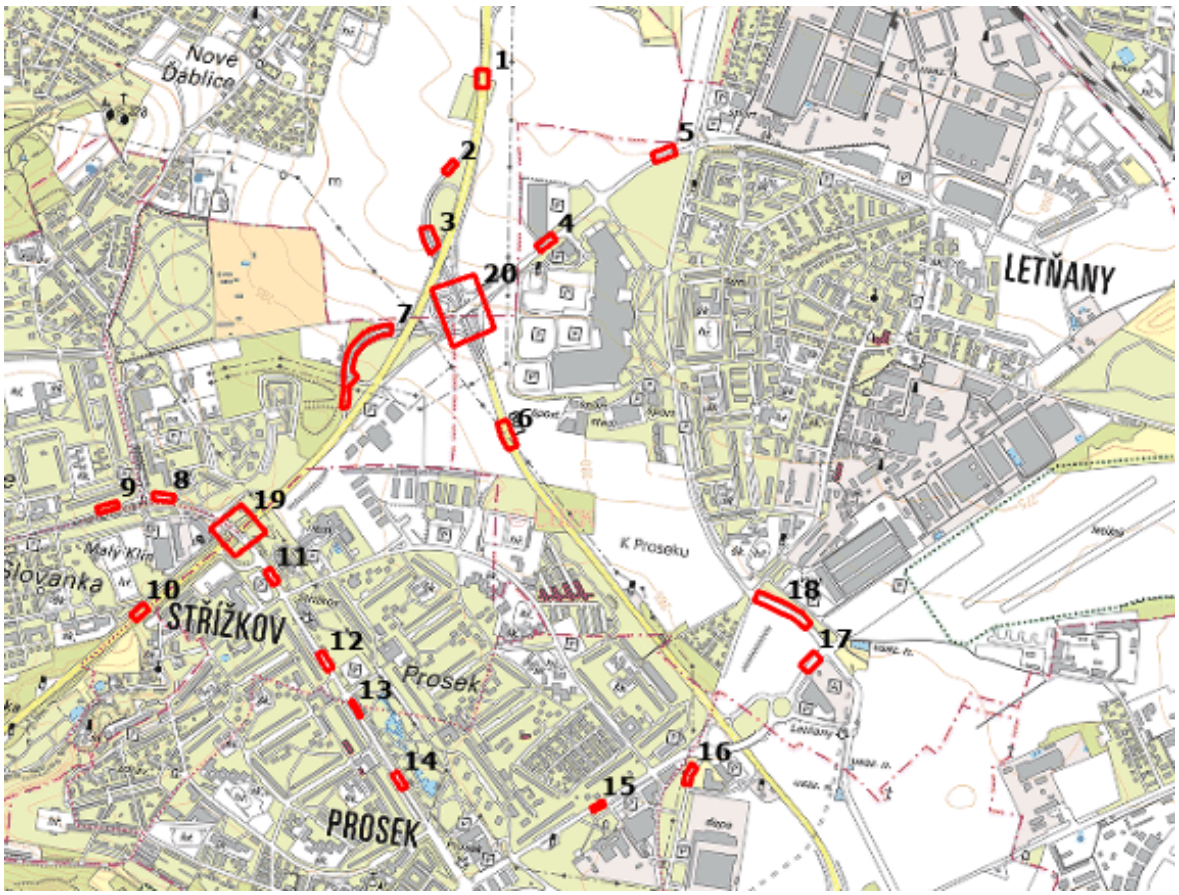

Figure 40: Overview of case study areas

| Number | Area [m$^2$] | Description |
|---|---|---|
| 1 | 2655 | Motorway with three driving lanes in each direction |
| 2 | 1170 | Motorway slip road with two driving lanes |
| 3 | 3554 | Motorway slip road with two driving lanes in each direction |
| 4 | 2016 | Major road with five driving lanes (incl. a turning lane) |
| 5 | 2794 | Major road with four driving lanes (incl. a turning lane) |
| 6 | 3886 | Major road with separated directions (two driving lanes each) |
| 7 | 11272 | Motorway slip road with two driving lanes in each direction |
| 8 | 1906 | Major road with three driving lanes (incl. a turning lane) |
| 9 | 1692 | Major road with two driving lanes |
| 10 | 1848 | Motorway with two driving lanes in each direction |
| 11 | 1457 | Major road with four driving lanes |
| 12 | 1791 | Major road with four driving lanes |
| 13 | 637 | Major road with two driving lanes |
| 14 | 1480 | Major road with four driving lanes |

| Number | Area [$m^2$] | Description |
|---|---|---|
| 15 | 848 | Major road with two driving lanes |
| 16 | 1803 | Minor road with two driving lanes |
| 17 | 2673 | Major road with three driving lanes (incl. a turning lane) |
| 18 | 7482 | Major road with five driving lanes (incl. a turning lane and bus stops) |
| 19 | 17899 | Motorway overpass |
| 20 | 32789 | Motorway overpass |

Table 5: Case study areas with description

## 6.3. HW and SW requirements

To run the proposed algorithms one does not need specialised hardware. All algorithms were developed and tested on a personal laptop with an Intel Celeron B800 processing unit (1.5 GHz) with an operating system of Windows 10 Home 64 Bit.

As for software, three different programs were used during the process. The code was written in PyCharm 2019.3.3 (Community Edition), powered by open-source software and JetBrains s.r.o. Chosen scripting language was Python 3.6 (the most up to date version of Python at the time of writing). The code itself uses an ArcPy library which offers access from Python to all geoprocessing tools of ArcGIS Pro (ESRI 2020). The library as well as the ArcGIS Pro is licensed.

e-Cognition Developer version 9.5.0. by Trimble was used for classification and segmentation. A powerful desktop computer with a licence was used to run both processes.

Visualising, data editing, data control, accuracy measurements as well as the rest of geospatial operations were done in ArcGIS Pro 2.5.0 on the same personal laptop as running algorithms.

## 6.4. Classification results

The classification results are presented in Figure 41. The output of the classification was inspected manually to evaluate the result. Minor edits were necessary to delete redundant polygons which had similar characteristics and thus were classified as RSM polygons despite the fact that they represent cars, ruts, parts of buildings or concrete pavements. Yet, manual inspection is still a common (and necessary) part of research workflow when working with automatically classified data in order to obtain relevant results.

Several causes of misclassification were identified in the input images and are discussed here. They can be, in general, divided into two groups – (1) objects commutable with RSM close to or on the road surface and (2) objects misclassified in the image with no relationships to roads.

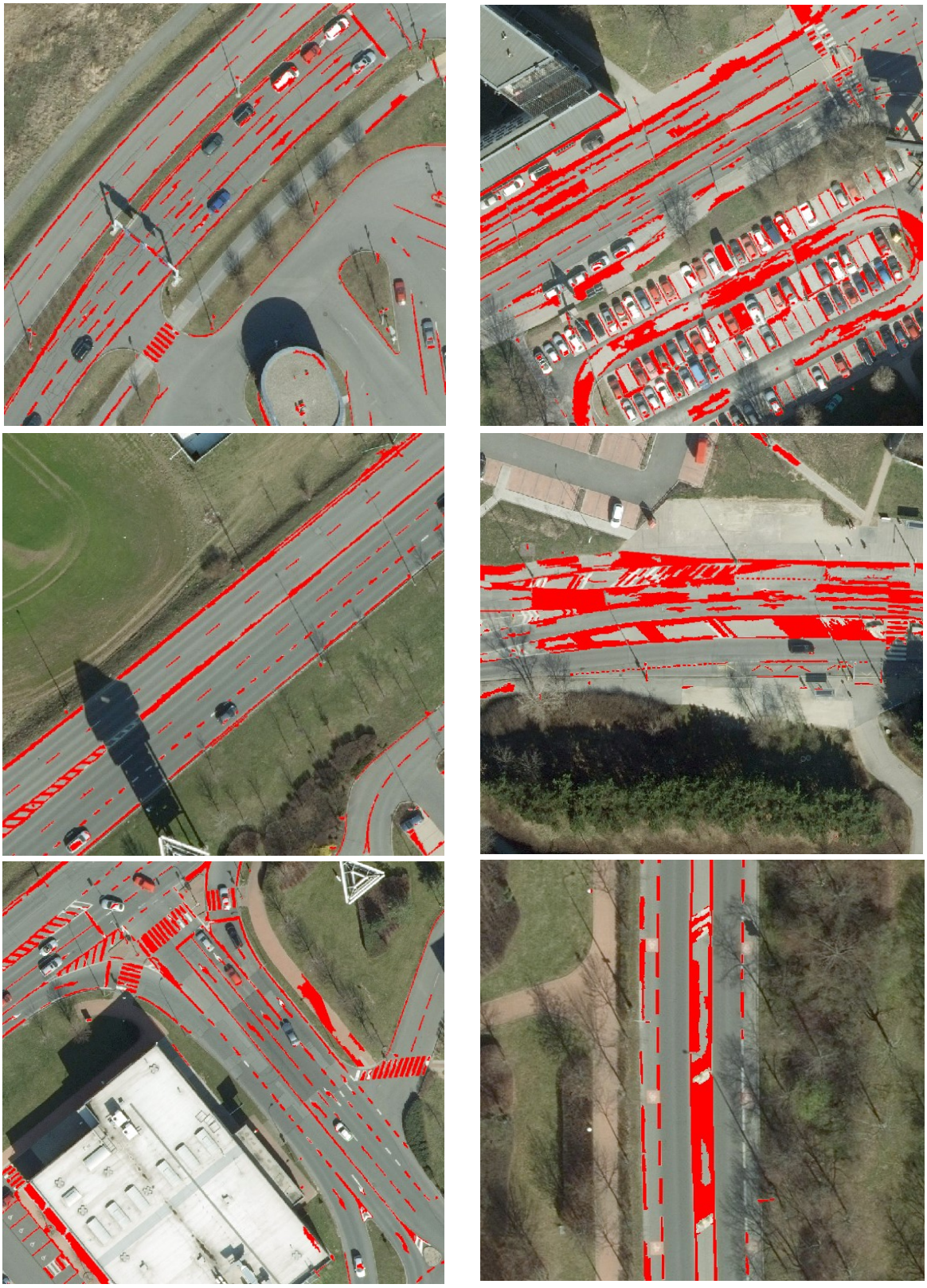

Figure 41: Results of classification, classified RSM polygons in red (correct classification results on the left, incorrect on the right)

Objects commutable with RSM which are depicted in the image close to the roads represent traffic signs, railings, crash barriers, drainage covers and other parts of the road surface and its surroundings. These would be very difficult to eliminate as long as they have similar shape and brightness value as RSM lines.


Figure 42: Objects commutable with RSM: (a) advertising billboards, (b) toll gates or directional sign stands or (c) cars and trailers

Secondly, dust, grit and dirt covering edges of roads have higher reflectance than tarmac or concrete and were misclassified as RSM (forming wide edges of driving lanes, especially on the edge of roadways). Such polygons do not have a significant influence on the quality of the result because the proposed method is able to handle wider input polygons that RSM lines thanks to merging. These 'dust polygons' (Figure 43) result only in a minor positional inaccuracy of resulting RSM lines; in some cases they are even beneficial as a substitution for RSM lines which are not visible enough or are not even drawn on the road surface.

Objects misclassified in the image with no relationship to roads such as buildings, pavements or other objects with similar brightness and shape characteristics are to be removed from the classified RSM which represents a possible improvement of the current method. The removal (within roads buffer with equal diameter) was done manually because no road data with wide international coverage containing such information in attributes were found.

However, the removal could be automated by excluding all polygons identified as RSM with no spatial connection to roads (i.e. which do not lie within a specified distance from the lines of road axes). To do this, information about road width (for all road polylines) is necessary as well as higher positional accuracy of road centrelines (the line has to always be in the middle of the road). Another way of reducing manual input here would be performing a two-step classification – classifying only roads in the first step and identifying white (and yellow) RSM only after that in the second step (see Chapter 2 for details).

Figure 43: 'Dust polygons' misclassified as RSM can have both beneficial (left) and undesirable (right) effect

As can be seen in Table 6 the amount of manual edit differs in different case study areas. This is caused by the fact that different areas cover different types of surface and therefore different classification shortcomings appear. A brief explanation of the most significant shortcomings is further discussed.

| Case study area | Classified RSM before manual edits (Area [$m^2$]) | Classified RSM after manual edits (Area [$m^2$]) | Change ratio [%] |
|---|---|---|---|
| 1 | 94,33 | 89,17 | 5,47 |
| 2 | 37,28 | 36,38 | 2,40 |
| 3 | 279,83 | 235,48 | 15,85 |
| 4 | 139,58 | 110,37 | 20,93 |
| 5 | 238,11 | 154,72 | 35,02 |
| 6 | 312,85 | 225,07 | 28,06 |
| 7 | 967,38 | 533,02 | 44,90 |
| 8 | 109,13 | 87,26 | 20,04 |
| 9 | 34,81 | 32,11 | 7,75 |
| 10 | 148,85 | 131,62 | 11,58 |
| 11 | 300,80 | 78,95 | 73,75 |
| 12 | 226,21 | 82,14 | 63,69 |
| 13 | 150,08 | 54,89 | 63,43 |
| 14 | 90,92 | 90,12 | 0,88 |
| 15 | 62,07 | 50,30 | 18,96 |
| 16 | 132,42 | 71,03 | 46,36 |

| Case study area | Classified RSM before manual edits (Area [$m^2$]) | Classified RSM after manual edits (Area [$m^2$]) | Change ratio [%] |
|---|---|---|---|
| 17 | 164,76 | 123,31 | 25,16 |
| 18 | 1074,65 | 547,09 | 49,09 |
| 19 | 641,05 | 297,36 | 53,61 |
| 20 | 1865,64 | 1385,55 | 25,73 |

Table 6: Amount of manual edits after classification

The areas which have the change ratio lower that 10 % needed only minor changes such as removing small patches of sunlight reflections from cars, drainage covers or small objects which were classified as RSM (Figure 44).

Figure 44: Small patches of sunlight reflection and other misclassified objects (red)

Areas with high numbers of change ratio show a need for a lot of manual edits. However, the reason for a vast amount of misclassified segments is the same in all three areas – a concrete pavement with high reflectance in bright sunlight with long and narrow shape (and thus the same length/width ration as RSM lines). The situation is illustrated in Figure 45 on the following page.

The rest of case study areas did not need a vast amount of manual editing. Furthermore, the algorithms are built in a way so that they do eliminate minor classification errors such as small and remote misclassified patches. However, deleting pavements classified as RSM lines is necessary for a correct result.

Figure 45: A concrete pavement misclassified as RSM lines

## 6.5. RSM identification

The aim of the first algorithm described in 5.3.1. is to create lines which represent RSM. Input data are classified polygons of RSM (after correcting classification errors manually). The algorithm produces correct results in general with the overall positional accuracy of 0.126 meters (as presented in Section 6.10.). However, several shortcomings were identified which are further discussed on the following pages.

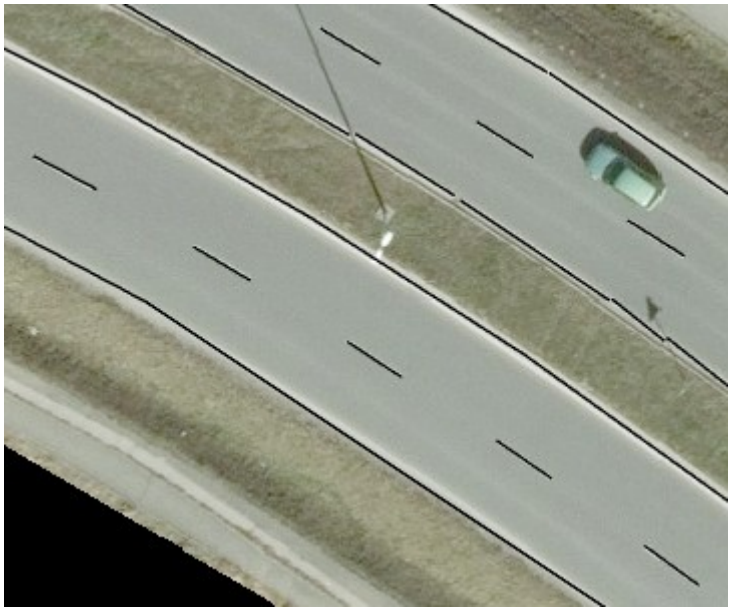

Figure 46: Identified RSM in an image

### Diagonal hatching

The algorithm creates oscillating lines occasionally when a straight line touches diagonal hatching lines. Resulting line is simplified by Douglas–Peucker method yet with connections to diagonal hatching lines the overall shape is not changed much after simplification.

Figure 47: Diagonal hatching: oscillating line (on the left) and correctly created straight line (on the right)

One possible solution would be to remove all diagonal hatching lines from RSM lines before the simplification is applied and then add them into the simplified layer again. However, then the hatching lines would not be touching the straight line and there would be gaps within RSM marking lines.

The solution depends on specific usage (and thus needs) of the resulting data. If all RSM lines (including diagonal hatching) are supposed to be in the result as a properly interconnected network, then the network either has to be simplified together (as it currently is) or all gaps created by straight line simplification have to be bridged as another step. In this paper, the firstly presented result (interconnected network with occasional line oscillation) is preferred.

### Crash barriers and outer separators

There are several road infrastructure objects which look in aerial images like RSM lines but represent other objects than RSM. These are, among others, crash barriers (both metal and concrete ones), outer separators, toll gates, directional sign stands, acoustic walls or curbs. Some of them (road curbs for example) are useful (and desirable) if being classified as RSM because as a matter of fact they represent an edge of the road. In some cases where the RSM lines are missing, they serve as a substitute of missing lines.

However, when such substituting objects are further from the line (or the place where the line should have been drawn) they cause a spatial shift of the created line from the position of the actual border of a driving lane to the outside of the paved surface (roadway shoulder).

Figure 48: Metal crash barrier classified as RSM line causing spatial shift of the border a driving lane

The solution of the problem would be to delete such polygons after classification as misclassified redundant features and to approach such areas as gaps and bridge them with suggested bridging algorithms. However, this solution would require more manual editing (assigning gap vertices and inspecting road points) which would lead to a longer editing time.

### Traffic islands

Traffic islands which are marked on the road surface with RSM (including diagonal hatching) and are not physically divided from the road surface with raised kerbs, curbs or other physical obstructions are another problem for the algorithm. In cases that the (for the algorithm input) classified RSM polygons of a straight line and diagonal hatching are not clearly identified.

Figure 49: Divergence of straight RSM lines; comparison between incorrect (left) and correct solution (classified RSM in grey, created RSM lines in red).

If the classification of a straight line is successful, than the resulting RSM lines are created in a correct way. Otherwise, the resulting line contains undesired curves and oscillation in places where the island begins (i.e. where two straight lines of RSM diverge, Figure 49). However, it is possible that in Figure 49 specifically the mistake is caused by the fact that the line divergence is close to the border of the case study area.

Possible solution would be similar to the one presented in the Diagonal hatching section of this chapter with all its advantages and disadvantages.

### Railing

In one case, a protective metal railing with a bright red and white paint appeared in a case study area which was classified as RSM and has length parameters (length of lines and length of gaps with regular alternation). Therefore, the algorithm processed it as a broken line and identified it despite the fact that it is not located on the road surface anymore. The railing was not removed during manual editing due to the same reasons (the author mistook it for a broken line of a right-turn lane).

Figure 50: Protective metal railing with bright red and white paint; from driver's perspective (left) and in aerial image (right)

Possible solution to this is (beside manual removal) automated deleting of all classified objects which are not located on road surface anymore (as suggested in Chapter 6.4.); or implement a two-step classification (see Chapter 5.1. for more detail).

### Straight line oscillation

In some cases the generated RSM line slightly oscillates although the classified RSM polygons are in line and the oscillation is thus a mistake. It was found out that the oscillation is caused due to the operation of line simplification. The simplification tolerance was set

to 0.3 meters to prevent unnecessarily breaking of continuous curves into straight line segments (Figure 53). However, there are vertices on the original line (which is to be simplified) which are already further than 0.3 from the connection of start and end vertices and thus another vertex is created on the resulting lines causing the resulting line to oscillate.

Figure 51: Oscillating resulting line (black) and a line before simplification (red), zoomed in

The problem is caused by missing RSM which is replaced by 'dust polygons' (see Figure 43 for explanation) which are wider and more irregular in shape than RSM. Finding a solution to this problem is complicated and such solutions have their drawbacks. In case of using a bigger simplification tolerance during line simplification smoothness of resulting curves will be limited even more. On the other hand, manual editing would be more time and manpower consuming. The theoretical solution would be to connect these short oscillating lines to their neighbouring lines (then they would be simplified correctly) but this would mean adding more steps to the algorithm; and it is not desirable in all cases to be connecting short gaps between RSM lines.

**Shadows**

Shadows of trees, buildings, traffic lights, lampposts and other object shading road surface and RSM cause the resulting line being cut into multiple shorter line segments (in case of narrow shadows of traffic lights or lampposts) or vanishing completely (trees, buildings and other wide objects).

Effect of shadows can be reduced by choosing imagery with less (shorter) shadows. However, it is not possible yet to eliminate all shadows in images, therefore there are further measures taken in the following algorithms to deal with this problem.

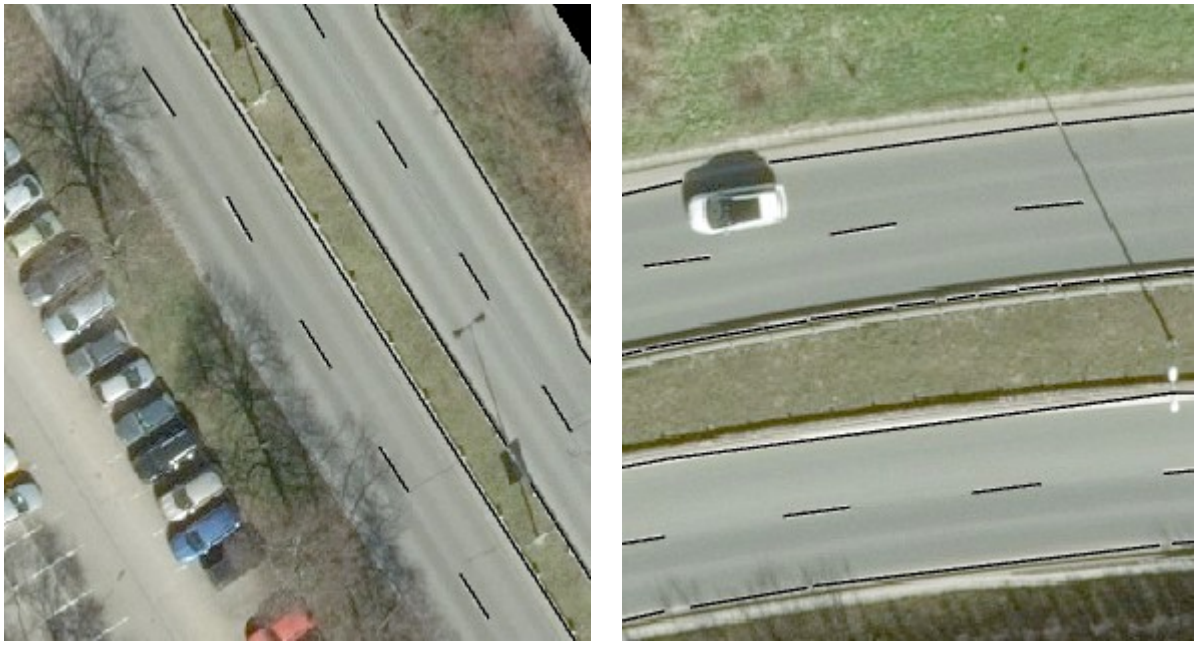

Figure 52: Effect of shadows of trees (left) and lampposts or cars (right) on created RSM lines

**Insufficient arc continuity**

It was found out that in some curves the created RSM line loses its fluency. This effect is caused by line simplification, more precisely by setting a simplification tolerance parameter. The parameter has to be set consciously because it influences complexity of the result and eliminates errors of classification in one hand, yet it decreases fluency of arcs and causes breaking arcs into straight line segments on the other.

Figure 53: Limited fluency in curves; resulting line (black) and a line before simplification (red), zoomed in

This shortcoming is not possible to be fully solved, it can only be mitigated. With a standard width of driving lanes of 3 (3.5) meters and a standard width of passenger cars around 2 meters and of a bus or a truck around 2.5 meters, a 0.3 meters difference in width of a driving lane is still acceptable. Furthermore, it should never happen that RSM lines of one driving lane will be curving towards each other and thus the simplification parameter of 0.3 meters would have to be multiplied twice (i.e. both lines would be further from each other that the simplified line).

### Pedestrian crossings

Proposed algorithm was not designed to model pedestrian crossings correctly as they are not relevant for driving lanes. However, they are marked with white RSM polygons, which mean that they were classified and analysed (unless removed during manual editing) together with the rest of classified polygons. The process did not have to be adjusted to proper modelling of pedestrian crossings, therefore the only solution is to delete them further in the process so that they do not collide with RSM lines and thus do not cause errors.

Figure 54: Pedestrian crossing with resulting RSM lines

### Horizontal directional arrows

Horizontal directional arrows are together with other horizontal marking such as pictograms, horizontal traffic signs, speed limits, text and other relevant information drawn on road surface. Their colour is white in most cases and therefore these objects were correctly classified as RSM during the classification.

However, just like pedestrian crossings, they are not relevant for driving lanes and therefore are to be deleted further in the process so that they do not collide with RSM lines and thus do not cause errors as well.

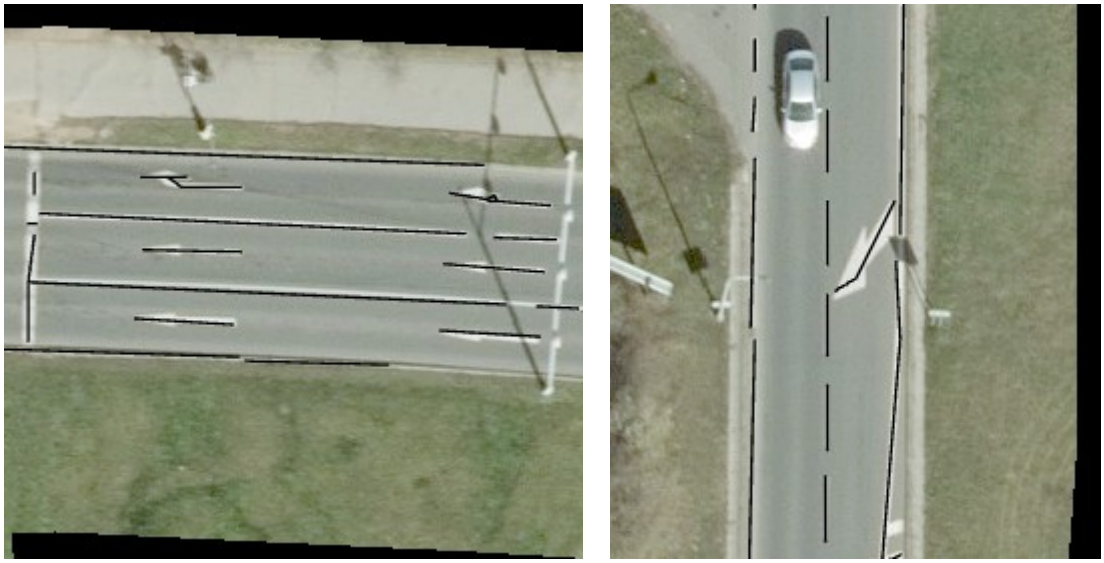

Figure 55: Horizontal directional arrows with resulting RSM lines

**Bus stops**

Bus stops are next to the pedestrian crossings and horizontal directional arrows another type of objects which are marked on road surface and thus correctly classified as RSM. Unlike pedestrian crossings or directional arrows, bus stops can be relevant for RSM and driving lanes in cases that a bust stop forms a bus bay (also called bus lay-by in some regions). Then, it may be desired to create another (although short) driving lane connected to the main driving lane.

To solve this and to fully put bus bays in the database of driving lanes, a more complex solution would be necessary which may lead to another input dataset of bus stops. Furthermore, the classification result would have to be improved because not all segments of bus stop marking were classified as RSM.

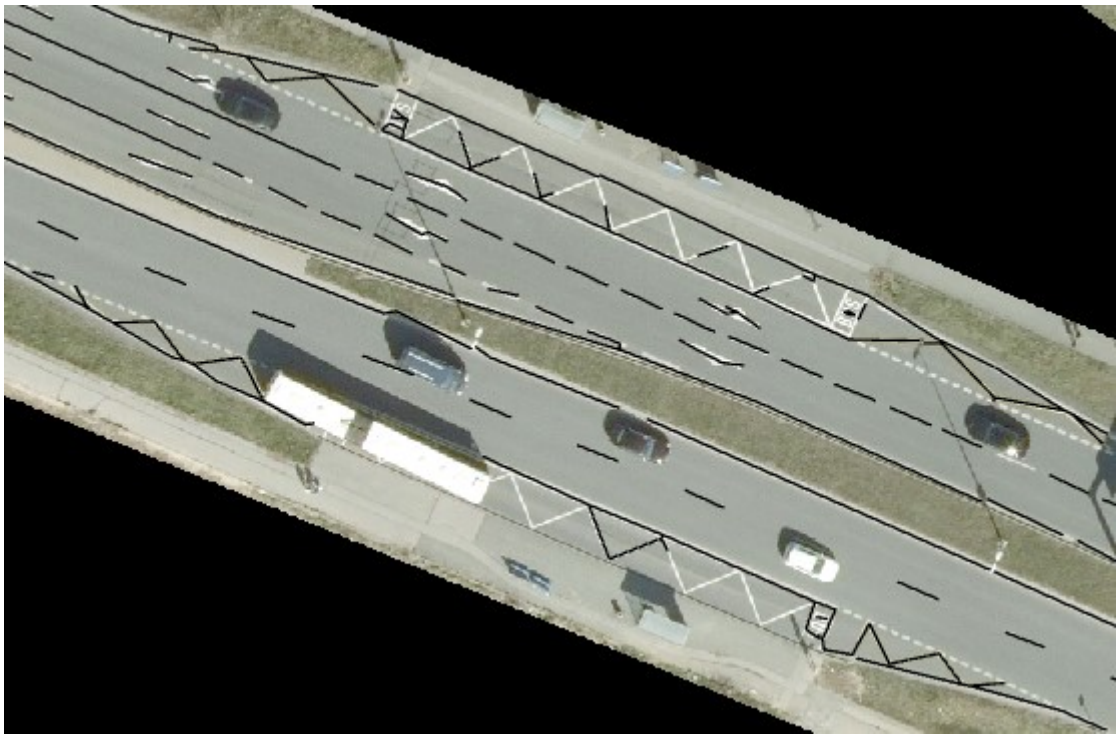

Figure 56: Bus bays with resulting RSM lines

**Gaps with no visible RSM lines**

In places where the real RSM lines are either not drawn on road surface or are fading or covered, no RSM lines were modelled by the algorithm. This represents an important disadvantage of reconstruction of driving lanes boundaries from RSM and solving this issue as presented in 5.3.3. is one of the key contributions of this thesis.

Figure 57: Gap with no visible RSM line (on the very left)

## 6.6. RSM geometry detection

The crucial step on the way from identified RSM lines to driving lanes geometry is the detection of RSM geometry. In other words, in this step single lines that are drawn on road surface are interconnected together into a (in an ideal case) continuous network depicting boundaries of driving lanes.

The algorithm was designed in a way so that it is able to bridge short gaps between neighbouring RSM lines. Yet, there naturally are several shortcomings which are presented I the following section with a brief discussion about their reasons and solutions.

Figure 58: Successfully detected driving lanes (black)

### Incorrect length of RSM lines

Despite the fact that the length of RSM lines is officially defined in a binding document called Technické podmínky Nr. 133, Zásady pro vodorovné dopravní značení na pozemních komunikacích (TP 133), it was found out that the length can differ slightly. The length of a line is an important parameter because it informs drivers about context of the traffic situation (see Table 1 for more details). Several times the identified RSM line was longer (by more than 30 cm) than the official length leading to deleting or not selecting the line as an RSM line.

Figure 59: Incorrect RSM lines length leading to a gap in resulting line

The other extremity (a line too short) is presented in Figure 59 which in combination with a car shadow leads to a resulting gap. The shortening of the line was in this case caused by a road patch.

The algorithm itself was built to handle wrong length of one RSM lines; however, in this case two consecutive lines appear which are both too short – the first one is too short because of the patch and the other one is too short because it is shadowed by a car.

### Changing amount of driving lanes

A situation when two driving lanes merge into one or when one driving lane is splitting into two is appears commonly on roads and therefore should be counted on with. In such cases, the algorithm takes advantage of the fact that the line which creates a border of the starting/ending lane always has a smaller angle that 35° and therefore can be distinguished from diagonal hatching or other undesired objects.

Out of six such situations in case study areas, five were modelled correctly. Only one case (Figure 60) was not modelled correctly which presumably can be caused by two reasons. Firstly, the situation is at the edge of the case study area and the algorithm tends to create worse results at the edges (caused by a loss of neighborhood connections on the side of the border) and secondly, the diagonal hatching in this situation was not classified properly.

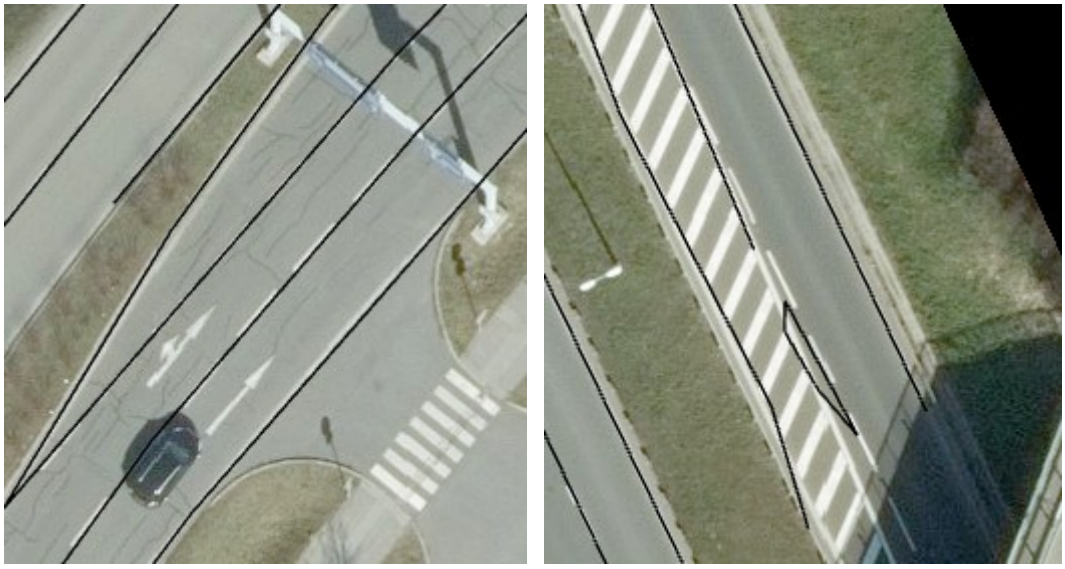

Figure 60: Changing amount of driving lanes, correct (left) and incorrect (right) modelling

### Line oscillation

Three cases of undesired line oscillation were identified in the resulting lines. In all cases, an object was blocking the straight line and was not classified as RSM (which leads to a gap after running the first algorithm to identify RSM lines). In this step, the gap is bridged (if it is shorter than 3 meters); yet if there were other objects in the surrounding of the bridged gap classified as RSM (road debris etc.) than this can cause an oscillation of the resulting line. The resulting line is not simplified in this step; therefore all (even minor) oscillations can appear). Lampposts, toll gates, directional sign stands, traffic lights or cars can cast such shadows as well as objects of drainage covers or patches can cause such oscillation.

Figure 61: Oscillating line caused by a drainage cover (left) and by a shadow of car (right)

### Bus bays

As described in the previous section, bus stops are marked by longer lines (compared to pedestrian crossings lines, which have been eliminated) and might be relevant for driving lanes in case of bus bays. Therefore, their lines were not completely removed.

However, due to not correct classification result, the bus stops tend to have incomplete structure with gaps. A disadvantage of the process is that not all gaps are bridged during the reconstruction process, especially those gaps which are not parallel with road vector lines.

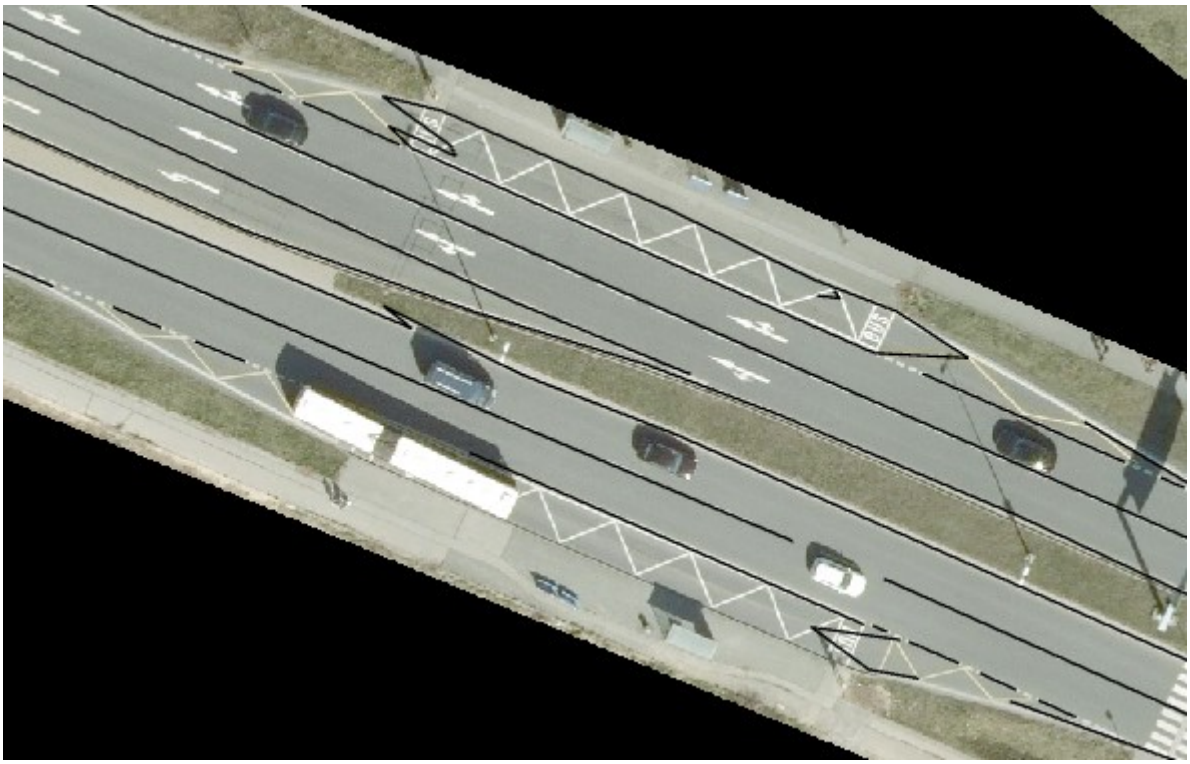

Figure 62: Bus bays with gaps

## 6.7. Bridging gaps

A specialised algorithm was created to bridge gaps longer than 3 meters. It combines information from classified RSM (vertices of modelled RSM lines) with polylines or existing roads (second input dataset) and thus is able to copy the shape of road polylines into resulting lines as well. The algorithm is able to solve the problem of bus bays if the input information about gap vertices is set correctly.

For most of the cases it creates correct solutions in places where the RSM is not marked or was not classified; such cases are presented in Figure 63. However, under several circumstances is the result of the bridging algorithm not fully correct; such cases are described and analysed on the following pages.

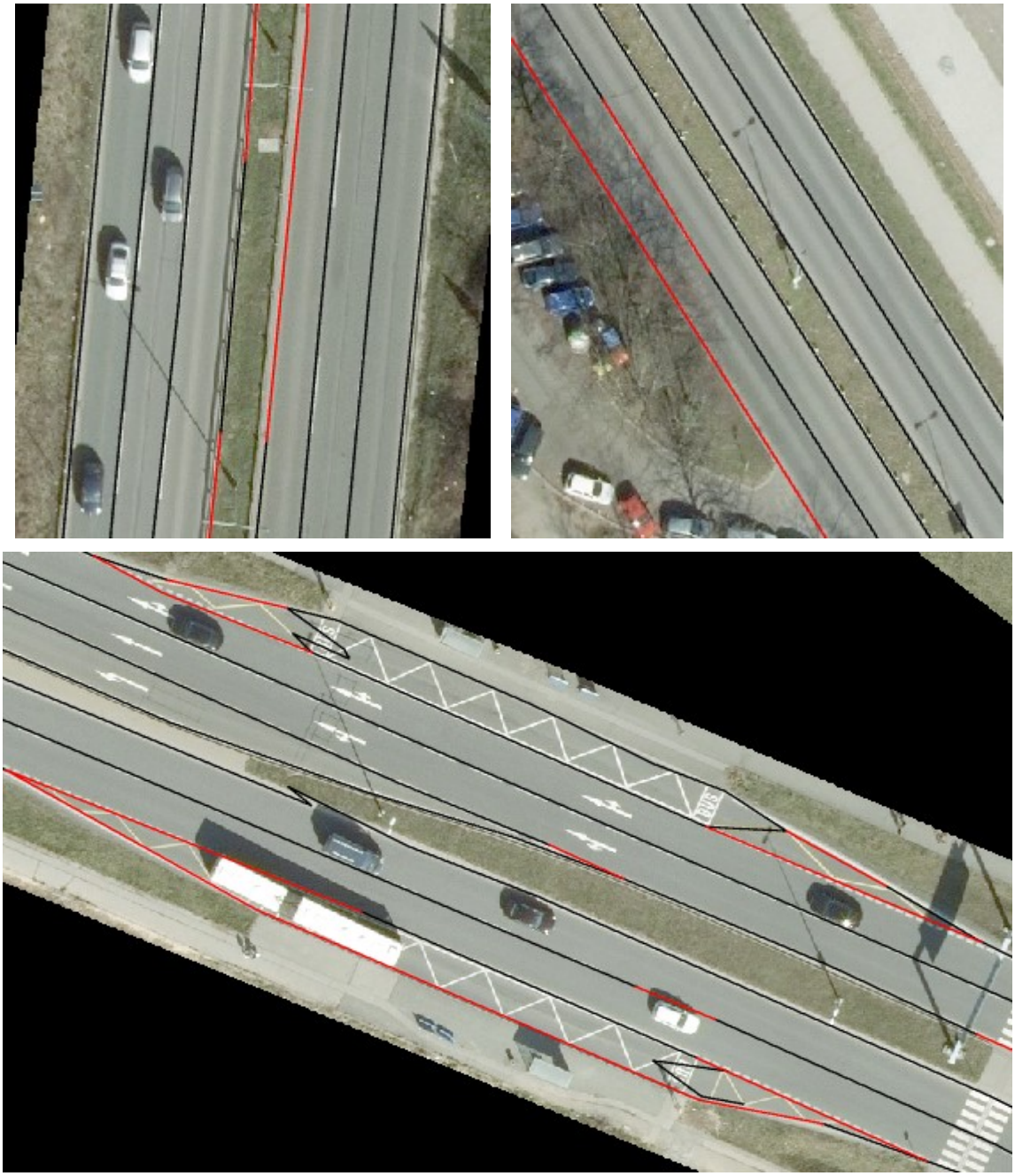

Figure 63: Successfully bridged gaps (red) and previously modelled driving lanes boundaries (black)

### Undesired shifting

Resulting gap line will always copy the shape of input road vector lines. Therefore, if the input line is not straight where the RSM line is straight, the modelled gap will not be straight either.

This shortcoming is dependant on the quality of input data and therefore the input data is supposed to be reviewed if it is fully fit for the purpose.

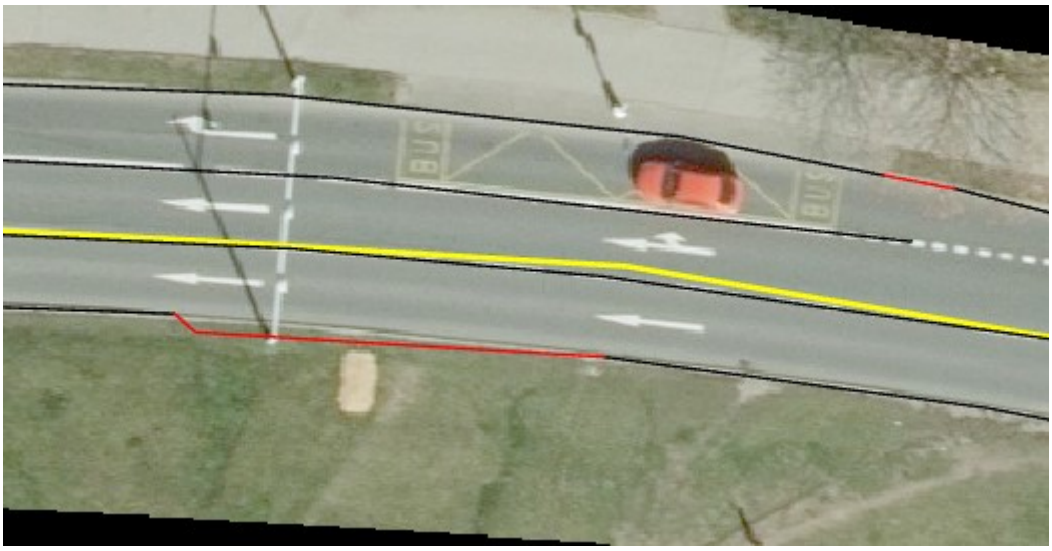

Figure 64: Undesired shifting of the bridging line (in red compared with shape of input road line (in yellow)

### Merging driving lanes

In some cases the created lines did not correctly bridge the gaps which lied in junctions where two driving lanes split/merge. The cause of incorect solution is the fact that the input road line (although logically and topologically correct) is not keeping the same distance from the RSM line throughout the whole gap. Two such situations are presented in Figure 65.

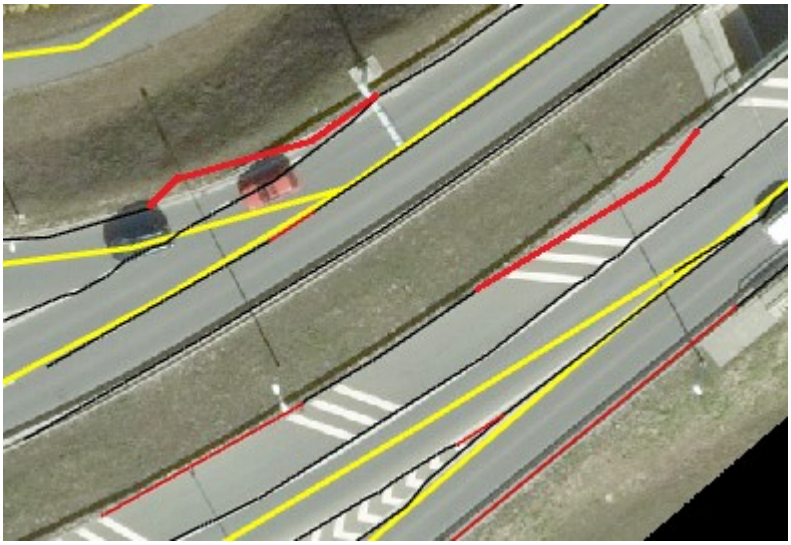

Figure 65: Incorrectly bridged gaps in junctions (bold red) compared to the shape of input road line (yellow) and correctly bridged gaps (thin red lines)

**Missing continuity**

The fact that the resulting bridging line copies the shape of input road line may occasionally cause missing continuity of the result. This has the same cause as the previous problem (in this case only the other way around), the input road lines are not fully copying the correct direction of the road (and thus RSM) and cause lack of continuity in the result as shown in Figure 66.

As stated above, the problem is caused by the geometry (low spatial accuracy) of input data.

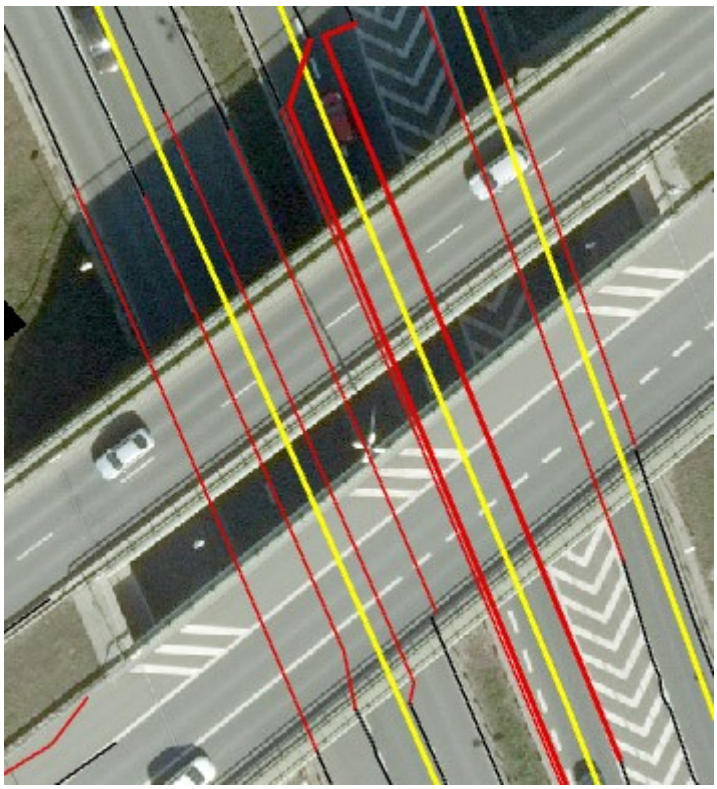

Figure 66: Logically incorrect bridging solution (bold red) compared with shape of input road line (yellow) and the correctly bridged gaps (thin red lines)

## 6.8. Over- and underpasses

One of the most significant contributions of this thesis is a solution of modelling driving lane boundaries even though the RSM line is hidden from view in the input aerial image such as in shadows, under bridges or in tunnels. The solution is simple (as presented in 5.3.3.) and requires fulfilment of two conditions – firstly, a vector data model must be used; and secondly, an input dataset of existing road infrastructure containing information about the direction of roads in areas hidden from view has to be used as a supplementary input dataset.

Both conditions were met in the workflow and thus results are presented. From the problems presented above, one is relevant to be mentioned with underpass motorway junctions – missing continuity. Missing continuity (as stated above) occurs in such situations when

a driving lane which is situated under a bridge does not have the same direction as the vector line representing the road (as illustrated in Figure 66).

Different road levels (underpass and overpass, i.e. surface level and bridge level) were analysed separately and merged into one resulting dataset only after the analysis was performed. The algorithm is not capable of working on more than one surface levels; such adjustment would involve one more loop over levels; besides that a level information would have to be added to attributes of both gap vertices and road points as well as resulting RSM lines.

Two motorway bridges were covered in case study areas of this paper and the results are presented below (Figure 67). They both represent a typical situation where two road levels cross with part of a highway junction covered in the first case as well.

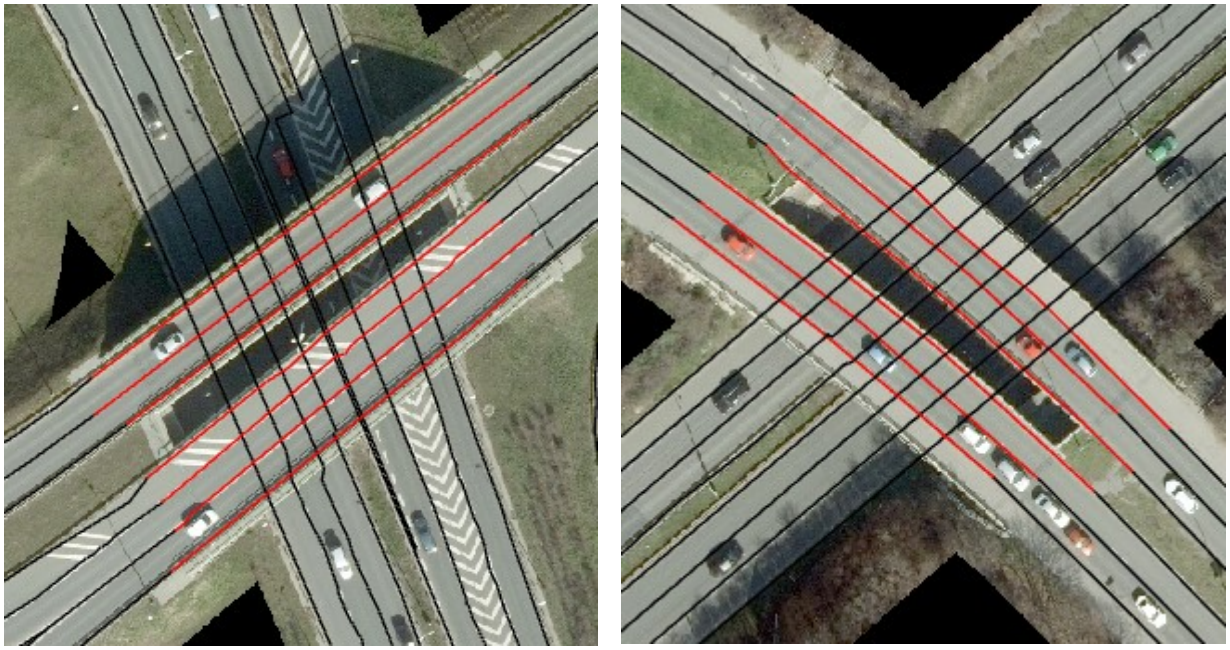

Figure 67: Motorway crossings with two surface levels (surface level in black and bridge level in red)

## 6.9. Identifying broken lines

To be able to distinguish between solid and broken lines, two specialised algorithms were created. One of them processes small RSM polygons which were too small for the ‘normal’ procedure (explained in Step 2 of the Section 5.3.1.) which is described in 5.4. and assigns the information about broken line to all created lines automatically. The other one is presented in the Section 5.5. and based on length criterion selects only those RSM line segments that match the defined length intervals.

The only limitations of the first algorithm (5.4. Small RSM Polygons) are classification errors. The algorithm incorporates the same short gap bridging process as the algorithm designed for ‘normal’ RSM polygons (5.3.). Therefore, it is able to bridge gaps shorter than 3 meters (this parameter can be set manually before running). However, it is not able to correct those gaps which are on the border of a broken line and thus have no neighbour on the other end.

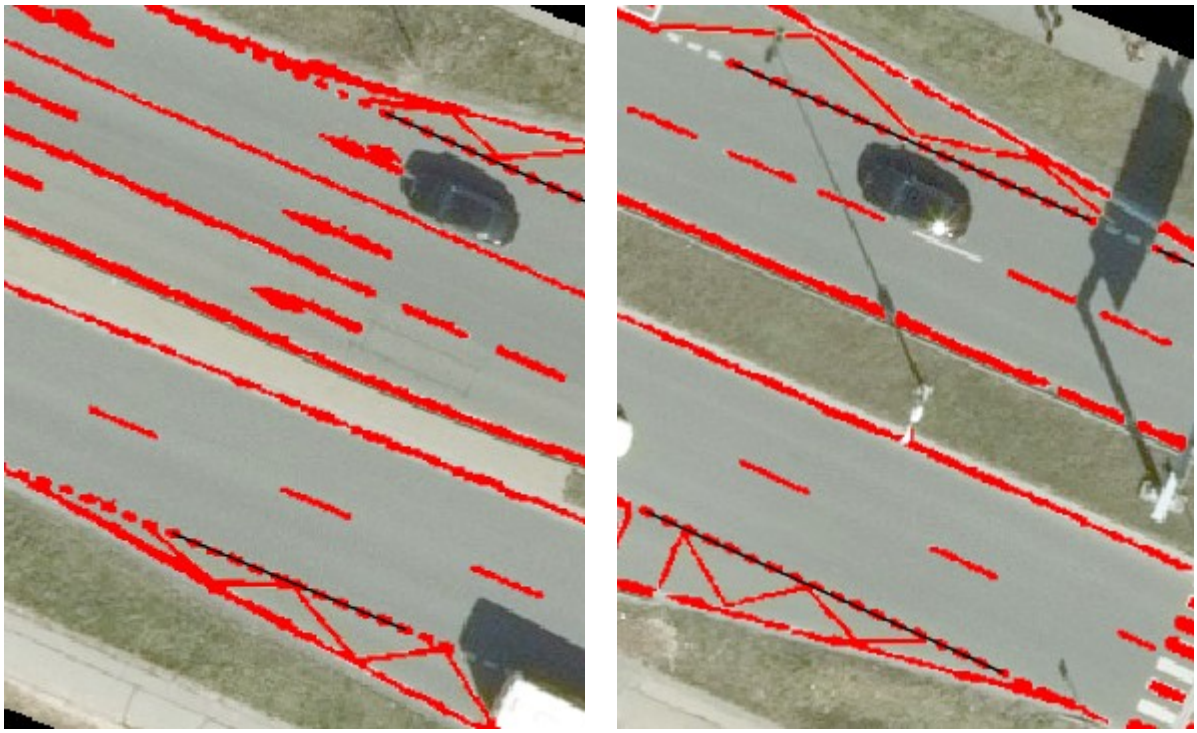

Figure 68: Modelled broken line (black) from ‘small’ RSM polygons (red)

The second algorithm was created to identify regular broken lines. This problem is more complicated than the previous one because there are more different kinds of broken lines allowed to mark different kind of driving lanes (see Table 1 for more details). The algorithm has to be built in a way to identify successfully all of them. However, it should not identify parts of solid lines or other similarly looking line segments as broken lines.

The main decision criterion is the length of RSM lines and the gaps between them (which are regularised as well). Therefore, all line segments with appropriate length will be perceived as broken line segments at first and will be connected. However, such connections which are too long are deleted in the following step and finally, too short standalone (isolated) lines are deleted as well. This should assure correctness of the algorithm in most of the cases.

However, it was found out that (especially for small case study areas which are only around 50 meters long) this might be a problematic solution because they only contain less than ten RSM lines. If it happens, that two of them are shaded or damaged, the whole case study area is not analysed properly. The same problem was discovered if two consecutive broken lines are shaded or damaged (Figure 69).

It is worth stating that the algorithm perceives a broken line as a connection from a first RSM segment to the last one; therefore the gap between the last segment of the broken line and the solid line is not identified as part of a broken line. Similarly, at the border of a case study area, a broken line only starts with a start of a first RSM line (Figure 70).

Two shortcomings of the proposed algorithm were identified which are further discussed in this chapter.


Figure 69: Modelled broken line (black) which is not continuous because two consecutive RSM lines are shaded (1) and covered by a patch (2)

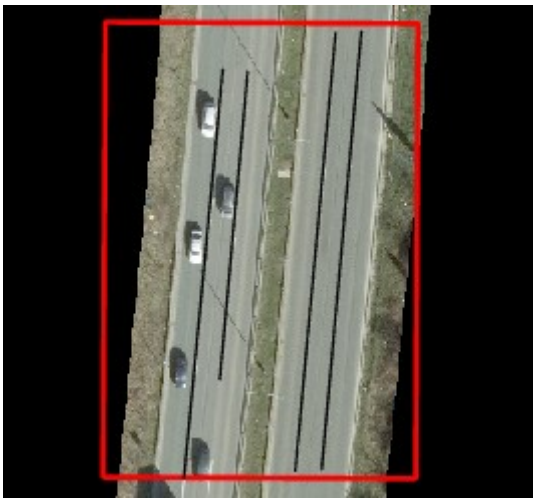

Figure 70: Modelled broken line (black) in a 60 m long case study area (red borders) starting from the first RSM line which is drawn on the road (and classified)

### Diagonal hatching

One problem of the proposed method of broken line identification can occur due to diagonal hatching. In some cases its ends are identified as line dead ends (when the classification was not performed properly) and if this phenomenon appears repeatedly along a line, it results into modelling a broken line.

The problem is to be overcome with a proper classification or with manual editing of the result. It is not possible to prevent the algorithm from creating such lines in cases that the space between hatching lines has the same length as broken lines and their gaps.

Figure 71: Diagonal hatching creating an undesired broken line (red) compared to a correctly modelled broken line (black)

### RSM lines length

In some cases it was found out that the RSM lines drawn on roads do not have proper length as they are supposed to have according to Czech national regulations causing the algorithm to create incorrect results (not to identify a broken line). This is clearly a mistake of workers when painting the physical lines.

The incorrect length problem can be solved by widening intervals for identified RSM lines; however, this solution might lead to other undesired line segments being identified as RSM lines. The correct length already has a tolerance of 0.3 m (for 1.5 m long lines) respectively 0.5 m (for 3 and 6 m long lines) and it was tested that making the interval wider would lead to new undesired modelled broken line segments.

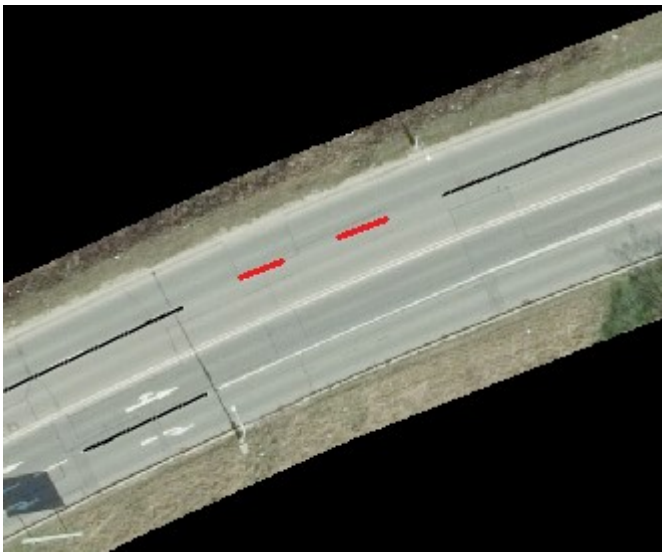

Figure 72: RSM lines which are too long to be considered broken line segments (red): 3.58 m (left) ad 3.94 m (right) instead of 3 m

## 6.10. Positional accuracy assessment

To assess positional accuracy of the result, a root mean square error (RMSE) was used. In each case study area, in average 13 points (per area) were identified. Such points were located in identifiable locations of RSM lines and driving lane borders such as end of lines or on line axes and in the aerial image. The location of points was not distributed randomly but as regularly as possible with a presumption to cover each case study area evenly.

Root mean square error is a generally used statistical measure which assesses differences between two populations. It is calculated as:

$$RMSE = \sqrt{\frac{\sum d^2}{T}}$$

where $d$ stands for a distance between measured point and its representation created by the algorithm and $T$ for number of assessed control points. RMSE values for each target area as well as combined overall RMSEs for all areas together are presented in Tables 7 (for modelled RSM lines) and 8 (for modelled driving lanes boundaries).

Two resulting datasets were assessed using RMSE. The first represents RSM lines as they were identified on road surface (result of the algorithm presented in 5.3.1.); the other represents modelled boundaries of driving lanes (combining results of 5.3.2, 5.3.3. and 5.4). Same points were identified in these datasets and their position was compared to the manually identified position in an input aerial image. Because of the manual selection of points, minor positional errors within pixel size (10 cm) are expected to appear.

**Positional accuracy of modelled RSM lines**

| Case Study Area | RMSE [m] | Case Study Area | RMSE [m] |
|---|---|---|---|
| 1 | 0.0517 | 11 | 0.0949 |
| 2 | 0.0704 | 12 | 0.0480 |
| 3 | 0.1271 | 13 | 0.1661 |
| 4 | 0.0620 | 14 | 0.1283 |
| 5 | 0.1018 | 15 | 0.0851 |
| 6 | 0.1170 | 16 | 0.0775 |
| 7 | 0.0925 | 17 | 0.1336 |
| 8 | 0.0899 | 18 | 0.1211 |
| 9 | 0.3005 | 19 | 0.1440 |
| 10 | 0.0869 | 20 | 0.1681 |
| | | **Total RMSE** | **0.1263** |

Table 7: Calculated RMSE values of modelled RSM lines

The average of around 0.11 meters (11 centimetres) and the overall RMSE of 0.126 meters (12.6 centimetres) means positionally accurate results. The only area with a high RMSE is the case study area 9 where a crash barrier was identified as a RMS line leading to positional inaccuracy. Otherwise, if we take into account the conditions (pixel size, width of RSM lines and line simplification tolerance), the total positional RMSE of 0.126 meters means that the results are satisfactory.

**Positional accuracy of modelled driving lanes boundaries**

| Case Study Area | RMSE [m] | Case Study Area | RMSE [m] |
|---|---|---|---|
| 1 | 0.1155 | 11 | 0.1627 |
| 2 | 0.1003 | 12 | 0.0746 |
| 3 | 0.1528 | 13 | 0.2115 |
| 4 | 0.0769 | 14 | 0.1254 |
| 5 | 0.1234 | 15 | 0.0903 |
| 6 | 0.1472 | 16 | 0.0969 |
| 7 | 0.1212 | 17 | 0.1383 |
| 8 | 0.1005 | 18 | 0.1311 |
| 9 | 0.3148 | 19 | 0.2065 |
| 10 | 0.1484 | 20 | 0.1427 |
| | | **Total RMSE** | **0.1486** |

Table 8: Calculated RMSE values of modelled driving lanes boundaries

It can be seen that with the average RMSE of around 0.14 meters (14 centimetres) the algorithm has sufficient positional accuracy, taking into account the pixel size (10 centimetres) and the line simplification tolerance of 30 centimetres. Exceptionally high is the value in the case study area 9 (where a crash barrier was modelled as RSM; see page 67 for more details) 13 (where a railing on a pavement was identified as broken line; see page 69 for details) and 19 (where the marking was fading and thus not classified properly). The overall RSME of 0.149 meters is a sufficient result after considering accompanying conditions.

# 7. Discussion

A semi-automated method of identifying driving lanes in aerial images has been developed and presented in this thesis. Two different data sources are needed to successfully perform the method and obtain relevant results. First input dataset is high resolution aerial imagery in which road surface marking is identifiable. The other information source are polylines depicting road network which help in minimising performance costs as well as serve as positional information input in places where the original line is not visible in an aerial image (such localities are called 'gaps' in this paper).

Firstly, input aerial images are split into segments and based on the segmentation a classification according to predefined criteria is performed (object-based image analysis). The aim of such classification is to identify polygons which represent road surface marking (RSM) marked on the surface of a road. Such data serves as the main input for proposed algorithms. The algorithms represent a key contribution of this thesis as they identify boundaries of driving lanes and create a representation of a road with the scale of single

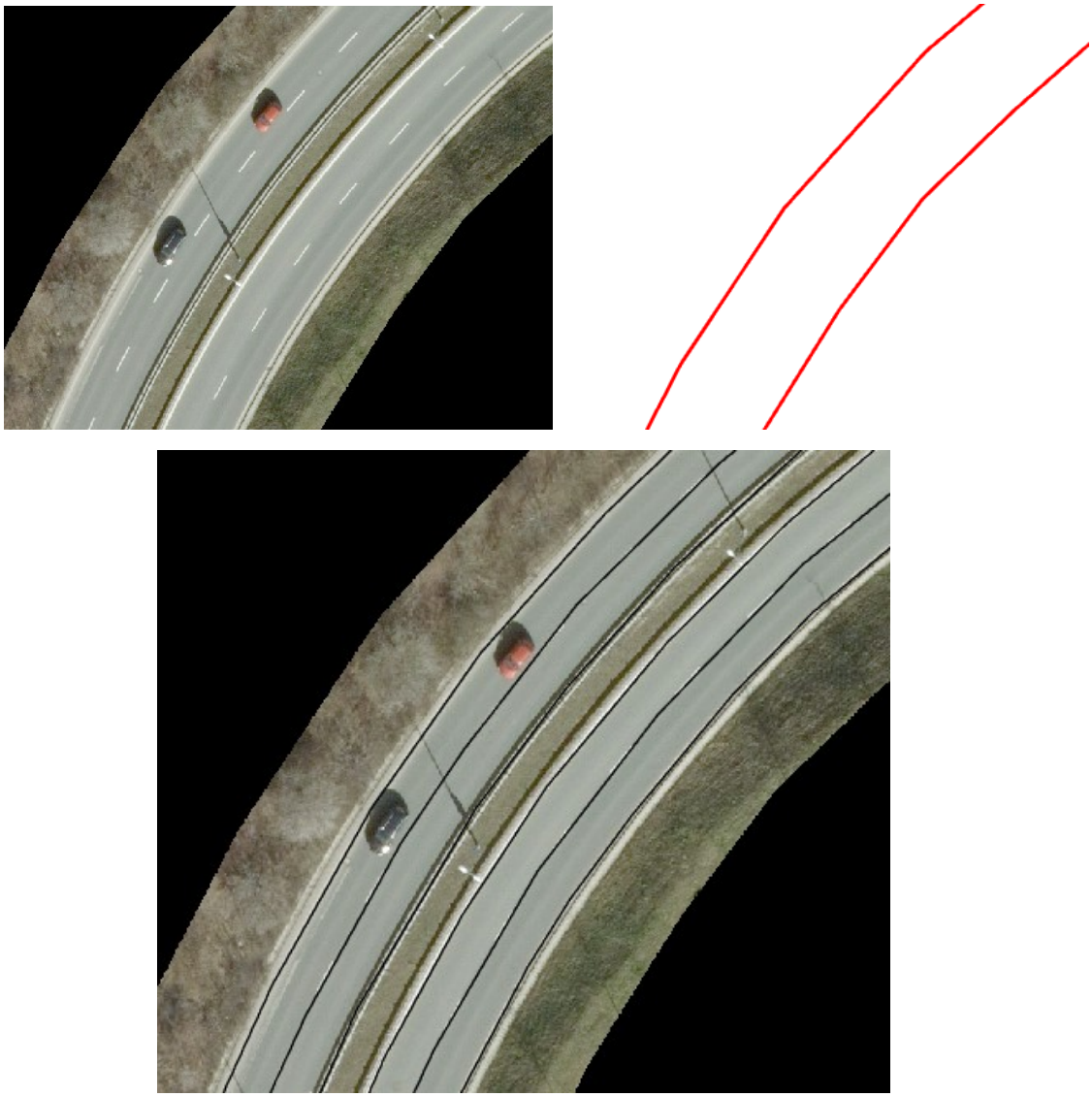

Figure 73: Input data and result: aerial image (top left) and polylines representing road infrastructure (top right) as input data and polylines representing driving lanes (bottom, in black) as result

driving lanes.

The method consists of multiple algorithms out of which each is specialised in one step of the process. First algorithm creates polylines representing RSM lines out of classified polygons. Second algorithm interconnects such polylines and models borders of driving lanes. Next algorithm is able to create vector lines in places where no RSM was classified (shadows, under bridges, misclassification etc.). A special algorithm was necessary to analyse very short (shorter than 0.5 meters) broken lines as these caused the previously mentioned algorithms to fail. The purpose of the last created algorithm is to improve information volume contained in the resulting dataset; it aims to identify broken lines and save the information in attributes of the respective lines.

Proposed algorithms were tested in a case study area of the suburban part of Prague, Czechia with various different types of road infrastructure (motorway, main roads with multiple driving lanes, bus bays, over- and underpasses, changing amount of driving lanes etc.). This served both for finding optimal numerical parameters for operations of the algorithms as well as for testing effectivity and correctness of the algorithms themselves. The results as well as eventual shortcomings of the algorithms (or of input data) are presented in the previous chapter.

Positional accuracy of modelled RSM lines as well as of modelled driving lanes boundaries was examined and calculated using RMSE. With the overall RMSE of 0.126 metres for RSM lines and 0.149 meters for modelled driving lanes boundaries it can be stated that the results have sufficient positional accuracy; especially when taking into account the pixel size of input images, width of RSM lines drawn on road surface as well as the line simplification tolerance.

Therefore it can be stated that the objectives of the thesis were achieved. Algorithms for creating vector data representing roads on driving lanes scale were developed. Testing on a case study area proved their effectiveness; several shortcomings, however, are leading to possibilities for future improvement of the method which are discussed in the following paragraphs.

First, more testing in a bigger case study area is necessary for a more thorough evaluation of the method. A lot of shortcomings presented in this thesis were localised close to borders of a case study area and might be caused by missing context of the surroundings (missing neighbourhood connections). From the already identified disadvantages of the method diagonal hatching, undesired line oscillation and dependency on input road vector data can be mentioned.

Diagonal hatching causes several problems of the algorithms. It can be identified as a broken line when the hatching distance equals broken line length and it may cause driving lane borders to oscillate. The problem might occur due to the fact that diagonal hatching lines are analysed as separate lines instead of parts of one bigger object. Therefore a possible limitation connected to number of lines coming from each vertex might serve as an applicable identification of diagonal hatching lines and thus limiting them from the problematic calculation procedures.

Multiple reasons can cause a resulting line to oscillate undesirably; therefore a line simplification is incorporated into the process. However, not always does the simplification bring correct results, this may be caused by the incorrect classification of aerial images

(leading to the fact that either objects which are not RSM lines are classified as RSM or parts of RSM are not classified). Solving this on the geometry level would possibly require working with much larger surroundings that neighbourhood to be able to identify undesired sudden oscillations in an otherwise straight line.

Similar solution might solve the problem with missing continuity around gap bridging lines, which appears in places where the input road dataset does not exactly copy the direction of a driving lane. In this case as well it might be beneficial to use a larger area to identify a directional trend of RSM lines rather than their immediate vector towards the road depicting polyline.

Finally, the broken line detection might be improved to provide correct results in areas close to case study area borders. Possible solution here would be not to analyse length of lines only but their neighbourhood as well. If a line segment was not identified as a broken line directly based on length criterion but it is relatively short and both its neighbours are broken lines, then it would most probably be part of a broken line as well. On the other hand, if a standalone segment with no other segments in its neighbourhood (i.e. its neighbours are both solid lines) is identified according to its length as broken line, it would most probably be a solid line segment too. Similar approach could possibly deal with line segments on dead ends (i.e. with only one neighbour) as well.

To conclude, this thesis should be perceived as a pioneer work in vector analysis of road surface marking lines. According to its author, the area itself is promising for obtaining such data from aerial images rather than from driver's perspective cameras (high costs for large areas) or GPS traces (inaccurate data). However, image classification as well as the visibility of surface marking itself in aerial imagery would have to be improved as a prerequisite for a fully successful analysis of this kind.

# References

## Scientific publications


- Aeberhard, M., Rauch, S., Bahram, M., Tanzmeister, G., Thomas, J., Pilat, Y., Homm, F., Huber, W., Kaempchen, N. (2015): Experience, Results and Lessons Learned from Automated Driving on Germany's Highways. IEEE Intelligent transportation systems magazine, volume 7, issue 1, pages 42–57
- Aichholzer, O., Aurenhammer, F., Alberts, D., Gärtner, D. (1995): A Novel Type of Skeleton for Polygons. Journal of Universal Computer Science, volume 1, issue 12, pages 752–761
- Bakhtiari, H. R. R., Abdollahi, A., Rezaeian, H. (2017): Semi-automatic road extraction from digital images. The Egyptian Journal of Remote Sensing and Space Sciences, volume 20, pages 117–123
- Baumgartner, A., Hinz, S. (2003): Automatic extraction of urban road networks from multi-view aerial imagery. ISPRS Journal of Photogrammetry & Remote Sensing, volume 58, pages 83–98
- Blaschke, T. (2009): Object based image analysis for remote sensing. ISPRS Journal of Photogrammetry and Remote Sensing, volume 65, issue 1, pages 2–16
- Blum, H. F. (1967): A transformation for extracting new descriptors of shape. In: Proceedings of Symposium on Models for Perception of Speech and Visual Form. Cambridge, Mass., M.I.T. Press (editor: Weiant, W., D.), pages 362–380
- Cheng, G., Zhu, F., Xiang, S., Wang, Y., Pan, Ch. (2016): Accurate urban road centerline extraction from VHR imagery via multiscale segmentation and tensor voting. Neurocomputing, volume 205, pages 407–420
- Český úřad zeměměřický a katastrální (2018): Katalog objektů ZABAGED®. ČÚZK, Praha 2018, verze 3.0 ve znění dodatku č.1a č. 2. [online] https://geoportal.cuzk.cz/Dokumenty/KATALOG_OBJEKTU_ZABAGED_2018.pdf
- Douglas, D. H., Peucker, T. K. (1973): Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. The Canadian Cartographer, volume 10, issue 2, pages 112–122
- Fischer, P., Azimi, S. M., Roschlaub, R., Krauß, T. (2018): Towards HD Maps from Aerial Imagery: Robust Lane Marking Segmentation Using Country-Scale Imagery. ISPRS International Journal of Geo-Information, volume 7, page 458.
- Gontran, H., Skaloud, J., and Janvier, N. (2006): Open-source software operated CMOS camera for real-time mapping. In the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science, Paris, France.
- Hormese, J., Saravanan, C. (2016): Automated Road Extraction From High Resolution Satellite Images. International Conference on Emerging Trends in Engineering, Science and Technology (ICETETS) 2015. In: Procedia Technology, volume 24, pages 1460–1467
- Jin, H., Feng, Y., Li, Z. (2009): Extraction of road lanes from high-resolution stereo aerial imagery based on maximum likelihood segmentation and texture enhancement.

Digital Image Computing: Techniques and Applications, Melbourne, Australia, December 2009.

- Jin, H., Feng, Y. (2010): Automated road pavement marking detection from high resolution aerial images based on multi-resolution image analysis and anisotropic Gaussian filtering. Proceedings of the 2nd International Conference on Signal Processing Systems (ICSPS), Dalian, China. Volume 1, pages 337–341.
- Jin, H., Miska, M., Chung, E., Li, M., Feng, Y. (2012): Road Feature Extraction from High Resolution Aerial Images Upon Rural Regions Based on Multi-Resolution Image Analysis and Gabor Filters. Remote Sensing – Advanced Techniques and Platforms (editor Dr. Boris Escalante). InTech, Shanghai, China, pages 387–414
- Jeong, P., Nedevschi, S. (2005): Efficient and robust classification method using combined feature vector for lane detection. IEEE Transactions on Circuits and Systems for Video Technology, volume 15, issue 4, pages 528–537
- Kettig, R. L. , Landgrebe, D. A. (1976): Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects. IEEE Transactions on Geoscience Electronics, volume 14, issue 1, pages 19–26
- Kreucher, C., Lakshmanan, S. (1999): LANA: a lane extraction algorithm that uses frequency domain features. IEEE Transactions on Robotics and Automation, volume 15, issue 2, pages 343–350
- Lee, S., Kim, J., Yoon, J. S., Shin, S., Bailo, O., Kim, N., Lee, T.–H., Hong, H. S., Han, S.–H., Kweon, I. S. (2017): VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition. 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, pages 1965–1973
- Leninisha, S., Vani, K. (2015): Water flow based geometric active deformable model for road network. ISPRS Journal of Photogrammetry and Remote Sensing, volume 102, pages 140–147
- Máttyus, G., Luo, W., Urtasun, R. (2017): DeepRoadMapper: Extracting Road Topology from Aerial Images. 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, page 3458–3466
- Seo, W.-Y. (2012): Augmenting Cartographic Resources and Assessing Roadway State for Vehicle Navigation. PhD Thesis, April 2012, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA.
- Shahi, K., Shafri, H. Z. M., Taherzadeh, E., Mansor, S., Muniandy, R. (2015): A novel spectral index to automatically extract road networks from WorldView-2 satellite imagery. The Egyptian Journal of Remote Sensing and Space Sciences, volume 18, pages 27–33
- Ministry of Traffic of the Czech Republic, Department of Roadways: TP 133 Zásady pro vodorovné dopravní značení na pozemních komunikacích. Technické podmínky, editor Ing A. Seidl, 2013. [online] www.pjpk.cz/data/USR_001_2_8_TP/TP_133.pdf
- Wang, W., Yang, N., Zhang, Y., Wang, F. (2016): A review of road extraction from remote sensing images. Journal of Traffic and Transportation Engineering (English Edition), volume 3, issue 3, pages 271–282
- Xia, W., Zhang, Y., Liu, J., Luo, L., Yang, K. (2018): Road Extraction from High Resolution Image with Deep Convolution Network – A Case Study of GF-2 Image. 2nd

International Electronic Conference on Remote Sensing, Proceedings, volume 2, article number 325
- Yuan, J., Wang, D., Wu, B., Yan, L., Li, R. (2009): Automatic Road Extraction from Satellite Imagery Using LEGION Networks. Proceedings of International Joint Conference on Neural Networks, Atlanta, Georgia, USA. Pages 3471–3476
- Zhang, Z., Zhang, X Sun, Y., Zhang, P. (2018): Road Centreline Extraction from Very-High-Resolution Aerial Image and LiDAR Data Based on Road Connectivity. Remote Sensing, volume 10, issue 8, article number 1284

## Other sources

- ESRI (2020): ArcGIS Pro Python reference. Redlands, California. Cited 02/04/2020. https://pro.arcgis.com/en/pro-app/arcpy/main/arcgis-pro-arcpy-reference.htm
- Holmes, O. (2019): High Resolution Digital Aerial Imagery vs High Resolution Satellite Imagery – Part 1. Glynde, Australia. Posted 20/01/2012, cited 02/03/2020. https://aerometrex.com.au/technical/high-resolution-digital-aerial-imagery-vs-high-resolution-satellite-imagery-part-1/
- Ministry of Traffic of the Czech Republic, Road and Motorway Directorate of the Czech Republic: Politika jakosti pozemních komunikací. Prague, Czechia. Last update 22/05/2018, cited 17/01/2020. http://www.pjpk.cz/technicke-podminky-tp/

## Used data

- Aerial images: Prague Institute of Planning and Development (IPR Praha), licensed under CC BY-SA 4.0 [http://creativecommons.org/licenses/by-sa/4.0/], provided on 24/01/2020.
- Vector road dataset: OpenStreet Map contributors (OSM), licensed under CC-BY-SA 2.0 [https://creativecommons.org/licenses/by-sa/2.0/], downloaded on 18/02/2020

# List of appendices

This chapter represents a list of appendices which are attached to the thesis with a brief explanation of their meaning and of how to work with them.

## Created codes

Original codes created by the author to perform the analysis of RSM as presented in the thesis. All of them are written in Python 3 and use arcpy module with an advanced license (including Spatial Extension). Input data as well as a path to a database for temporal files (Default database for example) and to a resulting dataset must be correctly set in the code prior to running it. The order of running the codes to comply with the process is identified by the number in the name of respective codes.

Following codes are appended:

**1_PolygonToMarkingLine.py**

*Code to create polylines representing RSM lines on road surface*
Input data: polygon file with classification result (road marking polygons)
Output data: polyline file representing road surface marking

**2_MarkingLinesToRSM.py**

*Code to create polylines representing (boundaries of) driving lanes*
Input data:

- polygon file representing road surface marking (result of *1_PolygonToMarkingLine.py*)
- polyline file representing roads

Output data: polyline file representing boundaries of driving lanes

**3_IdentifyGapVertices.py**

*Code to create points representing vertices of gaps in boundaries of driving lanes*
Input data:

- polygon file with borders of case study area
- polyline file with lines representing boundaries of driving lanes (result of *2_MarkingLinesToRSM.py*)

Output data: point file representing vertices of gaps in boundaries of driving lanes

**4_IdentifyRoadPoints.py**

*Code to create and order points representing roads (vertices of road polylines)*
Input data:

- point file representing vertices of gaps in boundaries of driving lanes (result of *3_IdentifyGapVertices.py*)
- polygon file with borders of case study area
- polyline file representing roads

Output data: point file representing roads

**5_EditLongGaps.py**

*Code to bridge gaps in boundaries of driving lanes*
Input data:

- point file representing vertices of gaps in boundaries of driving lanes (result of *3_IdentifyGapVertices.py*)
- point file representing roads (result of *4_IdentifyRoadPoints.py*)

Output data: polyline file representing bridged gaps in boundaries of driving lanes

**6_ShortMarkingLineSegments.py**

*Code to create polylines representing short RSM line segments (< 1 meter)*
Input data: polygon file with classification result (road marking polygons)
Output data: polyline file representing broken lines with short line segments (< 1 meter)

**7_IdentifyBrokenLine.py**

*Code to identify broken lines in boundaries of riving lanes*
Input data:

- polygon file representing road surface marking (result of *1_PolygonToMarkingLine.py*)
- polyline file with lines representing boundaries of driving lanes (result of *2_MarkingLinesToRSM.py*)

Output data: polyline file representing broken lines of RSM

## Resulting data

Two datasets are appended to the thesis as ESRI shapefiles. These are:

- **1_RSM**

  Polyline shapefile representing identified RSM lines (result of *1_PolygonToMarkingLine.py*) with no relevant attributes.

- **2_DrivingLanes**

  Polyline shapefile representing boundaries of driving lanes (combined result of *2_MarkingLinesToRSM.py* merged with gaps modelled by 5*_EditLongGaps.py* and short line segments created by *6_ShortMarkingLineSegments.py*). Attribute *TypeLine* distinguishes between solid and broken lines can have 4 options – "broken" (identified by *7_IdentifyBrokenLine.py* or created by *6_ShortMarkingLineSegments.py*); "solid" (lines not identified as broken lines by *7_IdentifyBrokenLine.py*); "unknown" (lines representing long gaps which are not seen in the aerial image and thus are unknown) and " " (empty information for short line segments located mostly at the end of case study areas which were too short to be analysed by *7_IdentifyBrokenLine.py*).