Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Master's Thesis

Automatic generation of route description for visually impaired users

Bc. Jakub Bokšanský

Supervisor: Ing. Zdeněk Míkovec, Ph.D.

Study Programme: Open Informatics

Field of Study: Computer Graphics and Interaction

May 6, 2013

Aknowledgements

I would hereby like to thank my supervisor, Ing. Zdeněk Míkovec, Ph.D. for motivation, assistance and introduction to this research project. I also wish to thank my opponent, Ing. Jan Vystrčil for providing literature. This thesis would not have been possible without people from SONS organization involved in testing the prototype of our application and I would like to thank them for priceless advise and support of this work.

Declaration

Abstract

This work is dedicated to designing a system for generating route descriptions for visually impaired people. Complete system for creating such descriptions is designed and implemented. Work starts by analyzing mobility and orientation capabilities of visually impaired people. Then it discusses, how can be a suitable source of map data used to create descriptions of urban areas for them. Complete algorithm for automatic generation of such descriptions is proposed. Solution includes schema for retrieving data from GIS data source, extracting essential information and creating naturally sounding route description with use of context of individual map elements. This algorithm supports customizing generated description according to end-user's preferences which makes it very flexible and easily modifiable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ability to move independently and freely is a certainty for most of us. Just knowing that we can get from one place to another in case we want or need guarantees us a certain quality of life. This is not necessarily true for visually impaired people. Their ability to move depends on severity of their visual impairment. They might be able to move almost normally, only with help or not at all. In the worst case they are immobilized until they get required training and adequate help. Even after that, their independence is usually limited to familiar places they are used to. This usually includes their home, close surroundings and organizations which provide services for visually impaired. In these places they can rely on help of their family, friends and on some places on dedicated workers or specialists.

While moving between these places through the exterior they are usually on their own. People on the street are not likely to give them proper help they might need and even less likely to guide them to destination. Furthermore, ability to manage it by self is priceless. It gives people confidence and feeling of self-sufficiency - inseparable part of full-valued life. Aim of this work is to help visually impaired people to be more independent of others by providing them suitable route descriptions fast and reliably.

In this work we will look at orientation and navigation of visually impaired people in exterior, more precisely in urban areas. We will cover various tactile markings which are part of modern cities and are dedicated to help visually impaired to orientate. We will also examine how visually impaired person orientates himself in the urban environment, what tools and aids he uses and what training he needs. By studying this, we will better understand how should ideal route description look and what should it contain. Our goal is to design and implement system for creating such descriptions programmatically. As a basis we take prototype application created in works [45] and [29] and continue development by improving it based on user tests. This system will be used by an operator who now creates description manually and is intended to load off most of work from him by creating these descriptions automatically.

1.1 Problem Statement

Simply said, problem is to make computer do work which is currently done by people. Let's look at what this work is and by whom it is done. These people are operators of navigation center [11] of SONS [26] (Czech Blind United). They have various tasks and they provide many services to their blind users. One of them is creating description of route in the city. E.g. a user needs to get from his home to hospital so he calls navigation center and asks them to prepare a description of this route for him. Process of creating route description is currently as follows:

1. Visually impaired person requests a description of given route
2. Operator tries to find existing description of such route stored on his computer. They use .doc format to store these descriptions. Descriptions are not organized; they store them simply on file system in a single directory.
3. If not found, he uses Google maps, Google street view, Seznam maps (Czech map server) to create such description. He uses satellite images and his knowledge of city to plan the route and to describe it. For measuring distances he uses Seznam maps which support easy distances measuring by dragging mouse between two points.
4. Created description is stored on the disk in .doc format
5. And it is sent to user by an email.
6. User reads the description using screen reader (usually on his smartphone) and navigates himself independently.
7. If needed, user calls to navigation center to query more information about the route or clear any misunderstandings.

Our system should be used in steps 2 and 3 to do as much work as it can to plan the route and create description. With help of this system operators could serve more users quickly and with much less effort and same or better quality of service. Currently, they need 3 days to create requested description (in emergency, they can provide it on next day). By using our system, this time could be cut down to hours or even minutes. Generated description could be ideally used by end user directly, but we expect operator to review it and edit if needed. Some changes might be needed because of missing information that is unavailable in server's database or when user prefers to have additional information in description. In such case, operator can use our system to generate „skeleton“ of description and adjust it accordingly.

Whether operator makes substantial changes to description or not, our system is always helpful by automatically planning optimal route, so it off loads at least this step from operator. It should be noted, that when we mention *system users* in this work, we mean operators of navigation center, who are not visually impaired. They provide route description created by them or our system to their end users who are visually impaired. They should not be interchanged.

Problem of designing such system can be split into three sub problems:

1. Finding and utilizing source of data and implementing API to access them
2. Creating system for planning routes
3. Developing system for generating descriptions

First two problems were solved in works [45] and [29]. [45] was first to implement all three problems partially, but focused mainly on finding and utilizing suitable data source and creating platform for using it. Result was a functional prototype. Work [29] solved problem 2 – planning the route. It implemented system for finding best route from point of departure to destination according to given criteria and user preferences.

In this work we will focus on third problem – generating description of route. This problem touches various scientific disciplines from computer science to urban architecture. We will have to utilize data mining techniques, database systems, software architecture, geographic calculations, natural language generation, cognitive science disability studies and also design new algorithms to solve arising problems.

1.2 Orientation of visually impaired people in urban areas

A blind can be introduced to new unfamiliar urban area by several ways, which lead to different experience:

- Verbal description
- Tactile Map
- Direct experience

Assuming that a blind person wants to learn a new route, or get a spatial knowledge of urban area in his surrounding, paper [34] suggests that combination of at least two of listed is preferable. Furthermore, direct experience is a must for learning effectively.

To successfully introduce a blind to new environment, he must be able to move independently, confidently and without danger. Another requirement is ability to move quickly and effectively. Ability to move autonomously is essential for visually impaired people and their integration in society.

Generally, visually impaired people are able to acquire and store spatial relations just as a sighted people, however more effort and time is needed. It is desired to create strategies and utilities to help them gain this knowledge more quickly. Paper [34] analysed efficiency of different instructional methods by conducting two experiments in the city of Madrid (Spain) and Sheffield (Britain). Participant with access to tactile maps in combination with guide or spoken description of area showed great improvement in spatial behavior than participants with only one source of information. Also if only one source of information was available, tactile map alone showed better results than other sources.

Receiving route direction triggers complicated task of transforming verbal information into spatial layout and memorizing it. Spatial perception evolves over time and training. Ability to orientate in urban areas therefore improves with gained experience. This applies to sighted men and also visually impaired people.

Mental image of route is created from route description and previous experience. Quality of this image (whether it is helpful for locating desired destination) is dependent on determination of description i.e. how much information it contains. According to [48], overdetermination is in general worse than underdetermination. This is important while making decision, whether include some piece of information or not. In general it is better not to. When description is overdetermined, user tends to rely on it more and cannot solve unexpected situations flexibly. User should employ more of his own experience and thinking through whole route to react on various situations.

Giving directions in urban areas can be done from two points of views – either intrinsic or extrinsic frame of reference. Intrinsic is given with regards to location and direction of user (e.g. „to your left, ahead of you …“) and extrinsic is given from bird's perspective (usually formulated using compass directions - north, east, south and west). Paper [42] conducted a study of several groups of different people to examine preference of one frame of reference to other. Since extrinsic frame of reference is not practically applicable in navigation of visually impaired, we will work with intrinsic frame of reference.

1.2.1 Tactile Maps

Tactile map is extremely simplified image of mapped area (see figure 1.1). Its purpose is to provide visually impaired people option to learn the map by their haptic sense. Tactile maps are also used to teach geography visually impaired children. They are also sometimes used to describe other types of data such as paintings, drawings, graphs and others. These maps are also used by sighted people, however, their haptic print serves mainly for esthetical purpose.

They are usually made of plastic by a process called vacuum forming [25]. Sheet of plastic is heated and pressed onto form, which gives it desired shape. This process is simple and created maps are relatively cheap. This solves just a problem of making physical copies of the map, but before that a designer must create a model. Traditionally these models were made by a specialist. There was not standard way of creating such map for a long time, nowadays these designers should follow guidelines and standards for tactile graphics such as [5].

Very interesting project is TMAP (Tactile Map Automated Production) [24] which is dedicated to create software able to generate tactile map of arbitrary location in United States. Scope of the project is limited to street maps and area of United States. It uses GIS database to acquire data, generates map of desired location and user can print finished map on his Braille embosser machine.

Another interesting project is a combination of tactile map with audio description [36]. User is presented a tactile map with accompanying verbal description by software. According to results, these audio-tactile maps were seen as simple, understandable and fun to use.

1.2.2 Tactile Paving

Tactile paving is a feature of modern urban architecture intended to assist visually people move around the city. These are usually found on avenues, frequented streets, public transportation stations, bus and tram stops, in the vicinity of office buildings and shopping centers etc. Interior variant of tactile paving is also available on many public places. It is also a target of urban architecture research [38]. How is this helpful to visually impaired? Main idea is to keep him on correct course and stop him at the end or at a crossroad (see figure 1.3.

To accomplish this, the tactile paving has character of guide lines and stop marks. Visually impaired can follow guide line until he reaches stop mark (zebra crossing, crossroad,

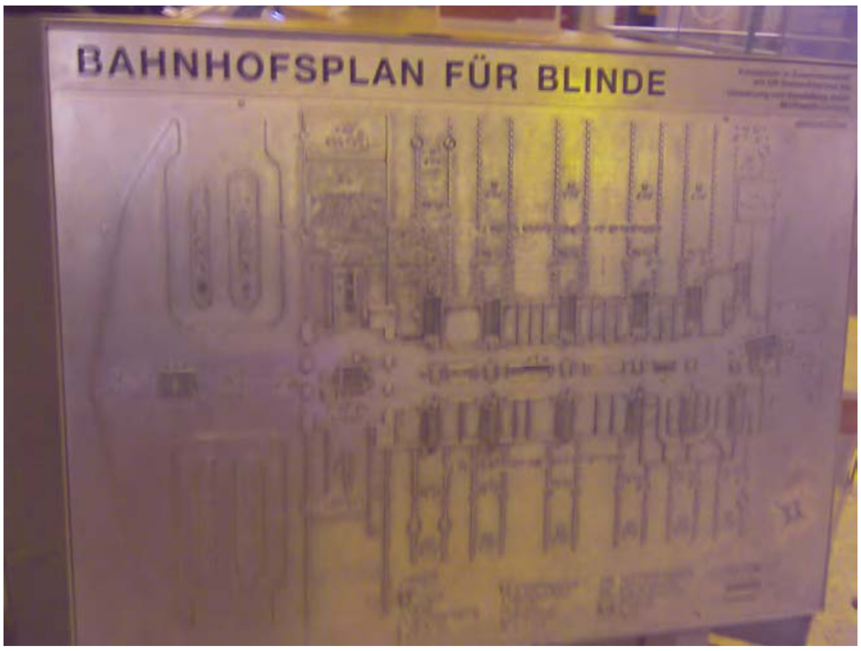

Figure 1.1: Tactile map on main railway station in Hannover, Germany. Image taken from OpenStreetMap [13]

etc.), see figure 1.2. There he can decide which way to continue – to do so, he must know the route, or ask for directions, or use some navigation system commonly found inside buildings. Zebra crossing and places of interest are identified by change of guide line to different type of paving and usually also by change of color (yellow, red or white markings). Since many visually impaired have some (although very limited) residual vision – they are called legally blind or people with low vision – the color highlight is also used.

Another purpose of tactile paving is to warn them before dangerous places. Typical example of such usage is tactile paving on train stations near railroad tracks (see figure 1.4). Tactile paving of bold color and distinctive paving texture is used to avoid visually impaired to fall onto tracks.

The problem with tactile paving is, how is visually impaired going to feel the guide line. This is simply solved by using white cane. By tapping around he can find and follow the guide line safely. He also uses his feet to feel the type of paving (some types have special meaning). In rare cases he can use his hand to follow wall of a building or hand-rail, but this is not considered very safe in exterior. Most troubles while using tactile paving are caused by obstacles placed on guide lines by people unaware of their purpose (often menu boards placed in front of restaurants) and by dust accumulated in notches of guidelines. Once filled with dust, they form surface resembling regular tarmac surface and cannot be found by white cane. Unfortunately, tactile paving is invisible under snow, so movement during winter months is even more complicated.

First tactile paving was used in late 1960's in Japan. Its designer was Seiichi Miyake [22] and it spread quickly around Japan and rest of the world. Since there was no standard

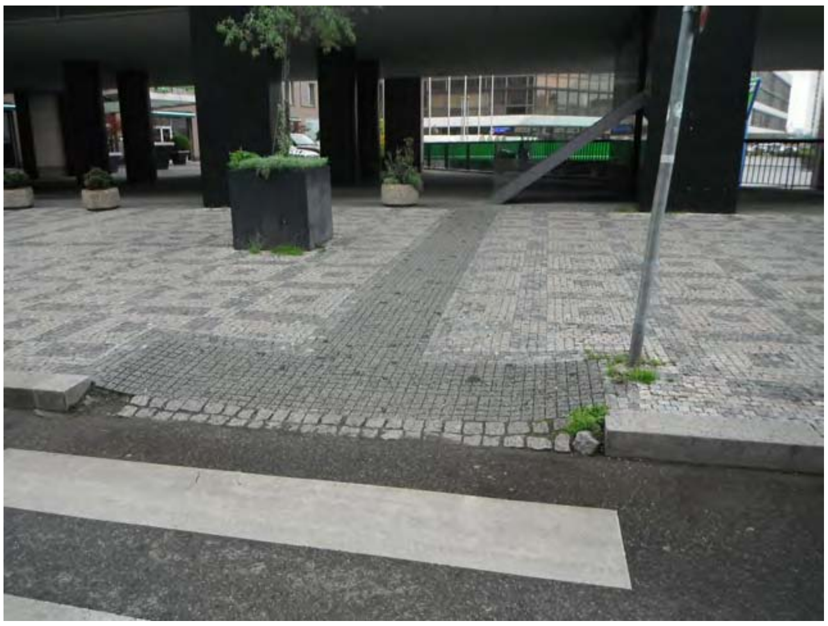

Figure 1.2: Tactile paving on uncontrolled crossing.

way of building tactile paving, today we can find tactile paving in different shapes and forms in cities around the world. Nowadays, some standards exist that should be followed while building new tactile paving. These standards are sometimes part of more broad standards enclosing barrier-free access in general. These standards differ in various countries but are mostly similar in fundamental parts. Most important principles for building useful tactile paving is to keep it simple, consistent over the city and logical [3]. Study [40] compared these standards in USA, United Kingdom, Japan and China. According to this study, these types of tactile paving are commonly found:

- Blister surface
- Corduroy hazard warning surface
- Platform warning surface
- Delineator strip
- Guidance paths

A textures commonly used to create these surfaces are raised dots and stripes. Their sizes and proportions vary according to different standards. Raised dots surface creates blister surface and warning surfaces, stripes create delineator strips and guidance paths. According to this, these textures are basic building blocks of most tactile paving found on the streets. Although other shapes than dots were tried (zigzags, ovals, domes . . .), these are not easily distinguishable with white cane neither with foot and their use abolished [40]. There is tendency (strong in Japan) to use only 2 basic types of textured surfaces in the future.

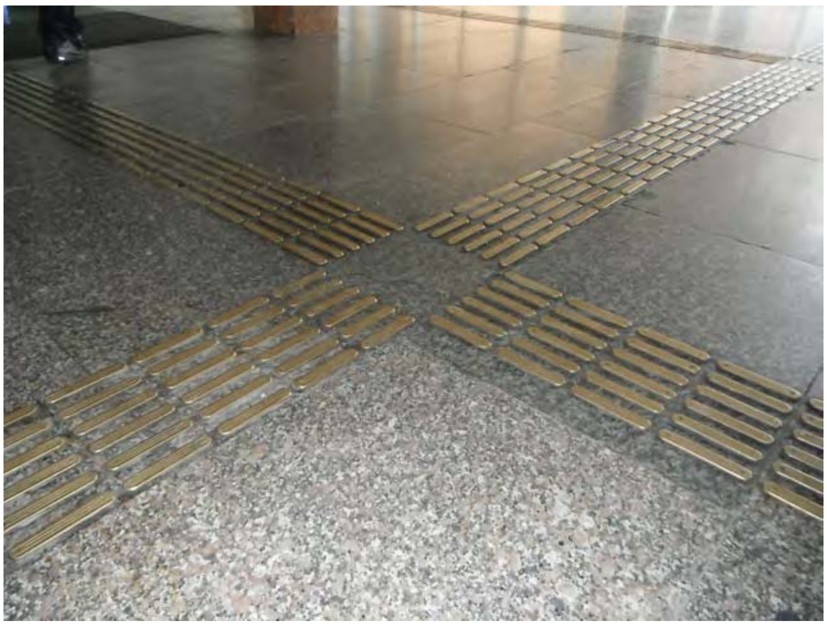

Figure 1.3: Tactile guidelines crossing.

Colors are used very differently across countries. In Japan, yellow is common color for tactile paving because of its brightness and because it stands out also at night. Red is commonly used to signal controlled zebra crossing in European cities and some other contrast color (e.g. white) to signal uncontrolled crossings. Combination of yellow and red is used in many cities as a warning on platforms (on railway stations), but in the UK use of red is completely prohibited in this situation. Basically, colors have to be contrasting with surrounding so tactile features are highlighted. Sometimes color of these features is adjusted to esthetically snap into architectural style of given area (often seen near hotels).

Blister surfaces

According to [3], purpose of these surfaces is to provide a warning for visually impaired who would not be able to differentiate footway and carriage way otherwise. This corresponds to usage of blister surface in the city of Prague, where our system will be used and nearby cities. These surfaces are usually highlighted with red or white color. This is most commonly found type of tactile paving as it server mostly for warning purpose. It is used primarily on occasions where curb is absent and therefore the edge of footway cannot be identified with white cane [3]. It is created by raising dots or domes from underlying surface (see figure 1.5).

Corduroy warning surface

Corduroy warning surface (see figure 1.6) is used as a hazard warning. It is used in presence of steps, elevators, crossings, etc. both in exterior and interior but is more likely to be found in interior. It has shape of elevated rounded bars which are parallel [1] (not to be confused with guide path, which has deeper notches and is not elevated). UK guidance on

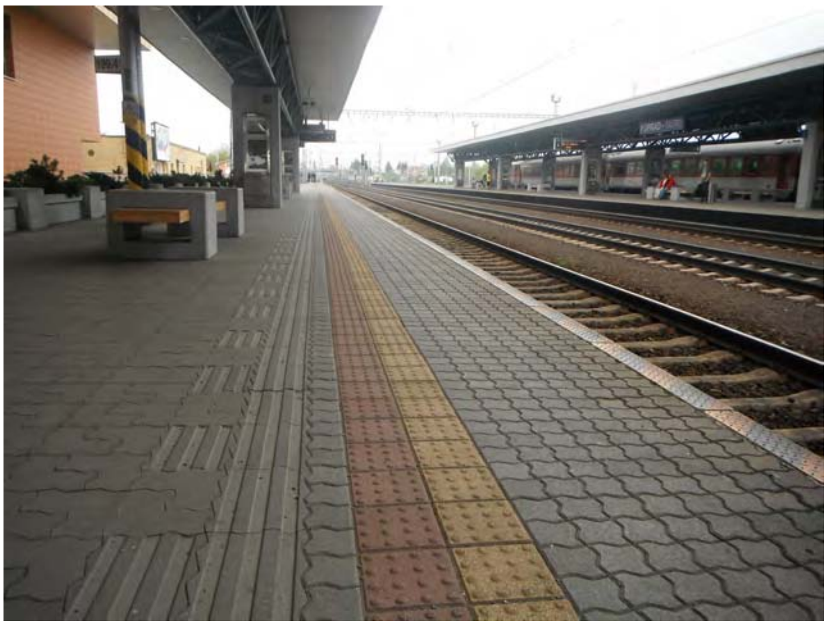

Figure 1.4: Warning surface on the edge of a platform

use of tactile surfaces [3] says that any suitable material can be used to create this type of surface. Many manufacturers offer concrete corduroy tiles colored white, yellow or red.

Platform warning surface

This is generally any surface that informs visually impaired about platform edge and prevents him from falling onto railroad track, see figure 1.4. These are very different over countries and even cities. They can be created by using blister surface (different texture can be used as on footways) or corduroy surfaces). As said earlier, color used to highlight this surface is not consistent, often combination of more colors is used.

Delineator strip

This type of tactile paving is used to separate pedestrians from cyclists, see figure 1.7. Usually it has shape of rounded curb, elevated above ground level so it creates natural barrier between footway and cycle track. This type of paving is also useful for other people than visually impaired, e.g. deaf people who cannot hear cyclist coming [3].

Guidance paths

This is one of most important tactile paving type. It is used to designate path for visually impaired in cases where natural guide line (wall, curb, tarmac-grass interaction) is not present. This surface is shaped as elevated bars or notches (see figure 1.8). To create trails that can be followed with white cane. This type of paving is also common both in exterior and interior.

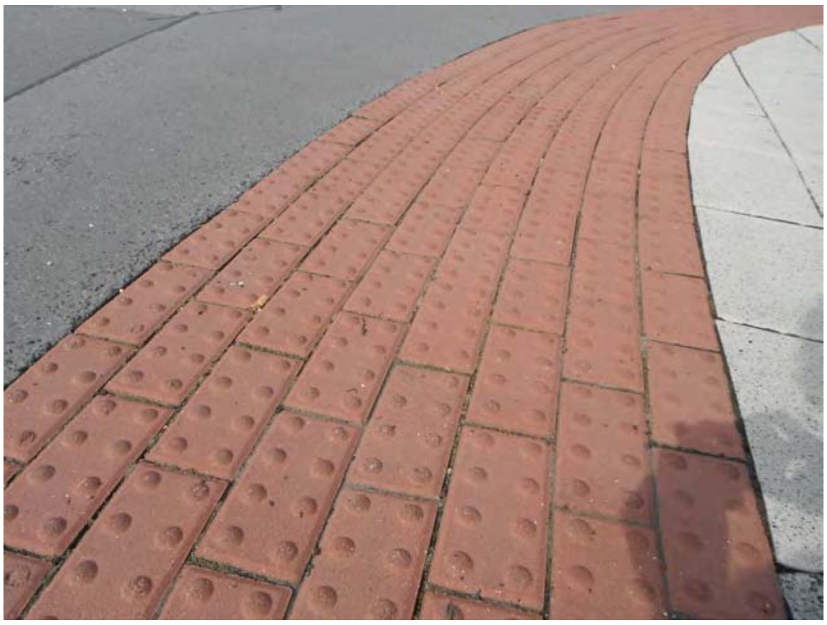

Figure 1.5: Blister surface

1.2.2.1 Possible use case

So how does movement of visually impaired looks like? One example might look like this. Coming out of his place of living to the street, he usually knows the close surroundings. He might know, that if he is going to go to the left and follow the wall, he will soon approach corner of the street, where guidance path starts. He follows this path to the right until he feels blister surface under his feet. This means that he approached zebra crossing and activates acoustic signalization by his hands. After he crossed zebra, he follows guide line further until he reaches different type of blister which is orthogonal to his direction. In this case it means he is on the bus stop (he must learn what this blister means before going out, he might contact someone to describe route to him, use tactile map or ask around). Now he looks for corduroy surface restricting bus platform so he is sure that he is not standing danger close to carriage way where bus stops. He uses bus and travels to desired stop using acoustic signalization. After getting out, he uses tactile paving and his white cane again to roam around the city. This is ideal use case and should work when blisters are not covered by snow, guide lines are cleaned of dust and no obstacles are places on them.

1.2.2.2 Installation problems

Not all existing tactile paving is automatically helpful for visually impaired. Incorrect use of tactile paving is not uncommon and causes problems and confusion. Study [39] examined tactile paving in 3 cities in the UK. Their conclusion is that more than 50 percent of installed tactile paving has some sort of deficiency, up to 5 percent misused warning patterns and more than 10 percent used wrong color. Such shortcomings be observed in other cities as well. Misuse of patterns and colors is only one problem; another problem is that even if used

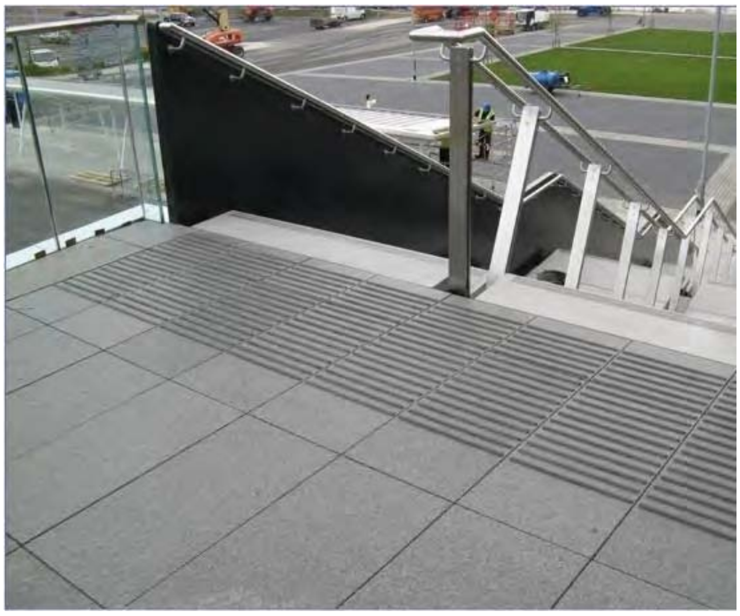

Figure 1.6: Corduroy hazard surface warning of presence of steps. Image source http://www.esi.info/detail.cfm/Visul-Systems/Surface-mounted-hazard-corduroy-tactile-paving-tiles/_/R-99426_NE388QA

correctly, they might not fall into architecture around. Guide lines might be on the wrong side of the road, where it could be more appropriate on the other side. Dangerous obstacles like elevated bus signs and road signs (table on a thin pole) should be avoided by guide lines, but sometimes it goes directly under such obstacle. In this case, visually impaired can detect thin pole by white cane which causes no trouble only to run into sign table with his head seconds after that.

1.2.3 White cane

White cane (see figure 1.9) is a symbol of blind people. It is a tool which grants them independence, enables them to move more freely and also alerts sighted people to their disability so they can act accordingly.

It is not just a tool, but also a symbol of impairment of its owner. White cane is simple, affordable and a must have for visually impaired people for them to be able to participate in a society. Its widespread use began in 1920's and 1930's when cane used by blind people was recommended to be painted white to make themselves more visible to motorists [49] and to distinguish from canes used by sighted people. White cane settled as a symbol of visual impairment. Soon, white cane was recognized by law and granted blind people carrying it right-of-way and protection on the streets [49]. In 1964, USA claimed 15th of October as a White Cane Safety Day [41].

White cane can be used in multiple ways and there are also several types of canes customized to these uses. Length of the cane is dependent on its use and height of the user [27]. Longest cane is used as a mobility tool to detect objects around. Shorter cane can be

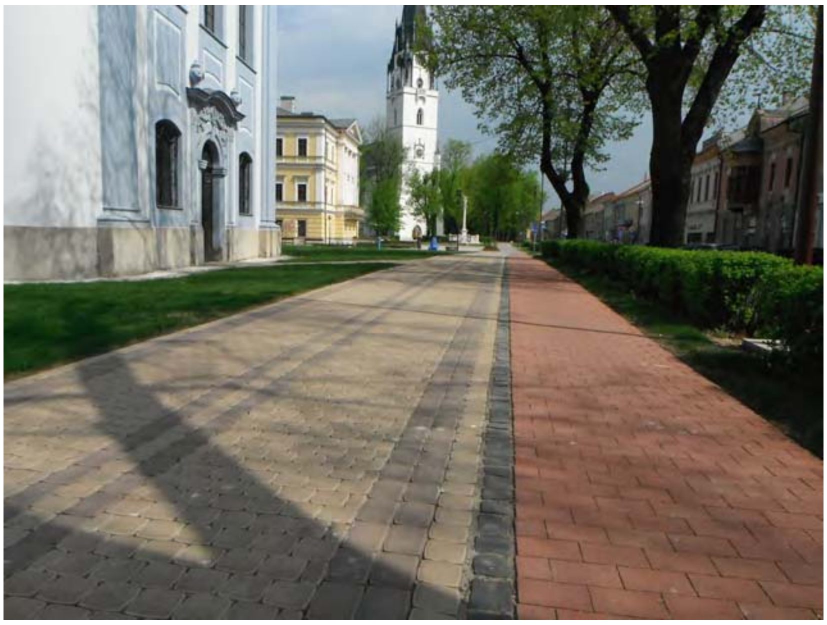

Figure 1.7: Delineator strip to separate bikers from pedestrians.

used as a mobility tool too, but is limited. Because it is shorter it can be carried diagonally across the body in front of the user as a protection and warning of obstacles. Even shorter cane called an ID cane has no mobility function at all and is only used to alert others to the fact that its bearer is blind [27]. Canes are usually made of aluminum to be light and durable. Some canes can be folded so they are easier to store and transport in a car.

1.2.3.1 Usage of white canes

Usage of white cane starts with learning how to use it in a right way. Children born blind are introduced to white cane in the age of 7 to 10 years [27].

Tip of the white cane can get caught in cracks on the sidewalk and can discourage a novice user from using it [44]. It takes time to learn how to handle it properly and comfortably. According to [44], user is over time using longer and longer cane as he feels more comfortable using it, increasing his range and mobility. It is also hard to use safely in a crowd.

One of the first things to learn is to swing cane slightly from side to side in front. The tip should slightly touch the ground. This is difficult to do in a crowd because one can accidentally hit other people and another problem is that cane can get stuck in a hole or crack. White cane is used also to walk down and up on stairs. Cane is dropped gently to the next stair to find its height and width, so one can safely do another step down [6].

But white cane is not the only mean for sensing surrounding. Listening to environment can give blind a very good image of his surroundings. Interior sounds different from exterior, sounds are reflected from walls, so he can sense these walls by hearing [23], road traffic can give him a hint, which way is road going, where are cars stopping on crossroad, etc. Blind can also feel the heat of sun on his face and turn himself or maintain course with help of this information.

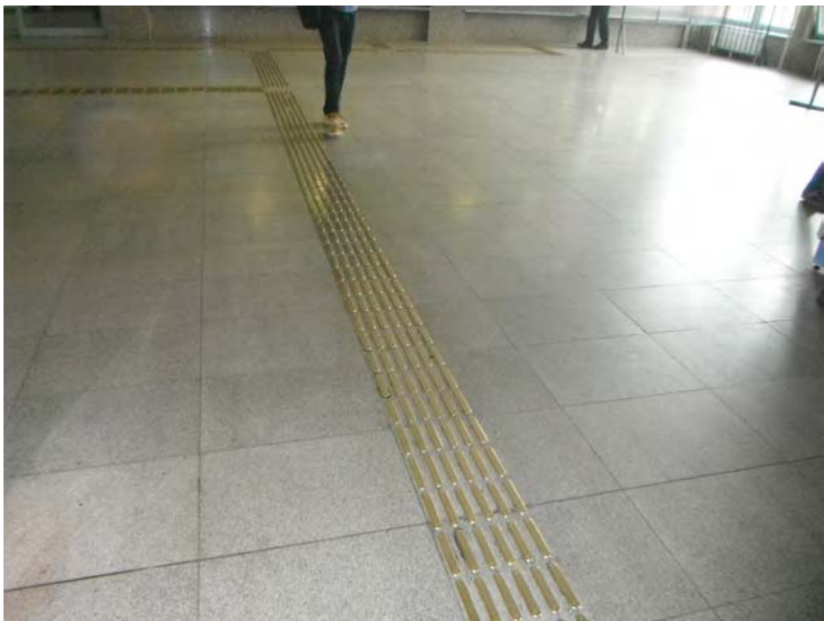

Figure 1.8: Guidance path.

1.2.3.2 Laser cane

In 1967, researchers developed an experimental laser cane [30]. This device was supposed to evolve into regular white cane extended to work as an early warning system to warn before open manholes, stairs and other dangerous obstacles. Since then, many such enhancements where proposed to the white cane. This includes sonar-based warning systems and other similar gadgets to help visually impaired move by detecting surrounding objects.

1.2.4 Guide Dog

Another help to visually impaired can be a guide dog. Guide dog is able to navigate blind person around obstacles, cross the street safely and many more. Guide dogs, just like a cane have been used for centuries. First professional guide dog training centers were established in Germany after World War I to help blinded veterans returning from war [46]. Since then, guide dog became widely used as a help to blind people. Training of a guide dog is complex and suitable dog has to be chosen for such training by passing the tests. These dogs are trained to walk on a straight line, stop at curbs and navigate around obstacles. After finishing this training, the team is formed – dog and his future master are matched and they are taught to work together. This team forming might take some time. Handler has to learn his new dog's moves and dog has to learn to obey his new master instead of an instructor [35].

This training makes a guide dog expensive and is not available to all visually impaired people. Organizations exist that help these people to obtain a guide dog, but waiting lists are usually long. In contradiction to white cane this is much more expensive option as after purchase more expenses are needed to vet care and food. Positive is that with guide dog,

Figure 1.9: White cane. Image source: http://www.brailleworks.com/blog/index.php/white-cane-law-announcements-from-the-nfb-fl/.

moving around can me much faster and fun. It also forces blind to go out regularly and also company of a guide dog is invaluable and helpful.

Guide dog and white cane are not meant to be used together, because it would occupy both user's hands, but on some occasions such people can be seen. While giving blind people directions, we should not rely on that he uses guide dog and a white cane together.

Guide horse

An experimental program dedicated to training and providing guide horses is run by Guide Horse Foundation [4]. They provide alternative to people who cannot use guide dog because of an allergy, fear of dogs or other reasons. Miniature horses (see figure 1.10) are used and trained similarly as a dog and can be used same way. Advantage of these horses is that their lifespan is much longer than lifespan of a dog. They are also intelligently disobedient – meaning that they are likely to disregard commands that can be harmful to handler. Disadvantage of having a guide horse is that is larger and heavier and might not be able to be transported using public transportation. It also has to live outdoors and is not suitable for indoor, which might be inacceptable to some owners.

Future of mobility tools

White cane and guide dog are two traditional and established mobility tools. Article [28] expands ideas how mobility tools would evolve using modern technology. It formulates three ways in which could evolution go. By embedding smart chips (ID tags) in environment a user could be able to easily identify surrounding objects by a handheld reader. This is called smart environments in [28]. Another proposed way is smart consumers – basically tools that are computerized and carried by consumer – in this case a visually impaired person. Final

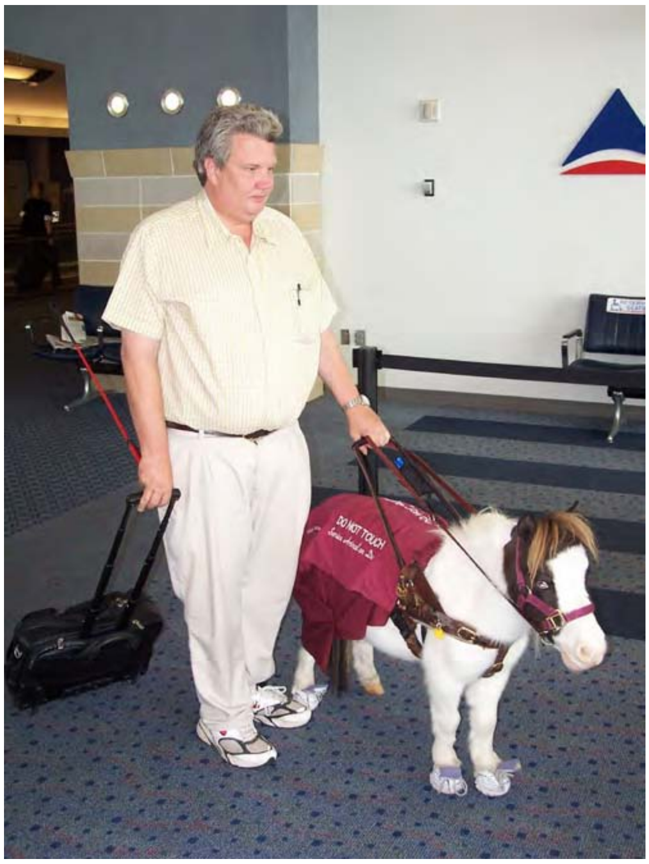

Figure 1.10: Guide horse. Image source: http://en.wikipedia.org/wiki/Guide_horse.

suggested way is to utilize humanoid robots as helpers for visually impaired people. [28] is concluded by 6 recommendations for discussion on future research.

1.2.5 Memory of blind people

It is a common knowledge, that blind people are superior to sighted people in memory intensive tasks. But is this true when it comes to proving this in real life? Study [47] shows that it is generally true. They conducted an experiment involving remembering a sequence of verbal input and reproducing it. Blind people were consistently better and they are indeed able to remember verbal information efficiently and faithfully – and most important, better than sighted people. They are also much better in remembering correct sequence of input. They are simply more capable of remembering things in right order – which is very good for remembering route directions. It has to be noted, that study worked with people with congenital blindness (they were born blind or visually impaired).

It should also be noted, that we are talking about auditory memory – memory that stores audio memories – sounds, speech, music etc. Sighted people also use their visual memory to perceive surrounding world. Essential part of this is human visual system and its visual cortex – part of the brain that receives visual input and processes it. This part of brain develops after birth by observing and perceiving surrounding world [20] so its ability to recognize patterns and other visual data is not given genetically. Blind people already born blind seem to utilize these parts of brain devoted to vision for other purposes – to develop other intellectual skills, such as superior auditory memory [31].

1.3 Data sources for computerized description generation

Crucial part of every GIS system is it's data storage. In general, any of usual means for storing data can be used, but they must fulfill three basic requirements. They must:

- be able to answer queries specific for GIS systems
- perform well while answering these queries
- work with very large amount of stored data (>RAM size)

Typical queries which are needed in GIS systems are these:

- Return all map features in specified area (rectangular, circular . . .).
- What is a distance from object A to B?
- Does map features A and B overlap?
- Does map feature A contain map feature B?

Map features are points, lines or areas representing all information that map contains. More specifically they are roads, statues, rivers, parks, bus stops etc.

Topological data is an example of two-dimensional data which is not located on a plane, but on a sphere (ever since the world is round). Note that measuring distance between features (and also length of roads, etc.) isn't as simple as computing distance between two points using Pythagorean theorem. This is intuitive way of doing so, but it only works in Euclidean space. Since maps describe surface of Earth, which is roughly spherical, we have to calculate distance along the surface of a sphere. In the field of navigation and geodetics this problem is called Great-circle distance [50] and is commonly solved by Haversine formula. Our data source has to compute distances in a correct way. Location of topological data is also usually specified in geographic coordinate system [2] by latitude and longitude.

Further, we would like our data storage to have additional features for ease of use and extensibility:

- import data from existing interchange formats (see [12] for exhaustive list of commonly used formats)
- interface for editing and updating stored data

We see that choice of data storage system is nontrivial. For our system we have several options:

- File storage + custom interface software
- Database + custom set of stored procedures for spatial data manipulation
- Web based data source + HTTP interface

File storage is a good choice as an interchange media and for archivation of data. Using it as a real time data source would be complicated, because extensive interface software would be needed for satisfying above-mentioned requirements. Such software exist (e.g. Osmosis), but is primarily accommodated for extracting specific pieces of data from file, to convert data to another format, etc. Not for querying in real time.

Main cause of poor performance is that volume of stored data is usually many times larger than the size of RAM available, therefore it must be read from disk sequentially. This problem can be solved by using one of database engines (MySQL, PostgreSQL, ...) for storing raw data. This solution will provide an effective way of caching data for fast retrieval.

Another problem is that queries for spatial data needed in GIS systems are complicated to implement effectively. They need an acceleration data structure such as kd-tree and fine-tuned algorithms to achieve good performance. Naïve implementation is out of question, because even linear asymptotic complexity is in this case unsatisfactory. Our goal is to achieve logarithmic query times (in relation to size of stored data).

Writing such software from scratch would time consuming and wasteful, since viable alternative exists in form of GIS extensions for database engines. We will look closer at PostGIS [18] extension for PostgreSQL.

PostGIS allows us to store spatial data in PostgreSQL database and retrieve them quickly. Query language is standard SQL so it is also easy to use. It allows us to answer spatial queries (some of them listed above) effectively. As an acceleration structure it uses generalized search tree [17]. It is compliant to OpenGIS standard.

1.3.1 Open Street Map Project

OpenStreetMap is editable and free map of the whole world [13]. Anyone can contribute to this project by editing the map and providing his own data. Data has to be acquired in any legal way – not by copying content from other GIS databases where license does not permit to do so. Preferable way is mapping are with GPS tracker and uploading data using one of OSM editors (Potlach [19], JOSM [7], . . .). Coverage of earth by OpenStreetMap expands very quickly and nowadays it covers most of the world in fantastic detail. Project is supported by non-profit organization that enables everyone to download and use its data.

Part of the OpenStreetMap project is plenty of supporting software dedicated to various tasks. This includes already mention map editing software Potlach and JOSM, Osmosis [15] – a command line tool for manipulating OSM data, Mapnik [10] for rendering map data as images, Osm2pgsql [14] to import OSM data into GIS database, The Rails Port [21] for web access to rendered maps and data access API and finally LibOSM [8] for accessing OSM API from Java based applications. We will look at these in detail in next section, where we describe our application architecture.

OSM provides all of its data through OSM API, which is REST based web service. Communication is done using HTTP protocol and its methods. This API however, does not support various spatial queries; it only returns information about individual elements keyed by their identification number. Intechange format is XML, which is well defined and can be processed by OsmLIB. OSM project classifies its data into 3 categories – nodes, ways and relations. Nodes are basically points on the map, they can represent large variety of map features. See [9] for complete list of XML tags that specify these features. Note that anyone can create new XML tags and use them. OSM API allows users to add previously unknown tags. This is feature that enables rapid expansion of project, because users are not limited to use only available tags and can add new features easily. Disadvantage is that anyone can user for same thing different tag, making it very hard to parse these data. Second data type - ways are usually roads, railways, rivers and similar objects. Third data type – relations is a combination of previous two to form new object. References to existing ways and nodes are specified to create relation. E.g. a tramway track (way) along with its stops (nodes) forms a relation of tram line with certain number and collection of bus stops.

OpenStreetMap gains popularity among researchers and also among general public. Because of flexible set of features and open API, many applications were created using this project that would not be possible using proprietary project. Examples are maps customized for cycling, history tours, geocaching, many other sports an also in humanitarian aid.

1.3.1.1 OSM stack

Our data source of topological data is combination of applications from OpenStreetMap project. We use Rails Port [21] application which is a core application that provides OSM API (currently API in version 0.6). This is HTTP-based API for retrieving and submitting data. Several client libraries exist which provide easy access to this API from client applications such as LibOSM [8] for Java.

Rails Port application stores its data in PostgreSQL database. They can be uploaded to database using Osmosis [15], another supplementary program from OSM project.

Two more programs from OSM project are utilized - Mapnik [10] and Osm2pgsql [14], detailed description of cooperation of these programs will follow.

1.3.1.2 Slippy map generation

As we not only want to store and read our topological data, but we also want to see them in a form of a map, we need a program for conversion data to visual form. OSM project provides application called Mapnik [10] for rendering data stored in GIS database. Another support program is Osm2pgsql, which has similar purpose as Osmosis, that is to say, upload data to GIS database, but it also preprocesses them for Mapnik.

Mapnik generates a slippy map (see figure 1.11) which can be viewed and manipulated in a web browser using Javascript API. This provides a basic user interface for spatial data manipulation. Slippy map is known GUI element of all modern browser-based map viewers.

1.3.1.3 Data acquisition

GIS data can be acquired from several sources. Many governments provide GIS data of their territory in one of many interchange formats. Several private companies provide GIS data and also NASA organization provides elevation maps of earth. Data can contain

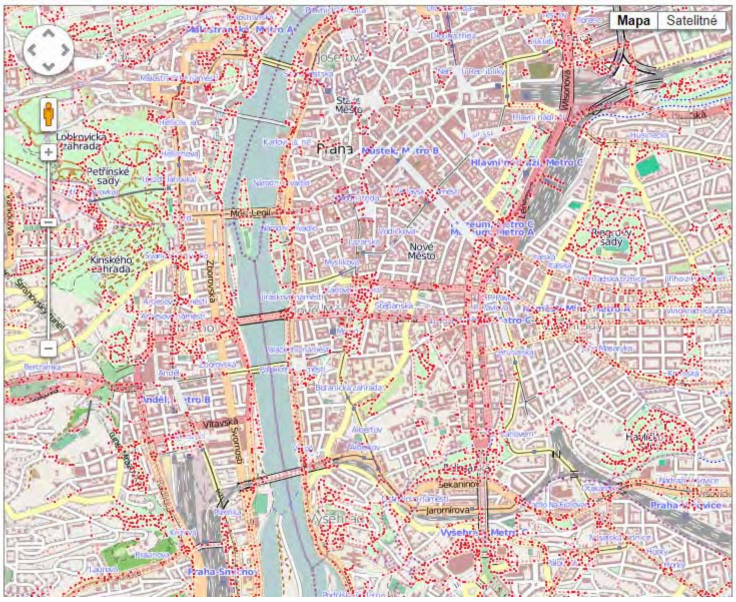

Figure 1.11: Slippy map screenshot of our application.

different sets of features and can have varying resolution. When choosing data source we need to consider their availability, price, license, resolution, features and area they cover. Care has to be taken for a license under which are data distributed. Some licenses can restrict usage to an extent that they become unusable in our system. We chose data provided by Open Street Map project. They are very detailed, updated frequently and provide an extensive set of features. They are free of charge and licensed under Creative Commons license.

1.3.2 GPS and RFID based systems

Effort has been made to make orientation of visually impaired easier by utilizing GPS and other localization systems. Some of basic requirements put on these systems is position recognition, obstacle detection and path finding. Paper [51] presents a Smart-Robot, a robot utilizing GPS, RFD and embedded systems to assist blind user. It provides audio and haptic feedback through small speaker and a glove with vibration motors. It enables user to follow a predefined or new route and warns him before obstacles. It uses infrared and ultrasonic sensors to detect obstacles. It guides user through haptic glove, turn to left or right are expressed as vibrations of different fingers. Reaching a landmark is announced by speaker. Routes can be predefined by guiding robot through the route, while it stores GPS locations periodically. Route can be activated later and robot guides blind person according to stored GPS coordinates.

1.4 Prototype system architecture

Our application is in development for several cycles and evolved into fully functional prototype. Core application is written in Java and runs on Java EE platform. Client application is web-based and is written in Javascript. It is part of Java EE application and it is served to end user like a regular web page. It communicates with core application using SOAP messages sent through HTTP socket to web services deployed on our server machine.

Server is a stack of numerous applications that feed our core application with various data. Complete list of running applications is as follows:

Apache HTTP server -– serves rendered images of map to slippy map software running on client machine in a browser through HTTP protocol. It is primarily used by our client application

Rails Port – Ruby on Rails application that provides OSM API v0.6 on dedicated port. This application is basically our own OSM server and we use it to acquire OSM data elements. It is run using Apache server module for running Ruby on Rails applications -– The Passenger [16]

MySQL database -– this is a database of Rails Port application where all OSM data reside

PostgreSQL with PostGIS extension -– this is our GIS database which supports answering spatial queries. Note that we cannot use only one database engine (MySQL or PostgreSQL) because Rails Port cannot use PostgreSQL and there is not powerful extension to MySQL to support spatial queries like PostGIS. Therefore, we use two databases and we reference data in MySQL from PostgreSQL by their id numbers so essential data are not duplicated. PostgreSQL is also used by Mapnik map renderer.

Glassfish application server -– this is server that runs our core Java application, provides web services that expose functionality of our system and server our web client to end users.

Except these applications, there are several supporting applications that our system needs to be fully operational. These are primarily used to import data to our system and process it so it can be used by our application. These applications are all command-line based and can be used remotely utilizing SSH client:

Osm2pgsql -– this uploads specified file to PosttgreSQL database (populates our GIS database with data)

Osmosis -– same for MySQL database (populates OSM API with data)

Mapnik -– renders images for slippy map. These images are used in clients browser to show map in various zoom levels.

Look of map can be tweaked by configuration XML file. We use two separate configurations and generate two layers. First layer represents normal urban map of the city. Second layer is semi-transparent and show only clickable footways and nodes that can user select. We call this an active layer. In case of need to update our map, only active layer needs to be generated again. Footways are still being added every day, so there is a need to update this layer frequently until all of the footways are mapped. Because city architecture does not change drastically over time, second static layer can be reused.

This was an overview of applications used by our system and their purpose. For installation and maintenance instructions see appendices B and C.

1.5 Delivering the route description

While solving the problem of delivering description to blind user we have to take into account, that blind person can perceive it only by hearing or using Braille. Therefore, written form in this context means that user will read it using screen reader like Jaws or nvda, therefore it will be still „spoken“ to him. Difference is, that while communicating with another person who gives him directions, he can communicate with this person, provide feedback and ask questions.

Written form might be preferable, because it can be read in small parts, does not have to be remembered, can be replayed many times and can be stored a long time.

Paper [33] concludes that major role in understanding spoken versus written route directions plays working memory, which is responsible for temporary storage of perceived information. Limitations of working memory have to be taken into account and description has to be properly adjusted so it does not overwhelm its target audience. Spoken descriptions are more memory intensive, but sentences are usually shorter and are less complex grammatically.

Interesting result of study conducted in [33] is that people who created spoken descriptions were using more prescriptions – they were specifying actions user has to do. On the other hand, people who created written descriptions were more descriptive – they were describing surroundings and landmarks. Although we will generate our descriptions by machine, this is important knowledge of how people think about route descriptions; there are two ways of doing it – prescriptive and descriptive. Paper [33] states that there is not a preferable way, both ways are roughly equally good. We will have to find balance between these two and also try to evaluate which is better for visually impaired users.

In both cases, receiving the route description activates visual and spatial thinking in emotional and creative part of brain. Since we deal with blind users, let’s look how this is different from visual thinking of a sighted man.

Chapter 2

Analysis and design

2.1 Route description analysis

We will divide a process of creating a route description into several steps. After that we can analyze each step and discuss a solution.

1. Find optimal route through city from point of departure to destination. Optimal might be shortest or optimal in other sense, e.g. route that avoids dangerous crossings or steps.
2. Split route into segments. Independent segments are autonomous description fragments that can be understood and executed individually one by one. Their complete sequence forms a whole description.
3. Generate description of individual segments with respect to each other, so they are interlocked and can be read fluently.

Steps (1) and (3) have been also covered in works [45] and [29]. We will now focus on step (2) – splitting route into segments and improving step (3).

2.1.1 Splitting route into segments

Our task is to identify points, where will current segment end and a new one begin. Obviously, the start of the first segments is in point of departure and the end of last segment lies in destination point. Although very simple to identify, these points are special in a way, that there are some requirements on them:

- **In the point of departure**, we have to be sure that our user is really on the right spot and that he is facing right way (he isn't coming from any direction – or at least we don't know about it). This might need extended description.
- **In destination point** we have to make sure, that he will recognize the end of route. Usually, destination isn't just an address or place on the street, user wants to get to the shop, enter the office building or find a bus stop. Describing these final steps can be helpful to make sure he finds what he looks for at the end of the route.

Splitting the route into segments is necessary to make route description usable. There are two main reasons for that. 1. User cannot memorize the whole description and 2. It is useful from time to time to stop, make sure that user is progressing correctly and proceed to next waypoint. To deal with these requirements we have to split the route into reasonable pieces. Making a decision, where to split the route is nontrivial to do programmatically. Firstly, we have to state properties that should route segments have:

- **Reasonable length** -– this might be different from user to user, because it is dependent on their preference. We must not forget that we are dealing with blind users that may have superior memory skills [31].
- **Information about how to proceed** which should be as foolproof as possible. Make instructions clear so there is a little room for error.
- **What we encounter during the walk** might be useful to make sure, that we are proceeding the right way
- **Warn before possible danger** -– some situations might be dangerous for blind people, we have to warn them if possible
- **Enough information to safely identify the end of segment** -– so user know he reached the end and can proceed to next segment

Looking at properties we want to achieve, we see that they are dependent on user preferences and even more dependent on amount and detail of data that can be obtained to construct description. From possibly very large set of data that can be obtained from GIS storage we have to select only relevant data. Quantity of information has to be balanced. Too much information might be overwhelming and confusing and too little might be not enough to ensure safe passage or user might get lost.

2.1.2 Selecting relevant data

While there might be a lot of map features in vicinity of given segment, our question is, what will be relevant to the user. Before answering this question, we have to decide from what set of features are we going to pick important ones. We need to take into account technical capabilities of computer, because it is not able to process whole map in reasonable time. Intuitive answer would be, that we will load all the features that can affect the user or guide him during his route and pick important ones.

But what can affect user on its route? It can be anything that can be felt by his senses. Whether it is radio tower far away or noise of the highway it can be useful for description, but it can be also far from segment effectively enlarging the set of features in question.

Since we are dealing with blind people we can focus on their remaining senses: hearing, touch and smell. We can make further assumption that important features he or she can encounter is directly on the street. Other features that can affect him are in the range of his hearing.

Taking this into account we can limit loading of features for generating description to these located several dozens of meters away at max.

2.1.3 Identifying landmarks

Importance of features that will be useful for blind people is dependent on their preference and context in which they arise. We will call such features landmarks. One might be relevant in some situations and not in others. For example a corner restaurant can be easily identified by noise and smell. It is very relevant if one has to make a turn on that corner so he has a restaurant by his right hand. But if one passes by restaurant without any change of direction it might not be important at all. Description should in such a case point out a crossing that user has to take. Landmark does not have to be a building or a sign. It can be any object that can be perceived by user.

In paper [43] they conducted research where they tried to identify good landmarks in different contexts and proposed method for extracting these landmarks from written descriptions. Quality of landmark was evaluated using agent-based algorithm. They state that landmarks can be divided into two types – large and small. Large are usually buildings easily visible, so they can be followed. They specify direction. Small are landmarks are used to mark corners and turns.

This approach is not directly applicable to our situation, since blind user is not able to see large building on the horizon, but we can make use of concept of two types of landmarks. One type to specify direction such as tram track identified by noise that can be followed and second type to specify stops and turns.

2.1.4 Description text generation

Generation of human readable text tackles with a problem that machine generated text tends to be highly bound to underlying data representation. Data structures from GIS database tend to project directly to nouns and adjectives to describe map objects. If not used carefully, this can lead to not very ergonomic text full of abrupt descriptive information without context and relations between them. A desired feature is to produce natural sounding linguistic descriptions. Paper [32] lists 3 cases where machine and human generated route descriptions differ:

- Humans often omit steps that seem obvious
- Machines define turning points by amount of time and/or distance to them. Humans use rather description of landmarks by the turning point
- Machines typically produce one sentence per action to be taken, humans use more complicated sentences

Fact that description was created by human does not automatically mean that it is better or more accurate, but is usually better understandable and better reflects how human thinks. Therefore it is desirable to produce such a description by machine. Except naturally sounding for easy understandability, it is also required that the description is memorable. According to [32], description of route consists of messages, which contain one standalone piece of information. To communicate messages to user, messages need to be aggregated to sentences. Authors of [32] aggregate messages to sentences by different mappings:

- **One message per sentence:** This mapping creates one sentence per message. This mapping is naïve, but might be preferable in some situations.
- **Path + Point:** This mapping aggregates several messages that describe point into one sentence ending by message describing a destination. Typically this is used in a situation where user has to walk along some path until desired location.
- **Path + Direction:** Similar to previous but at the end of the sentence prescribes new direction to be taken.

Second and third mapping can prescribe ending location at the beginning of the sentence or at the end. This is a matter of preference and might by adjust for different users [29]. Generated sentences should take into account what knowledge user has, they have to be effective and unambiguous.

Following route directions activates visual thinking center in our brain and triggers cognitive processes specific for this activity. Therefore, a good route description has to be cognitively ergonomic - comprehensive.

What happens is that route description consumer processes spatial relations given in direction and follows actions. According to [37] we have to convert spatial data to spatial relations described by correct semantics that puts reasonable amount of cognitive load on user. Also route description which has to be cognitively ergonomic should be focused on quality, not quantity.

One of ways to achieve this [37] is spatial chunking. According to this method a route is segmented into chunks. Each chunk representing autonomous instruction set to follow. Created hierarchy offloads user from all the details which are not needed at a time.

2.2 Algorithm design

2.2.1 Design goals

- Balance between descriptive and prescriptive navigation strategy
- Natural sounding linguistic
- Just sufficient amount of information
- Adequate amount of small and large landmarks
- High relevancy of information
- Adjustability for different users
- Description tailored for delivery in written (not spoken)
- Acceptable consumption of hardware resources

2.2.2 Proposed algorithm

To fulfill these goals we propose subsequent algorithm. It is divided into several steps that can work independently. These steps can be seen as black boxes that take input from previous block and give output tu subsequent block. By storing these intermediate results we can create database of data on certain level of abstraction that can work as cache of spatial data and still be capable of customization for different users.

Our algorithm works with several data objects. These object abstract raw spatial data in several levels, each step of algorithm means another level of abstraction (see table 2.1). See listing 2.2 for pseudocode. Algorithm works in following steps:

1. Segmentation
   (a) Non-informed
   (b) Informed
2. Route elements creation
3. Route messages creation
4. Description generation

2.2.2.1 Segmentation step

Input to segmentation step is a list of continuous fragments of route as it is received from client application running in a browser. They always start where previous segment ends but these common points are duplicated for each segment so they are completely independent. We assume that after this step, segments will be used directly to generate description without further processing of their locality or topology. Therefore we have to take into account that the only possible time when user can stop and get information how to proceed further is on the end of the segment. We have to bear this in mind while choosing where to split segments.

We accomplish segmentation of such route in two steps. In the first step we combine segments together based on their locality. We call this step **non-informed segmentation** because no information about segments except their position is exploited. In this step we carry only one operation – combining more segments into one larger segment. No segments are split nor created now. Like this we get rid of unnecessary (or possibly unnecessary) segments (see figure 2.1). These redundant segments will always be present, because they arise from topology of urban area and its representation by vector map. They can also be introduced by user who planned the route. Because of this, this step is always necessary and we can think of it as a sanity check of input data. We don't have to be worried about combining together segments, where we want to have a split, because in such a case the split will be generated in next step.

Second step – **informed segmentation** makes use of map features pertaining to segments. With this information we can split segments where a break is needed to provide user more information. Typical example would be a zebra crossing or steps.

2.2.2.2 Route elements creation

In this step, we introduce data object called route element. This is a wrapper object for OSM element, but carries more information. We only wrap OSM elements that are recognized by our system – we do not add any objects that are unknown (although they might be useful, we do not know how to parse them). This is different approach as used in previous prototype, which tried to extract name and type of such elements and write it out in description. This was considered very unhelpful and redundant by users, as it often added sentences like „you are crossing metro tunnel, line A“ which are of course completely useless.

To create route element, we recognize type by XML tags of element and add additional information needed that can be extracted from raw spatial data. We create collection of route elements for each input segment and sort it according to distance from segment start. Output of this step are segments enriched by route elements.

Result of this processing is higher abstraction level than provide raw spatial data – these route elements are of known type and purpose and can be used in a context they occur. Because of this, these elements are suitable for storing as an intermediate product of our algorithm in relational database. They contain all information that can be obtained from spatial database, but are not processed for specific route and preferences yet. This makes them ideal for storing and retrieving as data associated to given segment. Every time same segment occurs in route, we do not need to query relatively slow OSM API and spatial database, but we load these elements from fast relational database and use them as input for next step.

2.2.2.3 Route messages creation

In next step, we process incoming segments and parse elements to create new object called route message. Route message is another level of abstraction. In this case, these objects do not represent physical entities of real world as route elements, but are intended to communicate certain information. This means, that they are enriched by context. This is an important step in getting human-like description. Since now, we do not work with pieces of map data, but with pieces of reasonable information that can be translated into text.

Route messages are created by two ways. First, route elements are parsed into messages by evaluating their context and second, additional messages are included to resolve relations between segments and/or communicate other information. Details of this step are also discussed in next chapter.

2.2.2.4 Description generation

To generate description, we need to employ some type of mapping from messages to sentences. We designed a powerful many-to-one mapping that generates very good results, for implementation details see next chapter.

Alg. Step	**Working Data**	**Level of Abstraction**	**Notes**
Non-informed segmentation	Segments topology	Works with raw map data	
Informed segmentation	Topology + raw GIS data	Works with raw map data	Data from GIS database are loaded for each segment and segmentation is done according to them.
Route elements creation	raw GIS data	Raw map data and processed elements	Processes raw map data into elements recognized by next step, gives them meaning - higher level of abstraction, but still closer to map data
Route messages creation	Route elements	Evaluates context of route elements and creates pieces of meaningful information.	Abstraction level is much higher, raw map data or segments cannot be already recognized in this step. Data is closer to description than to spatial data.
Text generation	Route messages	Highest level of abstraction	Works with standalone pieces of information rather than map data.

Table 2.1: Algorithm steps and abstraction level of data they are using. Abstraction from raw map data is higher and higher until raw map data cannot be even recognized and data objects represent rather pieces of information than map features.

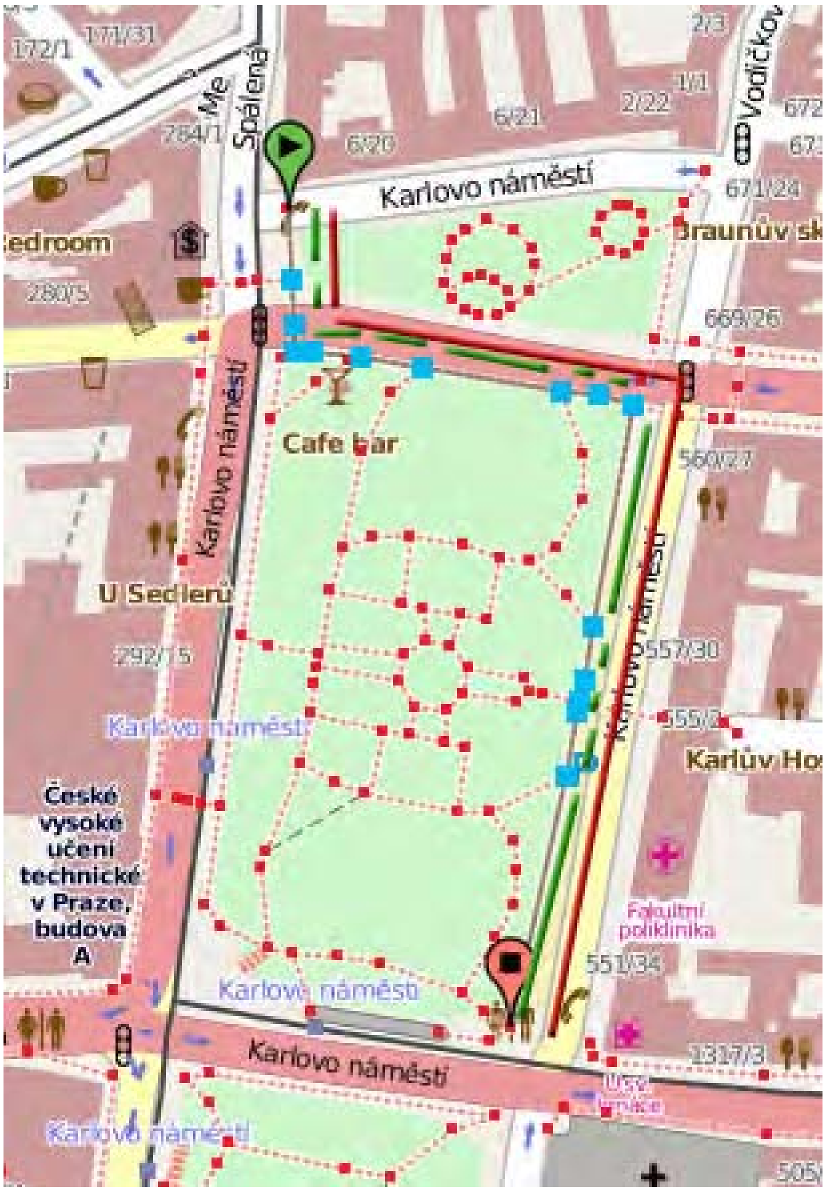

Figure 2.1: Fragmented route. 12 small segments (green) will be combined into 3 larger segments (purple) based on their course

```
GENERATEDESCRIPTION($InputSegments$)

main
  $ProcessedSegments \leftarrow$ DO-SEGMENTATION($InputSegments$)
  $RouteElements \leftarrow$ CREATE-ELEMENTS($ProcessedSegments$)
  $RouteMessages \leftarrow$ CREATE-MESSAGES($RouteElements, ProcessedSegments$)
  $Description \leftarrow$ MAP-TO-SENTENCES($RouteMessages$)
  $RETURN Description$

procedure DO-SEGMENTATION($InputSegments$)
  comment: Non-informed segmentation first
  $OutputSegments \leftarrow \emptyset$
  for each $s \in InputSegments$
    do
      $deltaCourse \leftarrow$ change of course between subsequent segments
      if $deltaCourse > threshold$
        then $OutputSegments \leftarrow$ new combined segment

  comment: Informed segmentation second
  $bbox \leftarrow$ CREATE-BOUNDING-BOX($InputSegments$)
  CACHE-ELEMENTS-FROM-DATABASE($bbox$)

  for each $s \in OutputSegments$
    do
      comment: Iterate intermediate nodes that formed original segments
      for each $node \in s$
        do
          LOAD-NODE-INFO-FROM-CACHE($node$)
          if has to be made on node
            then split segments into two
  $RETURN OutputSegments$
```

Figure 2.2: Pseudocode of our algorithm (Part 1)

```
GenerateDescription($InputSegments$)

 procedure CREATE-ELEMENTS($ProcessedSegments$)
  $RouteElements \leftarrow \emptyset$
  for each $s \in ProcessedSegments$
   do
        $bboxLeft \leftarrow$ CREATE-BOUNDING-BOX-LEFT-OF-SEGMENT($s$)
        $bboxRght \leftarrow$ CREATE-BOUNDING-BOX-RIGHT-OF-SEGMENT($s$)
        $elemetsLeft \leftarrow$ LOAD-FROM-CACHE($bboxLeft$)
        $elemetsRight \leftarrow$ LOAD-FROM-CACHE($bboxRght$)

        for each $element \in elemetsLeft$
         do
              comment: If element is recognized, its XML tags are parsed
              comment: otherwise it is discarded now
              $element.position = ON\_LEFT$
              $RouteElements \leftarrow element$
        for each $element \in elemetsRight$
         do
              comment: If element is recognized, its XML tags are parsed
              comment: otherwise it is discarded now
              $element.position = ON\_RIGHT$
              $RouteElements \leftarrow element$
        comment: Sort elements of segment by distance from beginning in ascending order
  $RETURN RouteElements$
```

Figure 2.3: Pseudocode of our algorithm (Part 2)

```
GENERATEDESCRIPTION($InputSegments$)

 procedure CREATE-MESSAGES($RouteElements, ProcessedSegments$)
  $RouteMessages \leftarrow \emptyset$
  for each $e \in RouteElements$
    do { comment: Evaluate context according to description in text
       { $RouteMessages \leftarrow Correspondingmessage$
  for each $s \in ProcessedSegments$
       { comment: If courses changes, insert instruction to turn message
    do { comment: Always generate pause message
       { comment: On end insert reached destination message
  $RETURN RouteMessages$


 procedure MAP-TO-SENTENCES($RouteMessages$)
  $RouteMessages \leftarrow \emptyset$
  $Groups \leftarrow \emptyset$
  for each $m \in RouteMessages$
    do { comment: Group messages of same type until next pause message
       { $Groups \leftarrow groupedmessages$
  for each $g \in Groups$
    do { comment: Generate corresponding message for each group
       { comment: Ordering and grouping is closely described in text
  $RETURN Description$
```

Figure 2.4: Pseudocode of our algorithm (Part 3)

Chapter 3

Implementation

3.1 Segmentation step

3.1.1 Non-informed segmentation

First step of segmentation is informed segmentation, as described in design section. In this step, we cannot use any map information. To implement this, we have to create set of rules by which will be route segmented (split or combine existing segments).

The rule for combining segments is this: If succeeding segment forms with current segment a turn and this turn is sharper than 20 degrees (value can be tweaked), then these segments are preserved as they are. Otherwise they are combined into one segment. Computing the turn angle is easily accomplished by taking dot product of normalized direction vectors of these segments. This however gives us a cosine of this angle and doesn't have to work well if vectors are in different quadrants (angle computed from inverse of cosine is ambiguous). This issue can be solved by using `atan2` function implemented in many mathematical libraries. Also this method works in Euclidean space, so now we run into issue that coordinates of segment's endpoints are given in geographic coordinate system (specified by latitude and longitude) and are actually representing points on earth (sphere). In navigation this is called a problem of computing course (or bearing) between two points. Like the Haversine formula for computing distance on earth surface, there is an algorithm for computing course and is described in Aviation Formulary [50]. This algorithm is also implemented in library LibOSM [8] used by our prototype application, so we use this to compute angle between our segments.

Ids of nodes that were removed during this step are stored in combined segment. This information will be useful in next segmentation step (otherwise it would have to be recovered).

3.1.2 Informed segmentation

In next step, informed step we already can use information from our data sources. We must load all the information about the area of our interest from GIS database. If we specify our area of interest by rectangle, this information will be basically all data stored in database which fall within this area. Now the clever choice of data source from chapter 1.3 comes handy, because GIS database systems are able to give us this data effectively. Typically this data will be map features like roads, statues, traffic lights, restaurants useful for our application. There will also be a lot of unnecessary ballast like underground railroad tracks in subway. To counter this we can create a custom database query to filter these data for us, possibly boosting database performance and shrinking dataset we will have to process, but since the set of XML tags and attributes used to describe data in OSM is not settled, it is not easy to do and it is safer to load all data without the danger of possibly filtering out useful data.

Our data source for this project consists of two independent data providers. First is OSM API – RESTful HTTP based interface for accessing data from OpenStreetMap project. We access OSM API using LibOSM – client library for Java applications. Results of our requests are data elements corresponding to given queries. These data elements are XML based representation of map features which can be either a node, way or relation. Each of these has a set of attributes (formulated as key-value pairs) stating their type and properties. They have also associated a position in world coordinates and unique identification.

Second data provider is our PostgreSQL database with GIS extension. We use this to quickly evaluate which OSM map elements fall into our region of interest (purpose of this database is also to generate customized slippy map for our application, see chapter 1.4).

Although OSM API supports retrieving elements in bounding box, this functionality doesn't seem to work in current configuration (OSM API v0.6 + LibOSM). Instead of this, we rely upon our PostGIS database.

By querying PostGIS database for ids of elements in our region of interest we get a collection of these ids and we can use this to request detailed information about them from OSM database through OSM API. Here we can use our own OSM server, but we can also redirect these requests to public OSM servers. Note that if ids in our database are not consistent with ids on public OSM server, we will get invalid data.

Obtained set is unorganized „soup“ of spatial data, so next step is to sort this data between our segments so it can be used in a meaningful way. Now a new problem arises. Some data are a simple point (node) but others form lines, polylines and regions. If assigning a point to single segment was a difficult task (in ambiguous cases), this makes things even more complicated.

For our purpose, at this moment we just need to know, where to split the segment. This split can be wanted in various situations:

- **A change of direction is needed** (this case is handled by previous segmentation step)
- **Obstacle has to be avoided (or warned before)** -– obstacles such as stairs, etc. This split is usually determined by change of attribute of way element; therefore a split has to be made according to some attribute change.
- **Stop at a zebra crossing is needed** -– technically can be handled similarly as previous case
- **Ambiguous situation has to be clarified** (Y shaped crossing, etc.) – we have to determine relations of segment with another ways. If there are any crossings, we might consider splitting a segment on a crossing to provide more information about new direction.

Implementation-wise, we have to detect change of chosen attribute along the path. To do this, we have to get OSM way elements associated with examined segment. This process goes as follows:

- Input to this segmentation step is a segment with world coordinates of its start, end and also intermediate nodes that existed before combination to single segment. We know that split can only occur on these intermediate nodes, because only there can OSM attribute change.
- We load node and ways elements from OSM server that correspond to intermediate nodes world coordinates
- We track change of selected attributes between ways going in and out of intermediate nodes. If such a change occurs that split is desired, we split segment at this node. Currently these changes are tracked, but it is easy to extend system by new attributes:
  - Attribute `highway:steps` -– indicates that steps are on the way, user should be noticed about this
  - Attribute `tunnel:yes` and `layer:-1` -– means that an underpass is on the way

Now we come into trouble that set of OSM tags is not fixed and meaning of these tags is not strictly defined. Every OSM user can use whatever tags he wants. Although this provides very easy way to introduce new map features and freedom of choice, it makes machine processing of such a map considerably harder. We can counter this by enabling easy customization of system, which we already established. There is, however, one exception to this and that is road crossing or zebra. Zebra crossing is map element which should be represented by a line, but for some reason, in OSM system it is represented by point (node). Therefore, we have to examine attributes of nodes on the way too . If one of the nodes has attribute `highway:crossing` or `railway:crossing` associated with it, it means that path from previous node on the way to the next node is a crossing. We handle this case so, that when we detect such a node, we remove it and split segment on node after it, so user can be instructed how to proceed after going through this crossing. According to user preference, we might also split segment just before the crossing.

Some users may prefer segments not to be split on crossings at all. In such a case, route description might look like this: „Continue straight ahead, pass 2 highway crossings with traffic lights and turn right on the corner“. This is however a target of experiments to be made to evaluate best configuration suitable for users of various level of experience.

Next step after segmentation is determining what information should be provided to user at the beginning of each segment. According to stated design goals, we have to find balance between quality, quantity and desired verbosity level.

3.2 Creating route elements

Now we have prepared segments we needed and we are going to generate route description for each of these segments. This description will be provided to user at the beginning of each segment. To create this description, we will use an idea of method [32] – aggregating messages into sentences. In our design, we will load content associated with given segment – we will call these standalone pieces of data „route elements“. Each element will produce a single message. More messages will be added later to describe relations between segments, such as command to turn etc. All of these messages will be later aggregated into final description by some mapping.

We start by loading all OSM nodes in the vicinity of segment. We query polygonal area to the left and to the right of segment separately. This way we get elements that can affect segment and we know their relative location to the segment. To implements this, we benefit from using GIS database again. PostGIS function `upgis_lineshift` is used to create polygonal area parallel with segment. Its width is matter of discussion, in our implementation we use value `0.00007` (world coordinates), which corresponds to approximately `7.8` meters. This value was used in prototype application in work [45] and was chosen to work well in this scenario. We get ids of nodes in this polygonal area from PostGIS and query nodes by ids from OSM data source. Although we could load nodes from „soup“ of spatial data we got earlier, it is easier to use PostGIS database to do our query, because effective algorithms are already implemented there, so we do not have to create them ourselves. Performance hit is marginal because all data in question are already cached in LibOSM by this time.

Duplicities in collections of nodes to the left and right does not mean that these nodes are on both sides, but on the contrary — they lie in the center and we can extract them to separate collection. Doing this we get three collections of elements -– collection of elements on the left, on the right and center.

We also load ways and relations for each segment in a similar manner as before and attributes of intermediate nodes of a segment (including start and end node). This is supposed to pre-cache data in LibOSM. Because single query (even large) is faster than many smaller queries, we load all elements in a single query like this. Even if we won't use all information we get, it is still faster than querying each element individually.

After this step, we have 5 arrays of OSM Elements of various types – there are nodes, waynodes, ways and relations. Waynode is a regular node that lies on the route. It is used to specify shape of the way and its class therefore does not conatin all available information – just an ID. We convert these to standard nodes by querying for a node with same ID and we get complete OSM node.

Now we have loaded all data relevant to given segment and we have classified it according to its data type and position into several arrays. Next step is to create route elements from these data. Route element is object representing information that can be used for decription creation and can be translated to „route messages“ which will be explained later. Here, many of route elements get discarded, because they might not contain useful or relevant information. It is here, where we can decide what information to include. Arrays of elements are:

- Nodes to the left
- Nodes to the right
- Nodes on both sides (center)
- Nodes associated with segment (on the path)

- Ways to the left
- Ways to the right
- Ways associated with segment (on the path)
- Relations to the left
- Relations to the right

With each route element we store its type, OSM tags and distances from beginning and to the end of segment. We also store relative position (relative to start of the segment) which is one of these 4 options:

- LEFT
- RIGHT
- CENTER (both sides)
- ON_PATH (originally associated with segment's way)

Now we sort all elements by their distance from beginning of the segment. Now we have prepared all relevant information about all segments (for each segments individually) and we can generate route messages from these elements and add more messages between segments to clarify any relation between segments.

3.3 Creating messages

Our goal is now to generate human readable text from array of route elements. We want information that is carried by these elements to be translated to final text. One possible solution would be to generate one sentence per element. This naïve approach would not work, because of tremendous amount of sentences, often very short would be generated. This would not sound natural at all. To solve this we would try to aggregate multiple elements into single sentence. We would soon get into trouble that the context in which route element is used is not trivial to see (programmatically). Take an example, when we are aggregating all elements on the left to create sentence: „On your left you will pass ...“. What if last element is zebra crossing? Do we include this in this sentence, or do we create another sentence: „On the corner turn left and use the zebra crossing“? Because we do not know the context in which the element occurs, we cannot decide correctly.

To solve this problem, we added another step in processing elements before generating description and that is creation of route messages. This message specifies, whether element is used in context of informing user about his surroundings, instruction how to proceed, information about surface, etc. While creating these messages, context of each element is examined and appropriate message is generated. Another advantage of this approach is that we can include messages not related to any element. These messages are typically instructions how to proceed at the end of each segment. There are two ways, how a message is created. First is parsing route element to create a message. You can think of a message in this case as a wrapper around route element that adds context information to it. In the second case, message is just carrying information that needs to be added to description without relation to any physical map element. For list of recognized map elements see table [?].

3.4 Parsing elements to messages

First step is to parse all elements of each segment and create messages. There is one-to-many relationship between message and element, meaning that each element creates one or more messages. More messages can be created when e.g. way element carries information about its slope and also specifies railway crossing. In such case two messages would be generated – one to tell user that slope of his way has changed and second that he has to use a crossing.

When parsing element, we start by processing its relative position. From 4 previous options we simplify to only 3 - it is either on the left, right or we are passing by. We also adjust position of banks, restaurants, and other „buildings“ that are on the left or right. If they are close we specify that we are passing by them and when they are far away, we do not take them in the account at all (we do not need to know, that 170 meters to the left is a restaurant).

Then we proceed according to type of underlying element (node, way or relation).

3.4.1 Processing node

To process node we first find out, whether there are any turn-offs. From OSM we load list of ways associated with this element. For each way, we test whether it is the way, user is walking on. If we find such a way, we evaluate all other ways. For each of these ways we load list of their nodes. One of these nodes must be our processed node. We load coordinates of its predecessor and successor on this way (if they exist). We compute change of the course from users current course to course to next node on evaluated way. According to course we detect, whether this is turn-off to the left or right and generate corresponding message to inform user about the turn-off.

Next step in processing the node is to get its context with regards to ending of the segment. If this node is close to the ending (<20 meters), and it is railway crossing, highway crossing, subway entrance or steps, we will generate message, that at the end he has to walk through these (otherwise, the message to go pass them would be generated).

3.4.2 Processing a way

Way is passed further as it is (meaning, that it is only wrapped around by message object). One exception is that footways are processed to get additional information about them. OSM tags are parsed and we extract type of the surface and slope. If it is changed from the previous, we generated messages about this change.

3.4.3 Processing a relation

In current implementation, we are interested in only one type of relation and that is a tramway track. We simply wrap this type of element into message.

3.4.4 Adding additional messages

Between each pair of subsequent segments we add several types of messages. First, we evaluate change of course and add message instructing user to change course by certain amount of degrees. We add this message for any change in course, even for as small as one degree. Reason for this is that threshold for reporting this is a matter of preference and should be tweaked during testing to suit to most users. It is on mapping from messages to sentences to use this information as specified by user preferences. It has also to choose appropriate level of turning described by words (value in degrees is not very useful).

Next we add message to instruct user to continue. It contains information about how far in meters he should walk. We associate a suitable element that can act as a small landmark, specifying end of a segment such as crossing, steps, subway entrance, etc. In the future, we hope to identify street corner as an ending element. When this message is mapped to sentence, it can and should say to continue certain amount of distance until user walk to crossing or any other ending element.

Finally we add „pause“ message that is just saying that segment has ended. Because segment is logical, smallest unit of description and in this phase, original segments are completely gone and cannot be recreated, we add this message, so the mapping will be able to divide description into smaller logical parts. It can for example start new paragraph or insert few blank lines. User can stop reading after reaching this, follow read instructions and resume reading from this part. If segment was last, we do not generate pause message, but ending message so he can be informed that he reached his destination.

3.5 Mapping messages into sentences

Final step in description generation is mapping route messages into sentences. Many different mappings can be created, from simple one-to-one mapping to complex mappings which employ m-to-n relationship between messages and sentences. We implemented two mappings – first was experimental one-to-one mapping that soon proved to be too simple to be used in final product. Obvious reason for that is that each element which was user passing generated single sentence. As a result there was a lot (up to 5-6) same sentences with different word at the end. This does not sound natural at all. Second mapping employs many-to-one relationship. We call this mapping a grouped mapping.

3.5.1 Grouped mapping

The idea is to group all messages of same type (or ones that can fit into meaningful sentence) and generate single sentence from this group of messages effectively creating many-to-one relationship. This will lead to more natural sounding, simpler and readable description.

First we have to decide, where we want to start and end grouping messages. Because segments are not preserved in this phase, we only get one continuous collection of messages. If we grouped all messages of same type now, we would not be able to create meaningful description. Imagine we would inform user right in the beginning that he will encounter a number of places to his left and then tell him to turn right. He would not know whether he should expect another number of places, or whether he should have passed them already.

Logical approach would be to tell him what he is going to encounter until next instruction to turn or use the crossing (which is at the end of the segment). Luckily, we have inserted pause message at the end of each segment, so we can recreate sequence of messages for individual segments. So we only group messages until next pause message, then we output sentences and proceed to next sequence of messages.

Our algorithm starts by counting messages specifying turn-offs. We count turn-offs to the left and to the right separately until next pause message. In same loop we also aggregate information messages informing about what is going user to pass along his way. We store these elements separately in arrays for left, right and center. Using this, we can inform user about turn-offs in a single sentence, about landmarks on his left and right side in next sentence and we have prepared array of elements he is going to pass.

After finishing this grouping we output a sentence informing about what is to his left and right side. Then we process remaining messages sequentially and output single sentence per message. For list of all message types see table 3.2. This includes:

- Message about change of surface or slope
- Need to use crossing or steps or entering underpass
- Reaching the destination
- In case of pause message we end the paragraph and output two new lines.

When we encounter „continue until" message, we use our prepared array of grouped elements user is going to pass along his way. We enumerate these elements in the sentence, then specify how many turn-offs will be on each side and then instruct him to continue certain distance. If ending point is specified as discussed above, then we add this to the sentence.

This grouping produces very natural sounding text and because of its structure it eliminates most of pitfalls during construction of grammatically correct Czech sentence.

3.5.2 Czech grammar in sentences generation

Although it might seem difficult to construct grammatically correct sentences programmatically, we can make use of the way we create the sentences to eliminate possible error. Source of possible error can be in majority of cases either:

- Wrong grammatical case of noun is used
- Misuse of plural/singular
- Incorrect declension of nouns after numerals

First we take advantage of the fact that we do not add any user-added content. We take data from OpenStreetMap and use only use elements we can recognize. We do not use any user content. Therefore, all nouns added to final description are hard-coded in our program and we can take noun in a correct grammatical case. Second, we take advantage of the way our sentences are formed. We use either nominative case or accusative case. Nominative case is used in first informational sentence where we enumerate landmarks on left and right, so we select appropriate grammatical case of noun. Same applies for sentence where we specify landmarks to pass by, but we select accusative case. Misuse of plural or singular is minimized, because we do not specify number of occurrences of any landmark except turn-offs. In this case we hardcoded all possible variations. Declension after numerals can also occur in this case and is also solved by hardcoding all possibilities.

Now we have finished generating our description that is sent back to user. Our algorithm took care of creating appropriate route segments, aggregated largest amount data possible, classified it according to its relevance, evaluated context of individual pieces of data and generated natural sounding sentences in meaningful context.

The problem of generating human-like descriptions soon gets into condition called Uncanny Valley. The more it resembles real human written description, any minor imperfections become more apparent and disturbing.

Element type	**OSM tags**	**Usage**
RESTAURANT	„amenity“, „restaurant“	Restaurant, also often specifies name
BAR	„amenity“, „bar“	Bar, also often specifies name
RAILWAY_CROSSING	„railway“, „crossing“	Railway or tram crossing, more parameters like acoustic signalization is also specifed
HIGHWAY_CROSSING	„highway“, „crossing“	Highway crossing, also more parameters are usually specified
TRAM_STOP	„addr:railyway“, „tram_stop“	Tram stop with specified name and line number
SUBWAY_ENTRANCE	„railway“, „subway_entrance“	Subway entrance, also line is specified.
HAIRDRESSER	„shop“, „hairdresser“	Hairdresser
MEMORIAL	„historical“, „memorial“	Memorial of some type, name is specified
TRAVEL_AGENCY	„shop“, „travel_agency“	Travel agency
TOILET	„amenity“, „toilet“	Public toilets
BANK	„amenity“, „bank“	Bank, also often specifies name
ATM_MACHINE	„amenity“, „atm“	An ATM machine, bank is usually also specified
POST_BOX	„amenity“, „post_box“	Public post box
TELEPHONE	„amenity“, „telephone“	Telephone booth
AMENITY	„amenity“	Closely unspecified public place, extending for new tags might me needed to parse this.
SHOP	„amenity“, „shop“	Unspecified type of shop, sometimes specifies name.
HOUSE_NUMBER	„addr:housenumber“	Entrance to building with specified address
ISLAND	„place“, „island“	Island
PARKING	„amenity“, „parking“	Parking lot
STEPS	„highway“, „steps“	Steps, their number is also specified
FOOTWAY	„highway“, „footway“	Footpath for pedestrians
PARK	„leisure“, „park“	City park
FENCE	„barrier“, „fence“	Fence
GRASS	„landuse“, „village_green“	Grassed area
BUILDINGS	„highway“, „residential“	Buildings, used to detect wall as a guideline
STREET	„highway“, „primary“	Main street or avenue
TRAM	„route“, „tram“	Tramway track

Table 3.1: Route element types and their interpretation according to OSM XML tags.

Message type	**Usage**
ON_YOUR_LEFT	User should be informed that route element associated with this message is on his left.
ON_YOUR_RIGHT	User should be informed that route element associated with this message is on his right.
SURFACE_INFO	Underlying surface has changed, associated element specifies new surface.
SLOPE_INFO	Slope of the surface has changes, new slope is specified.
CHANGE_DIRECTION	User should change direction. Change of course is specified.
CONTINUE_UNTIL	User should go straight ahead until he reaches associated element. Distance is also specified.
USE_CROSSING	User should use crossing. Underlying route element provides more information about type of the crossing.
USE_STEPS	User should go down or up the steps, associated element might contain number of steps.
PASSING_BY	Information about map features that user is going to pass along his way. Usually follows after CONTINUE_UNTIL message to specify more accurately features along the route.
INFO_END	User should be notified that he reached his destination.
USE_UNDERPASS	Same as steps, but specifies underpass or metro entrance.
TURNOFF	Information about the turn-off on the route. Similiar to PASSING_BY message, but also specifiec course of the turn-off.
PAUSE	This message is included at the end of each segment, so user should no proceed according to given information and resume reading after that.

Table 3.2: Route message types and their meaning. Each message is supposed to provide a piece of information to the user.

Chapter 4

Testing

Generated description of route was evaluated by professional writers of such descriptions in navigation center [11] of SONS organization [26] (further in text referred as operators). Operators of this navigation center are target users of our system. For current description creation process see chapter 1.1. Our system is intended to generate text for them (doing most of the work), so they can edit it as needed and present it to user. To evaluate quality and usability of generated descriptions was primary goal of our usability tests. These tests were focused to identify faults, imperfections, wrong stylistics, and incompleteness of generated descriptions. Given this, our primary concern is content of the text, after that form of the text and then anything else (like usability of our browser applet, etc.). With this in mind, we designed several experiments to collect as many findings as possible. With limited number of target users available we focused to qualitative tests and expert reviews by open ended questions rather than collecting quantitative, structured data.

4.1 Test setup

We wanted testing to be as close to real-life work as possible. Tests took place in SONS navigation center, directly on operator's workplace. Testing environment familiar to operators allowed them to focus fully on given testing tasks. Furthermore, they used their own computers so they could use their favorite web browser and other software they might use while creating route descriptions. Another reason for using their computers is that our application is deployed on server computer situated in SONS building and is connected to their network. While using server from outside of the network is possible, it causes lags and even lacks response in extreme cases. Communication form intranet does not suffer from these problems so using their computers is very beneficial.

Testing was split in 2 sessions. During the first session we collected information and used it to improve our algorithm, fix errors and add new features. Second session was intended to evaluate these new features and to suggest improvements for future work.

4.2 Testing sessions

4.2.1 Preparations

Test sessions required 2 operators, for each we wanted at least 40 minutes time. Three test cases were prepared (one primary and two secondary tests). Primary test was to be given to both operators, each of secondary tests to only one operator.

4.2.2 Test cases

4.2.2.1 Test case 1

This test is designed to find out, how generated description is different from one created by human and specifically enumerate and describe these differences.

Test assignment:

1. Create short (<300 meters) description of route by usual way (or use existing one you have created).
2. Plan the same route on the map in our application and let it generate description
3. What important differences do you see between the two descriptions?
4. Are there some differences that make one description superior to another? Explain why.

4.2.2.2 Test case 2

Second test on the other hand is intended to collect changes needed to be made to generated description in order to make it usable for end user. In this case, participant has no human created description of the route, so he has no reference description for comparison and he has to work solely with generated text.

Test assignment:

1. Plan some arbitrary but real-life (no cycle, zigzag walk, etc...) route on map in our application. It should not be very long. Generate the description by our application.
2. Now edit the generated text so it is understandable and ready for end user.
3. Explain what edits you have made and why.

4.2.2.3 Test case 3

Purpose of this test is to get quick summary of problems in generated description by expert review. Because this is the last test (probably only few minutes left for testing), participant does not have to edit the description, just report problems he sees. This test also gives participant chance to work with application unbounded and to give us his final thoughts about program.

Test assignment:

1. Plan some arbitrary but reasonable route on map in our application. It should not be very long, but it should contain problematic parts of route (steps, underpasses ...). Generate the description by our application.

2. Study the generated description. You do not have to edit it now.

3. Tell us about problems you see in the description. Be as much descriptive and specific as you can.

4.2.3 Tests execution

4.2.3.1 First session

We had two participants in the first session. Participant one was an experienced operator of navigation center. He was to some extent familiar with our system from previous testing conducted on earlier prototypes that took place 3 years ago. This was very helpful, because he knew how our system works and needed no introduction to its usage and inner workings. Participant two was newly employed operator, only one week on the workplace. Despite this, his independent look provided some helpful insights and showed us, how our application is received by beginner in the field.

Participant one was given test cases 1 and 2. For test case 1 he managed to use existing description fragment. Planning the same route was unfortunately not possible, because automatic path finder always chose another way and currently our application did not provide option to turn it completely off. Participant managed to plan fairly similar route (see figure 4.1) (note that English transcriptions were done manually, not by our application) and this posed no major problem further. Participant had several comments to route differences, but in general he accepted generated description as very accurate. Notice only very few differences in human created and machine generated description of this route (except the ending, where route was differently planned). One major difference was that human description contained changes in direction stated in degrees, our description contained only „binary“ change of direction. This is subject for discussion, because behavior of our application was changed from reporting degrees to simpler directions after previous testing. Although this description contains direction changes in degrees, all participants (also from previous testing) agreed, that this is not ideal. This participant suggested that current behavior should differentiate at least two levels of turning. Another insight he provided was that system should report turn-offs which user passes along the way. By counting turn offs passed he is more likely to stay on right way.

For second test case he used route show on figure 4.4. He stated that this route could be used with only minor edits directly. His final comments to application were that he is looking forward to use this application and he is sure that it would be helpful. He thinks that even in current form, he would use it to generate skeleton of descriptions which he would edit to finalize descriptions. He praised newly introduced segmentation feature very much, saying that it fits human created descriptions almost completely. The major problem he sees is lack of data in map and low coverage of city by footpaths. Findings and suggestions are summarized below.

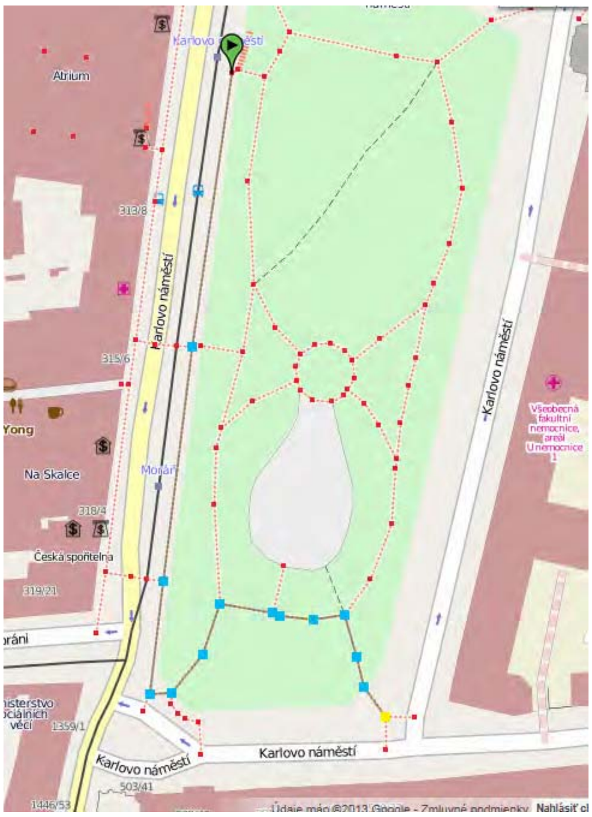

Figure 4.1: Route for test 1.

Quotes:

☺ Segmentation works great, it needs no adjustment whatsoever.

☺ I can use this description as a skeleton and do only minor edits.

☺ Automatic distances measuring load off a lot of work from me.

☺ Its great that it reports restaurants and other places by names. If they know them, it helps to. orientate.

☹ Map could provide more data about this crossing and street paving.

☹ Seems like only small area is covered by footpaths.

☹ At least two levels of turning should be distinguished, one is not enough.

Second participant was given only third test case, because of his lower experience and less time available. He planned a route from bank on Charles place to closest subway entrance as shown in figure 4.7. He has two comments on generated description. First regarded names of two restaurants listed in the description. They usually do not list names of such places, but provide them only when asked to do so. This can be subject of end user preferences

```
Nalevo od vás je park. Napravo od vás je zastávka tramvají Moráň,
provoz tramvají. Pokračujte rovně 200.0 metrů.

Otočte se vlevo. Pokračujte rovně 7.0 metrů.

Otočte se vlevo. Pokračujte rovně 15.0 metrů.

Pokračujte rovně 15.0 metrů.

Nalevo od vás je park. Napravo od vás je park. Otočte se vpravo.
Pokračujte rovně 40.0 metrů.

Nalevo od vás je park. Napravo od vás je park. Otočte se vpravo.
Pokračujte rovně 25.0 metrů.

Pokračujte rovně 10.0 metrů.

Jste na místě.
```

Figure 4.2: Description for test 1 (in Czech)

and could be adjusted accordingly. Second comment was about traffic island in the middle of the street surrounded by two crossings. He remarked that user should be notified about the island, because when user is said to use crossing, he would probably go through both crossings without stopping on the island. Because generated description says „use crossing“ twice, he would probably start looking for second crossing after he already crossed both. This could cause confusion so traffic islands have to be accounted for. He also noticed that report about tramways is repeated several times (in each segment). This is not needed and can be only reported once. Possibility of reporting „end of running tramways“ is to be discussed in future testing.

4.2.3.2 Second session

For second session, only one participant was available. It was the same as participant 1 in previous session. Because we could spent more time with single participant we agreed to do test case 3 three times with deeper analysis of generated descriptions by him. Since he is expert in the field we gained more insight into how he creates descriptions compared to machine generated descriptions.

In the first case he used route on figure 4.10 and generated description listed in 4.12. There was detected a problem that order to turn is completely missing. Operator checked the feature to distinguish more levels of turning in this description. He was satisfied with changes made according to discussion in previous session. Next he focused on checking another new feature – reporting of turn-offs. He used route and description on figure 4.13 and listing 4.15. He was also happy to see this new feature and recommended further improvement - to report only turn-offs on the side the user is going on.

```
On your left is park. On your right is tram stop Moráň,
tramway tracks. Continue straight ahead 200 meters.

Turn left. Continue straight ahead 7 meters.

Turn left. Continue straight ahead 15 meters.

Continue straight ahead 15 meters.

On your left is park. On your right is park. Turn right.
Continue straight ahead 40 meters.

On your left is park. On your right is park. Turn right.
Continue straight ahead 25 meters.

Continue straight ahead 10 meters.

You have reached your destination.
```

Figure 4.3: Description for test 1 (in English)

In the third case he moved to different part of the city than he used in all former tests. Here he evaluated route from metro station to KFC restaurant (see figure 4.16). In this test, he was also satisfied with generated description, but reported two issues – wrong rounding of distances and inappropriate position of tram track (it was reported to be on both sides to the left and right, but navigator expected it to be in front of him). In overall, tester was satisfied with current working of our system, but suggested more features that would make it even more usable. He discovered some bugs that are reported below.

Results are summarized and explained in detail below, just after the results of first session.

Quotes:

☺ I like turn-offs counting.

☺ Its great that whole city is now available.

☺ As a skeleton of description this is really great, I might let it be like this and only add. information on demand.

☹ Turn is missing here, this is wrong.

☹ Seems like only small area is covered by footpaths.

☹ System could work faster, this takes a long time.

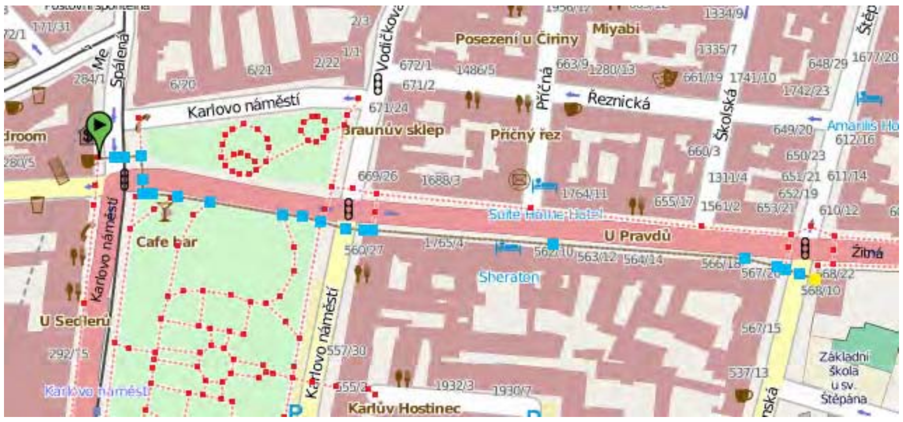

Figure 4.4: Route for test 2.

4.2.4 Findings and suggestions

In this section we summarize findings, analyze them and suggest solutions. First four findings are related to description generation system. These will be fixed and implemented before second testing session and will be subject to testing session afterwards. Findings 5 and 6 are related to other parts of system and are beyond the scope of this work. It is suggested to fix this in the future work. Rests of the findings were reported during second test session and since this was last development iteration, fixing them is also left for the future work.

Finding 1 – Turning levels First finding is regarding turns. Currently our application does not differentiate levels of turns and does not specify „how much“ to turn. This is very complex problem for visually impaired users and is subject to changes since first prototype application. After last testing session this solution is suggested. Turn is reported when direction changes by more than 30 degrees or when T-junction is reached. Instruction to change direction will have two levels. First level will be applied to change in course by 30-50 degrees, second level for more. We will rely on guide lines for keeping user on correct course and will report them if map provides such data.

Finding 2 – Turn-offs reporting Another problem we encountered during generated description reviews was that no turn-offs that user passes on his way were reported. For visually impaired using white cane this could be important mean for knowing how far he had gone. Using white cane, he can easily find such turn-offs and count them during his way. Because count can be easily read form our map, we should list this count in generated description. This way, our user gains more useful information and he can use it to walk correct distance confidently.

Finding 3 – Traffic islands Traffic island is also very easily identified by visually impaired because they are often equipped with tactile markings. Our application currently lacks reporting of this landmark and as testing results show, it brings confusion to description

```
Nalevo od vás je provoz tramvají. Napravo od vás je provoz tramvají.
Pokračujte rovně 15.0 metrů. Potom přejděte přes přechod.

Pokračujte rovně 10.0 metrů.

Nalevo od vás je park. Napravo od vás je provoz tramvají. Otočte se
 vpravo. Pokračujte rovně 20.0 metrů. Potom přejděte přes přechod.

Napravo od vás je park, stěna domu. Otočte se vlevo. Pokračujte
rovně 120.0 metrů. Potom přejděte přes přechod.

Nalevo od vás je stěna domu. Napravo od vás je stěna domu. Přejděte
přes přechodPokračujte rovně 250.0 metrů.

Jste na místě.
```

Figure 4.5: Description for test 2 (in Czech)

because of this. Problem explained earlier can be easily solved by telling user that he will reach traffic island after crossing. It should be made clear that next instructions for user will be said as if user was still on the traffic island, not after the second crossing (traffic island always has at least two crossings and some users might take this for certain and go through both crossings at once, but our application reports these crossings separately).

Finding 4 – Repetitive instructions When giving information about what is on user's left and right, we repeat in each segment all information again. Sometimes it might be redundant to inform him about things that has not changed. On the other hand, telling him that e.g. tramway track is still on his left can assure him about correct heading. This has to be tested more deeply and we suggest to parametrize this feature, so it can be set from GUI whether to repeat or not this type of information (preferably by filtering certain type of elements).

Finding 5 – Lack of data Participants were complaining about lack of data in the map (not in descriptions). Description generation was changed earlier from subtractive to additive system. This means that in previous prototype was reported everything that map contained except data that could not be decoded (unknown XML tags). As an answer to previous testing this was considered unsatisfactory and new system was designed that worked according to additive approach. Only recognized landmarks and map features are added and always in context when it is helpful. Lack of footpaths in map is a result of insufficient data provided by OSM system. Data to this system is added every day and grows very quickly, therefore system for importing data to our system should be designed and implemented. Automatic synchronization with OSM server would be ideal solution, but this was not main purpose of this work, so it will be left for future work.

```
On your left are tramway tracks. On your right are tramway tracks.
Continue straight ahead 15 meters. Then use the crossing.

Continue straight ahead 10 meters.

On your left is park. On your right are tramway tracks Turn right.
Continue straight ahead 20 meters. Then use the crossing.

On your right is park, wall of a building. Turn left. Continue straight
ahead 120 meters. Then use the crossing.

On your left is wall of a building. On your right is wall of a building.
Use the crossing. Continue straight ahead 250 meters.

You have reached your destination.
```

Figure 4.6: Description for test 2 (in English)

Finding 6 – Inability to plan wanted route This issue is related to automatic routing system that chooses best route possible according to given preferences. It tries to use footpath instead of steps, controlled crossing instead of uncontrolled etc. Although adjustment of these preferences is implemented on server, it is not supported by browser client. This led to inability to plan wanted route, because automatic routing system considered such route as worse than another possible. There was no way of turning this off in client application, so user had to use route that our application considered to be better (even if this is not always right).

Finding 7 – Missing turn command During second test session occurred a problem that command to turn to left was completely missing. There is also missing the order to turn left. This could be erroneous design or bug in the implementation. Since messages to turn are generated automatically at the end of all segments and only taken into account when change of course (supplied as a parameter) is greater than specified threshold – it is more probable, that sentences generator handles this change of course incorrectly and needs to be fixed. In this instance, the course change is greater than 90 degrees (approximately 110 degrees), so this is value that causes error to occur and has to be tested primarily

Finding 8 – Orientation on the beginning of route Another problem reported by operator was that the system did not take into account that if navigated person starts on the crossroad, system cannot know from which direction the person comes and chooses the direction pseudo randomly. Operators wish that they could specify from which direction the user is coming if this is possible. In previous work this was solved by putting message „Route starts on the crossroad, you need to orientate navigated person“, but since it is technically possible to solve programmatically, operators want this feature implemented. Suggested solution is to enable adding one additional point in GUI that creates oriented line with starting point and by turning it around it would be possible to specify incoming direction.

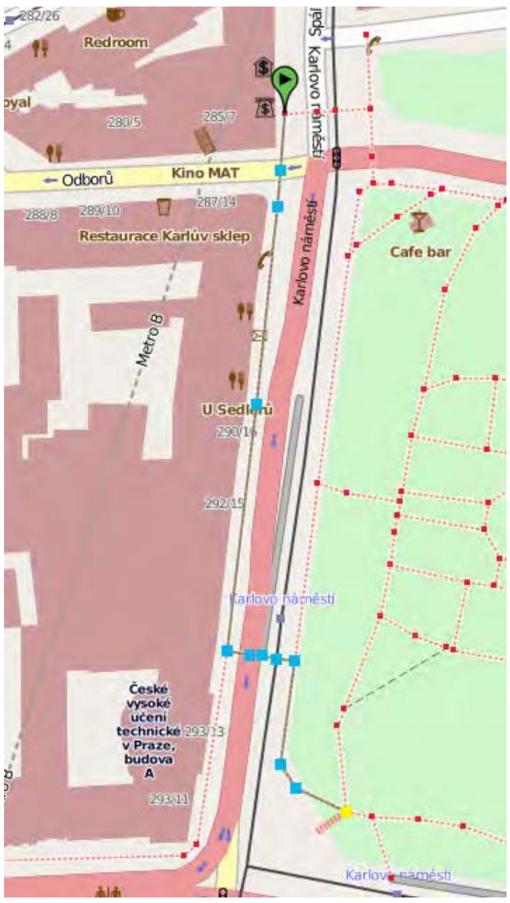

Figure 4.7: Route for test 3.

Finding 9 – Footpath side recognition When user walks on the street he usually sticks to some guideline, he does not go in the middle of the footpath. Guideline can be wall, curb, special tactile marking etc. This means that he chooses to go on one side and sticks to it. In our system we are reporting turn-offs on both sides of the road even if he can only feel turn-offs on the side he is going on (by his white cane). It would be convenient if system could analyze the segment of route and recommend user which side to use and then only report turn-offs (and possibly other features) on this side. By the rule „less can be more“ this lowers his cognitive load by providing less but more relevant information.

Finding 10 – Map elements in front of navigated person Current implementation of system does not support reporting some map feature (e.g tram tracks) in front of navigated. In such case it report it as it is both left and right. In situation that occurred during test 6, navigator stated that he would not formulate the description as our system at all. He would report that tracks are in front of him. This can be solved by implementing system that would detect these cases and according to distance to map feature it would report it as „in front“ or it would not report it at all.

```
Napravo od vás je stěna domu. Pokračujte rovně 25.0 metrů.
Potom přejděte přes přechod.

Nalevo od vás je provoz tramvají. Napravo od vás je stěna domu.
Pokračujte rovně 120.0 metrů. Cestou projdete kolem restaurace
U Sedlerů, restaurace Čínský bufet.

Otočte se vlevo. Pokračujte rovně 10.0 metrů. Potom přejděte
přes přechod.

Nalevo od vás je provoz tramvají. Napravo od vás je provoz
tramvají. Pokračujte rovně 9.0 metrů. Potom přejděte přes přechod
přes koleje.

Otočte se vpravo. Pokračujte rovně 30.0 metrů.

Otočte se vlevo. Pokračujte rovně 8.0 metrů.

Pokračujte rovně 15.0 metrů.

Jste na místě.
```

Figure 4.8: Description for test 3 (in Czech)

Finding 11 – Rounding distances Operators noticed that distances are always rounded to nearest fives or tens. This is true for numbers greater than 10 (rounded to fives) and greater than 100 (rounded to tens). They complained that with regards to small distances (20-30) this is very rough and should not be rounded so drastically. This can be solved very easily by tweaking parameters that specify rounding. Before it could be done, there has to be some experiments and testing to get ideal rounding that would satisfy most users and maybe provide GUI controls to tweak these parameters.

Finding 12 – Extra starting point in some browsers There is a problem with Javascript API to manipulate maps provided by Google does not behave consistently in all browsers when selecting point on map by clicking. It is expected to find nearest point to the position where user clicked and select this point. Behavior in some browsers is this: it creates point exactly where user clicked, then find nearest and connects them by line. This results in very small segment that is handled as part of route and introduces confusion to description, because it often contains senseless command to turn. Furthermore, this segment can be so small that is almost invisible on screen to it cannot be found and corrected easily. This problem occurs in current version of Opera and Firefox. In Chrome and IE this works as intended.

Finding 13 – Slow description generation Description takes some time to generate – depending on length up to 2-3 minutes in average. This is caused by large number of

```
On your right is wall of a building. Continue straight ahead 25 meters.
Then use the crossing.

On your left are tramway tracks. On your right is wall of a building.
Continue straight ahead 120 meters. You will pass restaurant
U Sedlerů, restaurant Čínský bufet.

Turn left. Continue straight ahead 10 meters. Then use the crossing.

On your left are tramway tracks. On your right are tramway tracks.
Continue straight ahead 9 meters. The use the railway crossing.

Turn right. Continue straight ahead 30 meters.

Turn left. Continue straight ahead 8 meters.

Continue straight ahead 15 meters.

You have reached your destination.
```

Figure 4.9: Description for test 3 (in English)

queries to OSM API called through LibOSM. To generate a description there are dozens or hundreds of queries. Each query is submitted using HTTP protocol to REST API. Depending on speed and latency of network this causes unwanted slowdown. In case of average ping 100 ms, 200 queries will take at least 20 seconds to answer not counting time needed to create answers from internal database and to process them by our system. From this it is clear, that speed of system is limited by network throughput and HW capabilities. Our system runs simultaneously very demanding services like 2 relational databases, Glassfish application server, Ruby on Rails applications, Apache HTTP server and others. For the future, this has to be considered and available HW has to be effectively utilized.

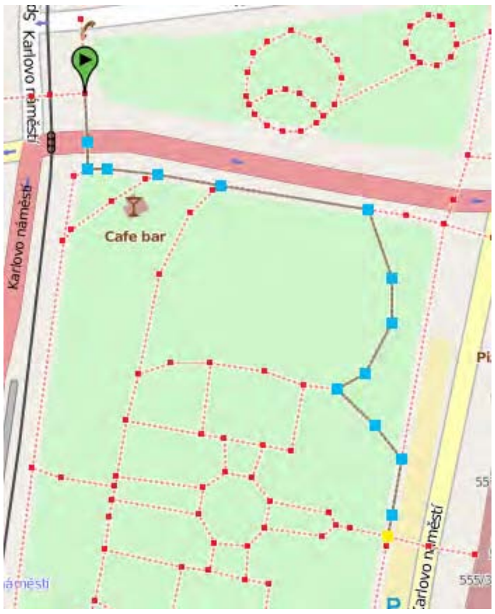

Figure 4.10: Route for test 4.

```
Nalevo od vás je park. Napravo od vás je provoz tramvají.
Pokračujte rovně 20.0 metrů. Potom přejděte přes přechod.

Napravo od vás je park, stěna domu. Otočte se vlevo. Pokračujte
rovně 75.0 metrů.

Otočte se vpravo. Pokračujte rovně 20.0 metrů.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 10.0 metrů.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 15.0 metrů.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 9.0 metrů.

Nalevo od vás je park, parkoviště. Napravo od vás je park. Pokračujte
rovně 25.0 metrů.

Nalevo od vás je parkoviště. Napravo od vás je park. Otočte se vpravo.
Pokračujte rovně 20.0 metrů.

Jste na místě.
```

Figure 4.11: Description for test 4 (in Czech)

```
On your left is park. On your right are tramway tracks. Continue straight
ahead 20 meters. Then use the crossing.

On you right is park, wall of a building. Turn left.Continue straight
ahead 75 meters.

Turn right. Continue straight ahead 20 meters.

Turn right slightly (45 degrees). Continue straight ahead 10 meters.

Turn right slightly (45 degrees). Continue straight ahead 15 meters.

Turn right slightly (45 degrees). Continue straight ahead 9 meters.

On your left is park, parking lot. On your right is park. Continue straight
ahead 25 meters.

On your left is parking lot. On your right is park. Turn right. Continue
straight ahead 20 meters.

You have reached your destination.
```

Figure 4.12: Description for test 4 (in English)

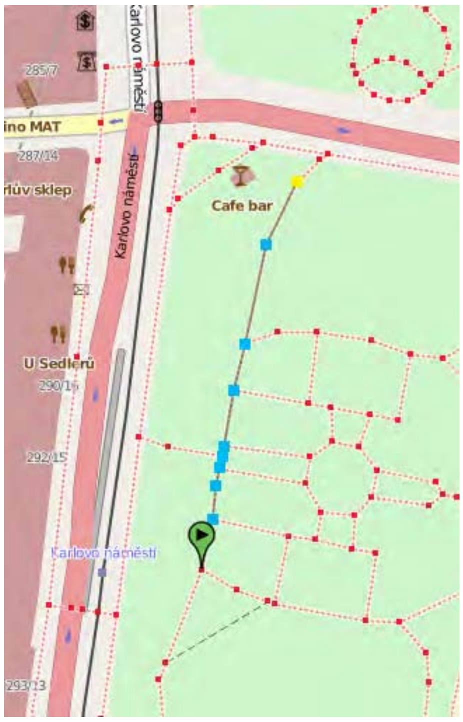

Figure 4.13: Route for test 5.

```
Nalevo od vás je park, stěna domu. Napravo od vás je park. Pokračujte
rovně 110.0 metrů. Minete 1 odbočku vlevo a 5 odboček vpravo.

Jste na místě.
```

Figure 4.14: Description for test 5 (in Czech)

```
On your left is park, wall of a building. On your right is park. Continue straight
ahead 110 meters. You will pass one turn-off left and 5 turn-offs to the right.

You have reached your destination.
```

Figure 4.15: Description for test 5 (in English)

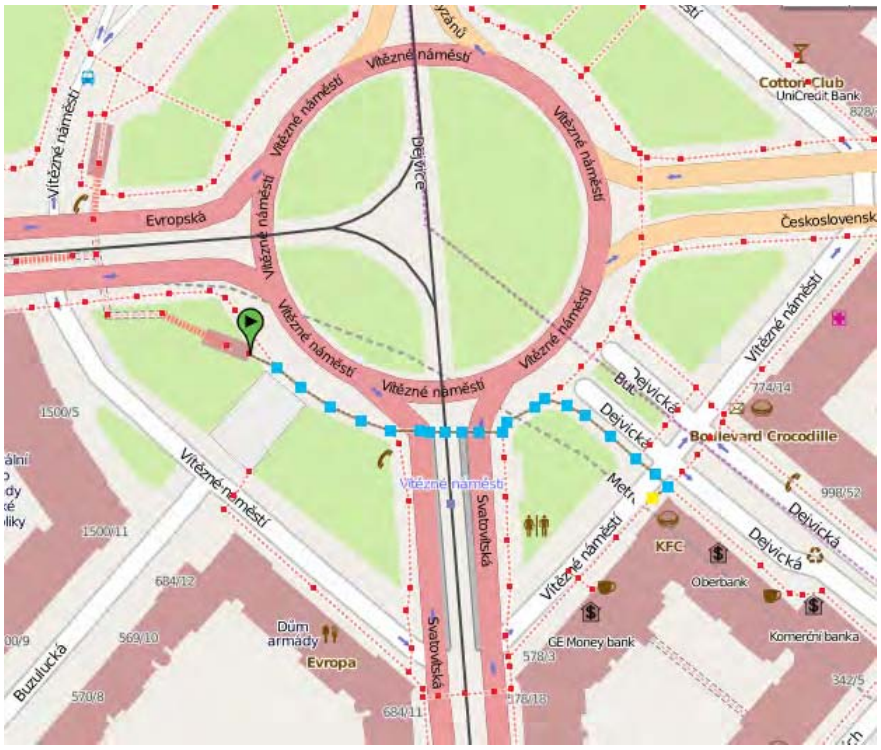

Figure 4.16: Route for test 6.

```
Nalevo od vás je trávnatá plocha, provoz tramvají, stěna domu. Napravo
od vás je trávnatá plocha, stěna domu. Pokračujte rovně 55.0 metrů.
Minete 1 odbočku vpravo. Potom přejděte přes přechod.

Nalevo od vás je provoz tramvají. Napravo od vás je provoz tramvají.
Pokračujte rovně 9.0 metrů. Potom přejděte přes přechod přes koleje.

Napravo od vás je provoz tramvají. Pokračujte rovně 10.0 metrů. Potom
přejděte přes přechod.

Otočte se vlevo. Pokračujte rovně 3.0 metrů.

Napravo od vás je trávnatá plocha. Otočte se mírně vpravo (45 stupňů).
Pokračujte rovně 10.0 metrů.

Napravo od vás je trávnatá plocha. Otočte se mírně vpravo (45 stupňů).
Pokračujte rovně 20.0 metrů.

Napravo od vás je trávnatá plocha. Pokračujte rovně 20.0 metrů.
Minete 2 odboček vlevo. Potom přejděte přes přechod.

Pokračujte rovně 5.0 metrů.

Jste na místě.
```

Figure 4.17: Description for test 6 (in Czech)

```
On your left is grassed area, tramway tracks, wall of a building.
On your right is grassed area. Continue straight ahead 55 meters.
You will pass 1 turn-off right. Then use the crossing.

On your left are tramway tracks. On your right are tramway tracks.
Continue straight ahead 9 meters. The use the railroad crossing.

On your right are tramway tracks. Continue straight ahead 10 meters.
Then use the crossing.

Turn left. Continue straight ahead 3 meters.

On your right is grassed area. Turn right slightly (45 degrees).
Continue straight ahead 10 meters.

On your right is grassed area. Turn right slightly (45 degrees).
Continue straight ahead 20 meters.

On your right is grassed area. Continue straight ahead 20 meters.
You will pass 2 turn-offs left. Then use the crossing.

Continue straight ahead 5 meters.

You have reached your destination.
```

Figure 4.18: Description for test 6 (in English)

Chapter 5

Conclusion

In this work, we designed and implemented system for generating route description suitable for visually impaired people navigating in urban environment. We successfully accomplished a goal of designing and implementing new approach to description generation. We tested our implementation with end-users of this system who are operators of navigation center of SONS organization.

Result of this work is functional prototype deployed in SONS organization that is usable and ready for further development. Improvements involve newly designed and tested description generator which takes into account need for natural sounding language, guidelines needed for visually impaired and provides adequate amount of information so it is not overwhelming while still being sufficient for safe navigation. It was developed with cooperation of SONS operators who are experts in the field of navigating visually impaired and their suggestions were implemented in prototype application.

Final implementation provides means for planning the route on the map so it is convenient to walk for visually impaired and generates description of this route so it is as close to human created description as possible. This generated description can be used as skeleton for navigator so he only needs to do minor improvements before presenting this description to blind user, or it can be used directly. This loads off a lot of work from navigators which can handle more requests and provide faster access to route descriptions for visually impaired.

5.1 Further development

There are several ways in which can this application further evolve. First way, probably most important is to fill application data source with more information and/or provide more detailed information. Second, there is the need for convenient and user-tested GUI which could effectively utilize API provided by our server application. Third way for improve our system is to enhance description generator.

5.1.1 Data source improvement

Currently, there are two major issues with our data source -– it is relatively slow (on current hardware) and it is not detailed enough.

Speed could be improved simple by using more powerful hardware. However there are other less costly ways that could have more advantages than boosting performance. If we were able to utilize main OSM server that is publicly available, we would vastly gain performance. Our server would not have to run MySQL database and Ruby on Rails application and could use spare hardware resources to speed up other applications. This would introduce problem of keeping our GIS database in sync with main OSM server. Because map is constantly updated, desynchronization would occur on daily bases. We need our GIS database to answer various spatial queries that OSM API is not able to answer, so we definitely need our GIS database. Solution might be to regularly download dataset from OSM API and import it to our GIS database. Only problem is that dataset for the city of Prague has hundreds of megabytes and is still growing and is therefore very slow to import. This could be solved by parsing XML changesets to get only latest changes and import these to our GIS database. Second problem is that we need to keep our slippy map update as well. On current HW it takes approximately 18 hours to generate complete slippy map for whole city, so it is out of question to do it daily. This could be also solved by updating only changed parts – and because our slippy map does not show all possible map features, we can update only if showed map features have changed.

Implementing this, we would get significant performance boost, constant updates to our map and possibility to support also other cities. Other cities would have to be imported to our GIS database, but this database does not store all information, only topology and references to OSM elements in their database, so it is much smaller than original OSM database.

Another improvement is to support more types of footpaths that navigated user can walk on. Currently we do not support all OSM tags representing footpaths, because OSM tags are not standardized and it depends on each map creator what tags he uses. This leads to problem that not all possible paths are showed and clickable for planning route. This has to be solved before system is fully functional. There has to be also support for underpasses, metro station entrances etc.

5.1.2 Improving GUI

Another possibility to improve this work is to create user friendly and GUI. It should be tested for usability and for functioning in all major browsers. According to testing (chapter 4) there are several findings that keep current GUI from fully utilize possibilities of application. Look of the GUI is very prototype-like and there is plenty of space for improvements.

Troubles with inconsistent behavior in different browsers should be solved. Next, possibility to turn off automatic shortest path finding should be implemented. Because current data source does not contain all footpaths, it has tendency to go around paths that exist, but are not in his database and there is not possibility to select desired way. Another problem is that slippy map is shown on top of existing google maps, but if our layer was transparent, we could render satellite images and on top our footpaths. This could help operators to see map features that are not in our database, but are visible on google map and still see also clickable footpaths. Because generation of description takes some time, it would be convenient to see progress or at least message, that application is working and generation has not failed. Same applies for shortest path finding. Path finding also supports setting preferences. User can prefer not to walk along tram tracks, not to cross road without traffic lights etc. System for specifying these preferences in user friendly way is also a good way to improve GUI.

5.1.3 Evolution of description generator

According to testing there is plenty of findings with suggestions that could be improved in description generator. These mostly regard existing functionality that needs to be tweaked or fixed. New functionality should be implemented, mainly possibility to navigate users through underpasses and in metro entrances. This was not implemented because current configuration does not provide this type of information from data source. After improvements to data source suggested above are implemented, description generator should be adjusted accordingly.

Another way of improving description generator is to store intermediate result of generation process in a database as XML data as described in chapter 2.2.2.2. In this way, segments of map need to be processed only once and then this semi-product can be used again without need to call OSM API and our GIS database. It should be much more efficient and faster. Advantage of this would also be that text could be generated again quickly with different preferences, another language or after minor change to route path. Note that these stored intermediate results also need to be updated after syncing with main OSM database.

It will also be needed to extend generator to recognize more OSM XML tags. Set of these tags is still growing and it will be desirable to add new tags after extending the data source.

Bibliography

[1] Corduroy warning sheet.
`http://www.accesscode.info/external/5_6a.htm`, cited 24.4.2013.

[2] The Geographic Coordinate System.
`http://www.nrri.umn.edu/worms/downloads/team/TheGeographicCoordinateSystem.pdf`, cited 20.2.2013.

[3] Guidance on the use of tactile paving surfaces.
`https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/3622/tactile-pavement.pdf`, cited 23.4.2013.

[4] The guide horse foundation website.
`http://www.guidehorse.org/` , cited 25.4.2013.

[5] Guidelines and standards for tactile graphics.
`http://brailleauthority.org/tg/web-manual/`, cited 27.4.2013.

[6] How to use a white tipped cane.
`http://www.wikihow.com/Use-a-White-Tipped-Cane`, cited 25.4.2013.

[7] JOSM Application website.
`http://josm.openstreetmap.de/`, cited 28.4.2013.

[8] The LibOSM Project website.
`http://sourceforge.net/apps/mediawiki/travelingsales/index.php?title=LibOSM`, cited 25.11.2012.

[9] List of map features tags of OSM Project.
`http://wiki.openstreetmap.org/wiki/Map_features`, cited 29.4.2013.

[10] Mapnik Application website.
`http://mapnik.org/`, cited 23.11.2012.

[11] Navigation center of SONS.
`http://navigace.sons.cz/navigacni-centrum.html`, In Czech.

[12] Open Street Map import data formats.
`http://wiki.openstreetmap.org/wiki/Converting_map_data_between_formats`, cited 24.2.2013.

[13] OpenStreetMap Main Wiki Page.
`http://wiki.openstreetmap.org/wiki/Main_Page`, cited 28.4.2013.

[14] Osm2pgsql Application website.
`http://wiki.openstreetmap.org/wiki/Osm2pgsql`, cited 23.11.2012.

[15] Osmosis Application website.
`http://wiki.openstreetmap.org/wiki/Osmosis`, cited 23.11.2012.

[16] Passenger Application Server website.
`https://www.phusionpassenger.com/`, cited 29.4.2013.

[17] PostGIS 2.0 Manual.
`http://postgis.net/stuff/postgis-2.0.pdf`, cited 25.2.2013.

[18] PostGIS Project website.
`http://postgis.net/`, cited 25.2.2013.

[19] Potlach Application website.
`http://wiki.openstreetmap.org/wiki/Potlatch_2`, cited 28.4.2013.

[20] The primary visual cortex.
`http://www.cs.utexas.edu/users/nn/web-pubs/sirosh/pvc.html`, cited 28.4.2013.

[21] The Rails Port Website and Installation Guide.
`http://wiki.openstreetmap.org/wiki/The_Rails_Port#Installing_Rails`, cited 23.11.2012.

[22] Tactile paving wiki page.
`http://en.wikipedia.org/wiki/Tactile_paving`, cited 23.4.2013.

[23] Teaching blind children to navigate.
`http://www.wayfinding.net/navigate.htm`, cited 24.4.2013.

[24] TMAP project website.
`http://www.ski.org/Rehab/TMAP/`, cited 27.4.2013.

[25] Vacuum forming.
`http://en.wikipedia.org/wiki/Vacuum_forming`, cited 29.4.2013.

[26] Web page of SONS organization.
`http://www.sons.cz/index.php`, In Czech.

[27] White cane wiki page.
`http://en.wikipedia.org/wiki/White_cane`, cited 24.4.2013.

[28] D. Baldwin. Wayfinding technology: A road map to the future. *Journal of Visual Impairment And Blindness*, 97(10):1–19, 2003.

[29] J. Bokšanský. Systém pro automatické generování popisu cesty pro zrakově postižené, 2011. In Slovak.

[30] D. Bolgiano. A laser cane for the blind. *IEEE Journal of Quantum Electronics*, 3(6):268, 1967.

[31] C. Q. Choi. Blind People Have Superior Memory Skills, 2007. `http://www.livescience.com/4503-blind-people-superior-memory-skills.html`, cited 28.4.2013.

[32] R. Dale, S. Geldof, and J.-P. Prost. Using natural language generation in automatic route description. *Journal of Research and Practice in Information Technology*, 37(1):438–443, 2005.

[33] M.-P. Daniel, E. Przytula, and M. Denis. Spoken versus written route directions. *Cognitive Processing*, 10(2):1–19, 2009.

[34] M. A. Espinosa, S. Ungar, E. Ochaita, M. Blades, and C. Spencer. Comparing methods for introducing blind and visually impaired people to unfamiliar urban environments. *Journal of Environmental Psychology*, 18(3):277–287, 1998.

[35] T. Harris. How guide dogs work. `http://science.howstuffworks.com/zoology/mammals/guide-dog.htm`, cited 25.4.2013.

[36] R. D. Jacobson. Navigating maps with little or no sight: An audio-tactile approach, 1998.

[37] A. Klippel, K.-F. Richter, and S. Hansen. Cognitively ergonomic route directions. *Handbook of Research on Geoinformatics*, 2009.

[38] J. Lehrer. Navigating cities, for the blind, 2011. `http://www.wired.com/magazine/2011/09/st_qafolska/` , cited 18.4.2013.

[39] M. Loo-Morrey. Tactile paving survey, 2005. `http://www.hse.gov.uk/research/hsl_pdf/2005/hsl0507.pdf`, cited 22.4.2013.

[40] J. Lu. A comparative study of tactile paving design standards in different countries. *Computer-Aided Industrial Design and Conceptual Design*, pages 753–758, 2008.

[41] M. Maurer. White cane safety day: A symbol of independence. `https://nfb.org/white-cane-safety-day`, cited 24.4.2013.

[42] C. Meneghetti, F. Pazzaglia, and R. D. Beni. Spatial mental representations derived from survey and route descriptions: When individuals prefer extrinsic frame of reference. *Learning and Individual Differences*, 21(2):150–157, 2011.

[43] Y. Murai. Characteristics of human route descriptions and their improvement by machine landmark extraction. *3rd International Conference on Awareness Science and Technology*, pages 438–443, 2011.

[44] A. Nichols. Why use the long white cane?, 1995. `http://web.archive.org/web/20100330050804/http://www.blind.net/g42w0001.htm`, cited 24.4.2013.

[45] O. Procházka. Systém pro tvorbu popisu cesty pro zrakově postižené, 2010. In Czech.

[46] K. Roberts. The history of guide dogs in britain, 2004.
`https://d3qkb2hv043xyy.cloudfront.net/fileadmin/gdmain/user/About_us/History/Documents/AboutUs_historyofGuideDogs.doc`, cited 25.4.2013.

[47] B. Rödera, F. Röslera, and H. J. Nevilleb. Auditory memory in congenitally blind adults: a behavioral-electrophysiological investigation. *Biological and Experimental Psychology, Philipps-University Marburg (Germany)*, 2001.

[48] L. F. Schneider and H. A. Taylor. How do you get there from here? Mental representations of route descriptions. *Applied Cognitive Psychology*, 13(5):415–441, 1999.

[49] P. Strong. The history of the white cane.
`http://www.njcounciloftheblind.org/brochures/history_of_white_cane.htm`, cited 24.4.2013.

[50] E. Williams. Aviation Formulary.
`http://williams.best.vwh.net/avform.htm`, cited 20.2.2013.

[51] K. Yelamarthi, D. Haas, D. Nielsen, and S. Mothersell. RFID and GPS integrated navigation system for the visually impaired. *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1149–1152, 2010.

Appendix A

List of abbreviations

SONS	Sjednocená organizace nevidomých a slabozrakých ČR (Czech Blind United)
API	Application programming interface
TAMP	Tactile Map Automated Production
GIS	Geographic information system
OSM	Open Street Maps
XML	Extensible Markup Language
REST	Representational State Transfer
SQL	Structured Query Language
HTTP	Hypertext Transfer Protocol
HW	Hardware
IE	Internet Explorer

Appendix B

Installation guide

This guide shows how to install complete system with all components needed to run our application. To start, we need a suitable server computer with connection to Internet and we need to download Debian installation CD. Be aware that exact version numbers of all software has to be used, because other configurations might not work correctly. Even newer versions of programs from OSM project do not have to work together, so pay attention and use versions listed in these instructions.

B.1 Operation system installation

We install basic Debian system (version 6.0.6) from CD that can be downloaded freely from Debian website. Afterr booting from installation CD, follow instructions and proceed according to following steps:

1. Choose install and choose your root password.
2. Select your language and keyboard setting and select automatic network configuration .
3. Create user named `osm` and choose your own password.
4. When asked, select option to use whole hard drive, create one partition and finish partitioning.
5. Confirm write to disk.
6. System is now being installed, when asked choose not to insert another disk and use network mirror (choose appropriate to your location).
7. During installation you will be asked to select software packages to install, be sure to select following:
    - Graphical Desktop Environment
    - Web server
    - SQL database

- SSH server
- Standard System Utilities

8. You do not have to keep WorkGroup for Samba server.
9. Install grub loader to master drive – select yes.
10. After reboot you should be able to log in as `osm`.
11. Go to menu Applications > Accessories and run Root terminal.
12. Create directory for our system `/home/osm/naviterier` (you will be saked for root password you entered during installation).

B.2 Installation of Osmosis application

This is most difficult installation step. Unfortunately, there is not stable Debian package available of the version we need (osmosis 0.40.1), so we have to use unstable package. This package is downloadable from debian `sid` sources. You can download package and install it using `dpkg -i filename` command, but you will have to manually resolve any dependencies. This means manually downloading and installing missing packages that will be reported.

B.3 Installaton of Rails Port + PostgreSQL

This is based on installation instructions on Rails Port website [21], but is modified for Debian and cleared of bugs. Start by running Root Terminal from Applications > Accessories menu. Note that MySQL root password is same as root password of operating system.

```
# apt-get install subversion
# apt-get install imagemagick

# aptitude install ruby ruby1.8-dev libxml2-dev libxml-ruby1.8
libxml-parser-ruby1.8 libmysql-ruby1.8 rails mysql-server rubygems
librmagick-ruby libmysqlclient15-dev

# gem install -v=2.0.2 rails
# gem install mysql
# gem install libxml-ruby
# gem install -v 0.9.93 composite_primary_keys

# apt-get install libmagick9-dev
# gem install rmagick
```

This will install subversion, image libraries needed for Rails Port, Ruby on Rails and configures ruby for Mysql. Now follow instructions that are specific or Debian to isntall and configure Rails Port application (see also `http://wiki.openstreetmap.org/wiki/The_Rails_Port#Platform-specific_instructions`).

```
# mkdir /home/osm/naviterier/railsport/source
# cp /home/osm/naviterier/railsport/source

# git clone https://github.com/openstreetmap/openstreetmap-website.git
# cd /home/osm/naviterier/railsport/source/openstreetmap-website/

# cp config/example.application.yml config/application.yml
# cp config/example.database.yml config/database.yml
```

Now open file `config/database.yml` and uncomment 3 times lines specifying username and password for databases. Type in username and password: `openstreetmap` to all three databases. Then continue by following commands to install PostgreSQL database.

```
# sudo apt-get install postgresql-contrib libpq-dev
# sudo -u postgres -i
# createuser openstreetmap -s -P

# createdb -E UTF8 -O openstreetmap openstreetmap
# createdb -E UTF8 -O openstreetmap openstreetmap
# createdb -E UTF8 -O openstreetmap openstreetmap
# psql -d openstreetmap < /usr/share/postgresql/8.4/contrib/btree_gist.sql
# exit

# sudo apt-get install postgresql-server-dev-8.4
# cd db/functions
# make libpgosm.so
```

Now continue by creating some functions missing from basic installation and needed by Mapnik.

```
# sudo -u postgres -i

# psql -d openstreetmap -c "CREATE FUNCTION maptile_for_point(int8, int8, int4)
RETURNS int4 AS '/home/osm/naviterier/railsport/source/openstreetmap-website/db/
functions/libpgosm', 'maptile_for_point' LANGUAGE C STRICT;"

# psql -d osm -c "CREATE FUNCTION maptile_for_point(int8, int8, int4)
RETURNS int4 AS '/home/osm/naviterier/railsport/source/openstreetmap-website/db/
functions/libpgosm', 'maptile_for_point' LANGUAGE C STRICT;"

# psql -d openstreetmap -c "CREATE FUNCTION tile_for_point(int4, int4)
RETURNS int8 AS '/home/osm/naviterier/railsport/source/openstreetmap-website/db/
functions/libpgosm', 'tile_for_point' LANGUAGE C STRICT;"

# psql -d osm -c "CREATE FUNCTION tile_for_point(int4, int4)
RETURNS int8 AS '/home/osm/naviterier/railsport/source/openstreetmap-website/db/
```

```
functions/libpgosm', 'tile_for_point' LANGUAGE C STRICT;"

# exit
```

After database is ready you can resume installing Rails Port.

```
# sudo gem install bundler
# sudo aptitude install libmemcached-dev libsasl2-dev libmemcached-dbg
# sudo gem install memcached -no-rdoc -no-ri
# sudo apt-get install libxslt-dev libxml2-dev
# /var/lib/gems/1.8/bin/bundle install
```

Open file `/home/osm/naviterier/railsport/source/openstreetmap-website/Gemfile` and add line `gem 'rake', '0.8.7'`. Without this, you won't be abel to run Rails Port.

```
# /var/lib/gems/1.8/bin/bundle update rake
# rake db:migrate
# gem install builder
# env RAILS_ENV=production rake db:migrate
# rake test
```

Now copy to folder `/home/osm/naviterier/data/` file containing OpenStreetMap data (extension .osm). You can download one from OpenStreetMap website, export from JOSM editor or use one provided on accompanying DVD (Prague.osm). Import this data file to database using Osmosis.

```
# osmosis --read-xml-0.6 file="/home/osm/naviterier/data/Prague.osm"
--write-apidb-0.6 populateCurrentTables=yes host="localhost"
database="openstreetmap" user="openstreetmap" password="openstreetmap"
validateSchemaVersion=no
```

Now we will update database using following commands. Run them twice, for second time replace `openstreetmap` on second line by `osm`

```
# sudo -u postgres -i
# psql openstreetmap     <*------- replace with osm the second time
# select setval('acls_id_seq', (select max(id) from acls));
# select setval('changesets_id_seq', (select max(id) from changesets));
# select setval('client_applications_id_seq', (select max(id) from client_applications));
# select setval('countries_id_seq', (select max(id) from countries));
# select setval('current_nodes_id_seq', (select max(id) from current_nodes));
# select setval('current_relations_id_seq', (select max(id) from current_relations));
# select setval('current_ways_id_seq', (select max(id) from current_ways));
# select setval('diary_comments_id_seq', (select max(id) from diary_comments));
# select setval('diary_entries_id_seq', (select max(id) from diary_entries));
# select setval('friends_id_seq', (select max(id) from friends));
# select setval('gpx_file_tags_id_seq', (select max(id) from gpx_file_tags));
# select setval('gpx_files_id_seq', (select max(id) from gpx_files));
# select setval('messages_id_seq', (select max(id) from messages));
# select setval('oauth_nonces_id_seq', (select max(id) from oauth_nonces));
# select setval('oauth_tokens_id_seq', (select max(id) from oauth_tokens));
# select setval('redactions_id_seq', (select max(id) from redactions));
# select setval('user_blocks_id_seq', (select max(id) from user_blocks));
# select setval('user_roles_id_seq', (select max(id) from user_roles));
# select setval('user_tokens_id_seq', (select max(id) from user_tokens));
# select setval('users_id_seq', (select max(id) from users));
# \q
# psql osm

<* repeat the selects from above *>

# \q
# exit
```

Rails Port should now work and you can run it from command line on test server using this line.

```
# /var/lib/gems/1.8/bin/bundle exec rails server
```

Open web browser and navigate to address `localhost:3000`. You should see slippy map which means that Rails Port is running. Note that Rails Port is not supposed to be run from command line like this. We will install module to Apache HTTP server to run it on background.

B.4 Installation of Apache Passenger

Installing this module will enable Rails Port to run on background and to be accessible from remote computers.

```
# gem install passenger
# apt-get install libcurl4-openssl-dev
# apt-get install apache2-prefork-dev
# apt-get install libaprl-dev
# apt-get install libaprutil1-dev

# /var/lib/gems/1.8/bin/passenger-install-apache2-module

# joe /etc/apache2/apache2.conf

// add three lines that were printed out by passenger installation

# /etc/init.d/apache2 restart
# joe /etc/apache2/sites-available/rails

// add these lines
<VirtualHost :*20001>
   ServerName rails
   DocumentRoot /home/osm/naviterier/railsport/source/openstreetmap-website/public
</VirtualHost>

# a2ensite rails

# joe /home/osm/naviterier/railsport/source/openstreetmap-website/config/environment.rb

// add line ENV['RAILS_ENV'] ||= 'production'

# /etc/init.d/apache2 reload

# joe /etc/apache2/httpd.conf pridat riadok

// add line Listen 20001
```

Now you can edit file `/etc/postgresql/8.4/main/pg_hba.conf` if you want to connect to PostgreSQL database from remote computers (useful debugging). Add lines specifying IP address of remote computers according to examples.

Also add line `listen_addresses = '*'` to file `/etc/postgresql/8.4/main/postgresql.conf`.

Run following commands to finish installation.

```
# chmod 777 -R /home/osm/naviterier/railsport/source/openstreetmap-website/tmp
# /var/lib/gems/1.8/bin/bundle exec rake assets:precompile
```

B.5 Glassfish installation

This installation is also overcomplicated by the fact that Glassfish application server is not fully compatible with OpenJDK runtime installed on Debian as default. Although you can install and run Glassfish on OpenJDK, you won't be able to open Glassfish console. This can be solved by installing Oracle JDK Java runtime first, prior to installing Glassfish.

```
# apt-get -f install
# apt-get install sun-java6-jdk
# update-java-alternatives -s java-6-sun
```

Now download installation file with `.sh` extension from `https://glassfish.dev.java.net/`. Open console as `osm` user (do not run this as root, because you will not be able to use Glassfish from NetBeans IDE) and run following command to install Glassfish (replace filename for downloaded file).

```
# sh  filname.sh -s
```

Glassfish is ready and you have to deploy our application and prepare database connection. Follow instructions in [29]. If you want glassfish to be run automatically when system starts, run these commands:

```
# crontab -e

// add line @reboot /home/osm/glassfish3/glassfish/bin/asadmin start-domain domain1
```

B.6 Installation of Osm2pgsql

This is easy to install, just run following command

```
# Apt-get install osm2pgsql
```

B.7 Installation of Mapnik

Last piece of software we need is Mapnik (slippy map generator). This is also not easy to install, becuase we have to compile it along with Boost libraries. Be sure to download and install exact versions of both Boost and Mapnik as specified (Mapnik 2.0.2), newer versions do not work together or cannot be installed on Debian. Download source tarball from `http://mapnik.org/download/` and save it to home folder. Extract it to separate folder using GUI tool in Debian. Continue by opening root terminal (see also `https://github.com/mapnik/mapnik/blob/master/INSTALL.md`).

```
// in terminal, cp to folder where you extracted source tarball

# ./configure
# aptitude install libproj-dev
# aptitude install libboost-dev
# apt-get install libboost-system-dev
# apt-get install libboost-regex-dev
# apt-get install libboost-thread-dev
# apt-get install libboost-filesystem-dev
# apt-get install libboost-python-dev
# ./configure
# make
# make install
```

This takes some time to install. After this is finished, continue by creating Mapnik database in PostgreSQL (see also `http://wiki.openstreetmap.org/wiki/Mapnik/PostGIS`).

```
# sudo -u postgres -i -H
# createdb -E UTF8 -O openstreetmap gis
# createlang plpgsql gis
# psql -d gis -f /usr/share/postgresql/8.4/contrib/_int.sql  #could cause "Please
 use a database without intarray" error on newer osm2pgsql

# psql -d gis -f /usr/share/postgresql/8.4/contrib/postgis-1.5/postgis.sql
# to populate the table spatial_ref_sys (mandatory for use with osm2pgsql):
# psql -d gis -f /usr/share/postgresql/8.4/contrib/postgis-1.5/spatial_ref_sys.sql
# psql gis -c "ALTER TABLE geometry_columns OWNER TO openstreetmap"
# psql gis -c "ALTER TABLE spatial_ref_sys OWNER TO openstreetmap"
```

Now you can upload same `.osm` file you used during Rails Port installation. Use `osm2pgsql` tool (instructions in Maintenance Guide appendix). Finally, run `find_nearest_point.sql` script from accompanying DVD to create functions needed by Mapnik and then follow instructions on page 63 in [45] to finish installation. After that, installation of system is finished.

Appendix C

Maintenance guide

C.1 Importing new map data

New data have to be imported into both MySQL database used by Rails Port (OSM API) and to PostgreSQL used by Mapnik and our application.

To import to MySQL database, you can either use JOSM map editor, which communicates directly with OSM API and changes are automatically reflected into database. Or you can export your new data in `.osm` format and use osmosis to import them from file.

```
# osmosis --read-xml-0.6 file="/home/osm/naviterier/data/Prague.osm"
--write-apidb-0.6 populateCurrentTables=yes host="localhost"
database="openstreetmap" user="openstreetmap" password="openstreetmap"
validateSchemaVersion=no
```

You will always have to import new data to PostgreSQL using command line tools. Start by using `osm2pgsql` to import mentioned `.osm` file into second database.

```
# osm2pgsql --slim -S/usr/share/osm2pgsql/default.style -H127.0.0.1 -P5432
-Uopenstreetmap -dgis /home/osm/uploaded_map.osm
```

Then you have to run SQL script `update_active_layer.sql` (on accompanying DVD and [45]) to update what we call active layer. This is a table of clickable points and footpaths. Then you will have to render slippy map again using Mapnik.

C.2 Generating slippy map

To generate slippy map, you have to run following script. It sets all variables needed to render tiles and starts rendering. Notice that it also converts configuration file to newer version, as this configuration file was written in our previous work [29] for older Mapnik version. Luckily, new version contains utility to do this conversion for us.

```
# source /home/osm/naviterier/mapnik/scripts/knp/set-mapnik-env_KN_p
# cd /home/osm/naviterier/mapnik/scripts/knp/
# ./customize_points >/home/osm/naviterier/mapnik/osm.xml
# /home/osm/naviterier/mapnik/mapnik-v2.0.2/utils/upgrade_map_xml/upgrade_map_xml.py
 /home/osm/naviterier/mapnik/osm.xml --in-place
# ./generate_KN_points
```

C.3 Deploying new version of application

To deploy new application version we first need to build sources (we use NetBeans IDE project). Ypu have to build OSMNavigationServices project (Enterprise Application project), which compiles into `.ear` file that can be found in `dist` subfolder of project. If you build one of other projects, you will still be able to deploy the result to Glassfish server, but only one part of application will be running (either client side or server side). Building main project contains both.

First, you have to undeploy existing version, then deploy new version. To do so, run following commands:

```
# /home/osm/glassfish3/glassfish/bin/
# sudo ./asadmin undeploy OSMnavigationSerivces
# sudo ./asadmin deploy /home/osm/naviterier/OSMApplication/
  final/OSMnavigationSerivces.ear
```

Appendix D

Results

Here, we include some examples of application results - generated descriptions. We show planner route first on the map and list generated description after that. Descriptions were not edited and are listed as they were generated. English transcription is done manually and is not generated by our application.

Example 1 See figure D.1 and resulting description D.2 and D.3.

Example 2 See figure D.4 and resulting description D.5 and D.6.

Example 3 See figure D.7 and resulting description D.8 and D.9.

Example 4 See figure D.10 and resulting description D.11 and D.12.

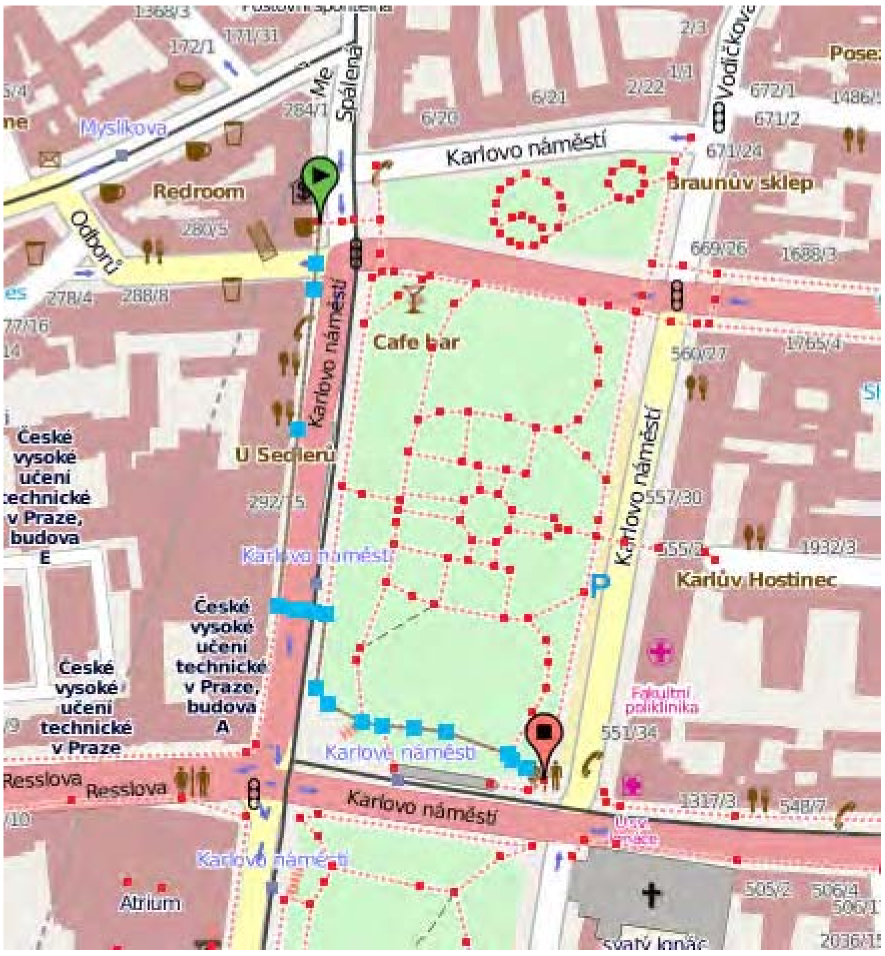

Figure D.1: Route on Charles place, Prague.

```
Napravo od vás je stěna domu. Pokračujte rovně 25.0 metrů. Potom přejděte
přes přechod.

Nalevo od vás je provoz tramvají. Napravo od vás je stěna domu. Pokračujte
rovně 120.0 metrů. Cestou projdete kolem restaurace U Sedlerů, restaurace
Čínský bufet.

Otočte se vlevo. Pokračujte rovně 10.0 metrů. Potom přejděte přes přechod.

Nalevo od vás je provoz tramvají. Napravo od vás je provoz tramvají.
Pokračujte rovně 9.0 metrů. Potom přejděte přes přechod přes koleje.

Otočte se vpravo. Pokračujte rovně 30.0 metrů.

Otočte se vlevo. Pokračujte rovně 8.0 metrů.

Nalevo od vás je park. Napravo od vás je park, schody. Otočte se mírně vlevo
(45 stupňů). Pokračujte rovně 70.0 metrů.

Nalevo od vás je park. Napravo od vás je park. Pokračujte rovně 15.0 metrů.
Minete 1 odbočku vlevo.

Jste na místě.
```

Figure D.2: Description for example 1 (in Czech)

```
On your right is wall of a building. Continue straight ahead 25 meters.
Then use the crossing.

On your left are tramway tracks. On your right is wall of a building.
Continue straight ahead 120 meters. You will pass restaurant U Sedlerů,
restaurant Čínský bufet.

Turn left. Continue straight ahead 10 meters. Then use the crossing.

On your left are tramway tracks. On your right are tramway tracks.
Continue straight ahead 9 meters. Then cross the tramway tracks.

Turn right. Continue straight ahead 30 meters.

Turn left. Continue straight ahead 8 meters.

On your left is park. On your right is park, steps. Turn left slightly
(48 degrees). Continue straight ahead 70 meters.

On your left is park. On your right is park. Continue straight ahead 15
meters. You will pass one turn-off to left.

You have reached your destination.
```

Figure D.3: Description for example 1 (in English)

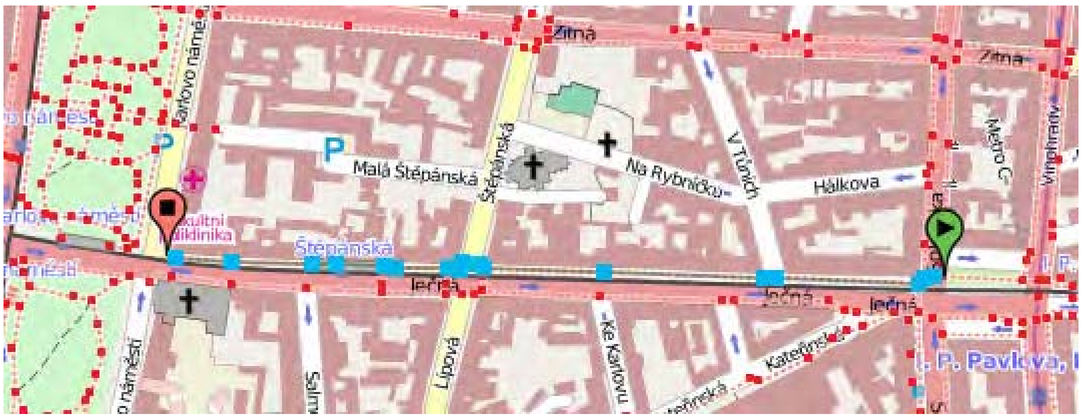

Figure D.4: Route from I.P. Pavlova to Charles Place, Prague.

```
Nalevo od vás je trávnatá plocha. Napravo od vás je trávnatá plocha.
Pokračujte rovně 15.0 metrů. Potom přejděte přes přechod.

Otočte se mírně vlevo (45 stupňů). Pokračujte rovně 6.0 metrů.

Nalevo od vás je provoz tramvají. Napravo od vás je stěna domu. Otočte
se mírně vpravo (45 stupňů). Pokračujte rovně 120.0 metrů. Potom přejděte
přes přechod.

Nalevo od vás je provoz tramvají. Napravo od vás je stěna domu.
Pokračujte rovně 215.0 metrů. Cestou projdete kolem banky FIO.
Minete 1 odbočku vlevo.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 4.0 metrů.

Otočte se mírně vlevo (45 stupňů). Pokračujte rovně 15.0 metrů. Potom
přejděte přes přechod.

Otočte se vlevo. Pokračujte rovně 5.0 metrů.

Nalevo od vás je provoz tramvají, zastávka tramvají Štěpánská.
Napravo od vás je stěna domu. Otočte se vpravo. Pokračujte rovně
220.0 metrů. Cestou projdete kolem restaurace Pizzeria Messalina,
restaurace Svatého Ignáce. Minete 5 odboček vlevo a 2 odbočky vpravo.

Jste na místě.
```

Figure D.5: Description for example 2 (in Czech)

```
On your left is grassed area. On your right is grassed area.
Continue straight ahead 15 meters. Then use the crossing.

Turn left slightly (45 degrees). Continue straight ahead 6 meters.

On your left are tramway tracks. On your right is wall of building.
Turn right slightly (45 degrees). Continue straight ahead 120 meters.
Then use the crossing.

On your left are tramway tracks. On your right is wall of building.
Continue straight ahead 215 meters. You will pass FIO bank.
You will pass 1 turn-off left.

Turn right slightly (45 degrees). Continue straight ahead 4 meters.

Turn left slightly (45 degrees). Continue straight ahead 15 meters.
Then use the crossing.

Turn left. Continue straight ahead 5 meters.

On your left are tramway tracks,tram stop Štěpánská.
On your right is wall of building. Turn right. Continue straight
ahead 220 meters. You will pass Pizzeria Messalina,
restaurant Svatého Ignáce. You will pass 5 turn-offs left and 2 right.

You have reached your destination.
```

Figure D.6: Description for example 2 (in English)

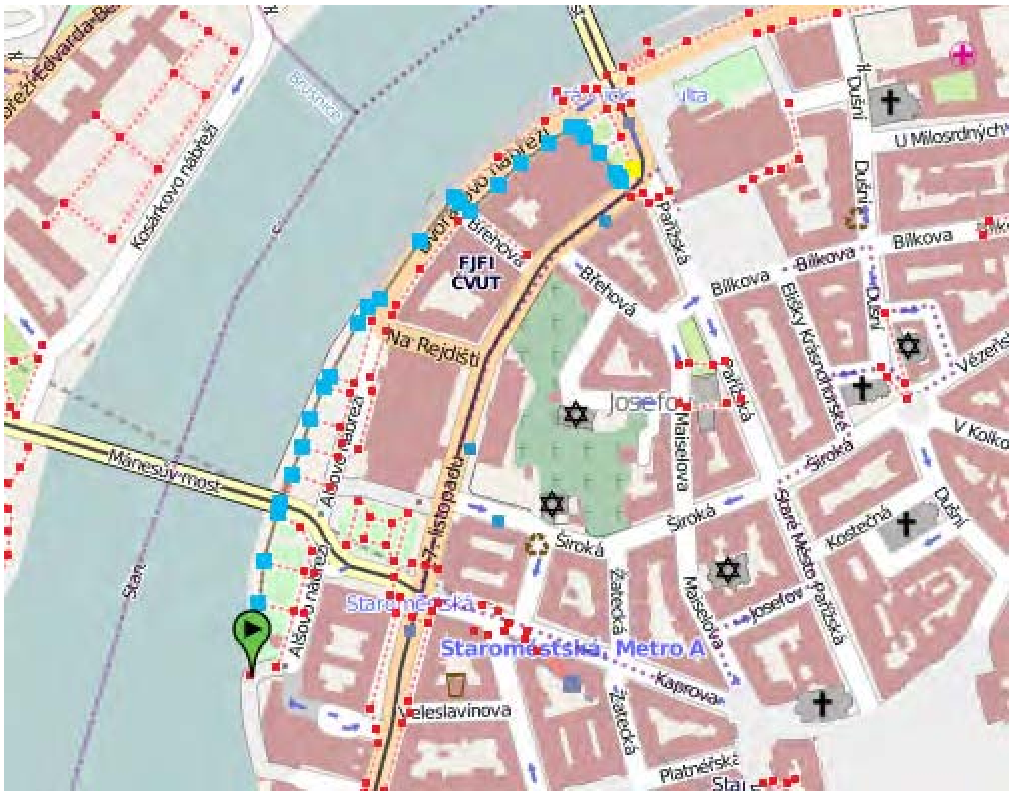

Figure D.7: Route on bank of Vltava river, Prague.

```
Nalevo od vás je park, provoz tramvají. Napravo od vás je park,
 provoz tramvají, schody, stěna domu. Pokračujte rovně 135.0 metrů.
 Minete 3 odboček vpravo.

Nalevo od vás je schody. Napravo od vás je provoz tramvají, schody.
 Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 165.0 metrů.
  Cestou přejděte po schodech. Minete 5 odboček vpravo.

Pokračujte rovně 105.0 metrů. Minete 1 odbočku vpravo.

Otočte se vpravo. Pokračujte rovně 15.0 metrů. Minete 1 odbočku
vlevo a 2 odbočky vpravo. Potom přejděte přes přechod.

Otočte se vlevo. Pokračujte rovně 100.0 metrů.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 6.0 metrů.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 6.0 metrů.

Nalevo od vás je trávnatá plocha, schody. Pokračujte rovně 40.0 metrů.
Cestou projdete kolem schodů. Minete 1 odbočku vlevo.

Napravo od vás jsou schody. Pokračujte rovně 8.0 metrů.

Otočte se vlevo. Pokračujte rovně 10.0 metrů.

Jste na místě.
```

Figure D.8: Description for example 3 (in Czech)

```
On your left is park, tramway tracks. On your right park, tramway
tracks, steps, wall of building. Continue straight ahead 135 meters.
You will pass 3 turn-offs right

On your left are steps. On your left are tramway tracks, steps.
Turn right slightly (45 degrees). Continue straight ahead 165 meters.
Then go down the steps. You will pass 5 turn-offs right.

Continue straight ahead 105 meters. You will pass 1 turn-off right.

Turn right. Continue straight ahead 15 meters. You will pass 1 turn-off
left and 2 right. Then use the crossing.

Turn left. Continue straight ahead 100 meters.

Turn right slightly (45 degrees). Continue straight ahead 6 meters.

Turn right slightly (45 degrees). Continue straight ahead 6 meters.

On your lef tis grassed area, steps. Continue straight ahead 40 meters.
You will pass steps. You will pass 1 turn-off left.

On your right are steps. Continue straight ahead 8 meters.

Turn left. Continue straight ahead 10 meters.

You have reached your destination.
```

Figure D.9: Description for example 3 (in English)

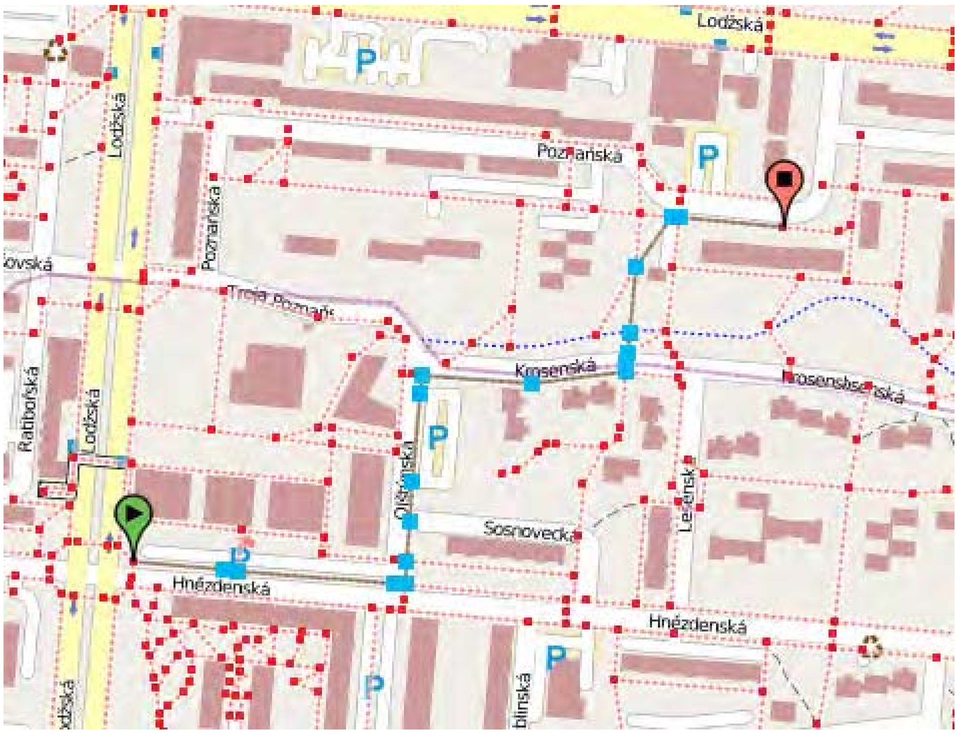

Figure D.10: Route in residential area, Prague.

```
Pokračujte rovně 80.0 metrů. Potom přejděte přes přechod.

Pokračujte rovně 130.0 metrů. Potom přejděte přes přechod.

Napravo od vás je parkoviště. Otočte se vlevo. Pokračujte rovně
160.0 metrů. Minete 4 odboček vlevo a 3 odbočku vpravo.

Napravo od vás je parkoviště, stěna domu. Otočte se vpravo.
Pokračujte rovně 155.0 metrů.

Otočte se vlevo. Pokračujte rovně 15.0 metrů. Potom přejděte
přes přechod.

Napravo od vás je stěna domu. Pokračujte rovně 65.0 metrů.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 45.0 metrů.

Otočte se mírně vpravo (45 stupňů). Pokračujte rovně 85.0 metrů.

Jste na místě.
```

Figure D.11: Description for example 4 (in Czech)

```
Continue straight ahead 80 meters. Then use the crossing.

Continue straight ahead 130 meters. Then use the crossing.

On your right is parking lot. Turn left. Continue straight
ahead 160 meters. You will pass 4 turn-offs left and 3 right.

On your right is parking lot, wall of building. Turn right.
Continue straight ahead 155 meters.

Turn left. Continue straight ahead 15 meters. Then use the crossing

On your right is wall of building. Continue straight ahead 65 meters.

Turn right slightly (45 degrees). Continue straight ahead 45 meters.

Turn right slightly (45 degrees). Continue straight ahead 85 meters.

You have reached your destination.
```

Figure D.12: Description for example 4 (in English)

Appendix E

Content of accompanying DVD

```
.
|-- OSMnavigationSerivces         - NetBeans project of our application
|-- Latex_sources                 - Latex source codes of this work
|-- scripts                       - Bash and SQL scripts for maintenance tasks
|-- Prague.osm                    - OSM data file of Prague
|-- DB_setup.pdf                  - Setup instructions for Glassfish database pool
'-- boksajak_2013dipl.pdf         - thesis text in PDF format
```