Traffic-dependent routing based on OpenStreetMap data and TMC messages

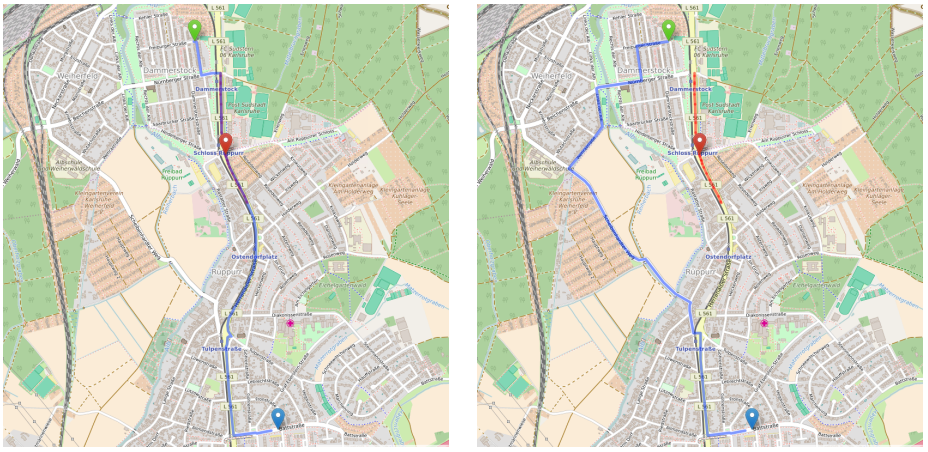
Jan Strauß
Matr.-Nr. 2727381
swt85730@stud.uni-stuttgart.de

In this report a routing application that uses data from the OpenStreetMap project and Traffic Message Channel (TMC) messages to provide traffic-dependent routing on a web-based user interface is presented. This application was developed within the context of the *Fachpraktikum: Algorithms for OpenStreetMap data* course during the summer semester 2016 at the University of Stuttgart.

1 Introduction

Participants of the course *Fachpraktikum: Algorithms for OpenStreetMap data* were tasked to build a routing application that uses OpenStreetMap [1] data to provide distance and time based routing for cars, bikes and pedestrians. After this base task was completed, the routing application should be extended. This application was extended to use current traffic information, as provided by TMC via radio broadcast, during the route calculation, to, for example, route around a traffic congestion. An example for routing around a traffic event is shown in Figure 1. A similar system was realized by Sanwald in the context of a bachelor thesis [2]. While the implementation by Sanwald is Java-based, the application presented in this report is written in Rust [3]. Rust is a new system programming language with strong guarantees on memory and thread safety. The language fits the requirements for a routing application very well: It is a compiled language that offers performance on-par with C++, the standard library contains easy to work with collection types and the language ecosystem offers libraries that provide needed features like a Http server or a parser for pbf files (the OpenStreetMap data format). For a more in-depth discussion see section 4.

The remainder of this report is structured as followed: First, the OpenStreetMap data format and TMC concepts are presented in section 2. In the next section, 3, the system is presented and in section 4 some implementation



(a) Ignoring TMC Events in the route calculation.

(b) Respecting TMC Events in the route calculation.

Figure 1: Effect of respecting TMC events in the route calculation on the shortest found route for the same start and end points.

details are presented and a brief comparision with Sanwalds system is given. Finally a conclusion and possible future work are presented in section 5.

2 Background

2.1 OpenStreetMap (OSM)

OpenStreetMap [1] is a project that aims to build a free map of the world. The geographical information is contributed by over 3 million users worldwide and the database contains over 3.5 billion nodes and over 370 million ways [4], the primitive data types the map is build from. The data can be freely downloaded and used and there are extracts available that are limited to certain areas (e.g. country, continent, region, city). The data is available in different formats, where to most efficient one is the *.osm.pbf binary format, that is also used by this application. The primitive data types have their associated information stored in key-value pairs called tags. For more information about the *.osm.pbf format and the data types used by OSM, see section 3.1

2.2 Traffic Message Channel (TMC)

The Traffic Message Channel is a service that broadcasts traffic information. It is typically received via RDS, the Radio Data System, a standard to embed digital information in FM radio transmissions. TMC messages consist of a location code, an event code, the direction, incident extend and other fields

that we will ignore here. The location code is defined in a document called LCL, the location code list. A location can have a neighbor location in positive and negative direction and the extend field of a TMC message is measured in hops over neighbors in the direction of the direction field of the message. The event codes are defined in the ECL, the event code list. So with the information from TMC messages one can determine the type of event, the location, direction and extend using the LCL and ECL. To use TMC messages in combination with routing based on OSM data two problems have to be solved: First it is necessary to have a mapping from TMC location code to the corresponding OSM data entry. Luckily the OSM dataset contains TMC Tags for some of its ways. The second problem is to determine the slowdown of a event code. To solve this, a simple lookup table is build, see section 3.4.

For Germany, a new tagging scheme for TMC-data was proposed and is widely adopted today. This new scheme drastically simplifies the handling of TMC tags. The new tagging format uses the string "tmc" as key and a string in the form of "DE:1234+5678". The first number is the current location id, the + is used to describe the postive direction and the second number describes the next location id in the posiive direction. There are, respectively, the - sign for the negative direction and the / sign, which is a shorthand for A+B and B-A. More information about the new scheme can be found at [5].

3 System

The general architecture of the system is shown in Figure 2. A OSM extract in the *.osm.pbf format is used as input by the parser to build the routing graph and to generate other routing information (see section 3.2). This state can be saved to disk and be loaded upon start, so that not every restart of the system causes the parser to run. The State is used by the Routing module which is called by the HTTP-Server when a routing request was received from the web-UI. The HTTP-Server also serves the static content of the web-UI, e.g. the leaflet library. The TMC module starts the RDSD and RDSQUERY binaries and reads the STDOUT of RDSQUERY and generates new TMC events from this data source and updates the State appropriately.

3.1 Parsing OSM data

OpenStreetMap provides its map data in different formats, e.g. osm.pbf or xml. The PBF (Protocolbuffer Binary Format) is the most efficient format. It is based on Googles Protobuf library. The File format uses several tricks to reduce the file size which makes reading such files complicated. However several libraries provide convenient wrappers that read the files and offer a abstraction on the level of the OSM primitives.

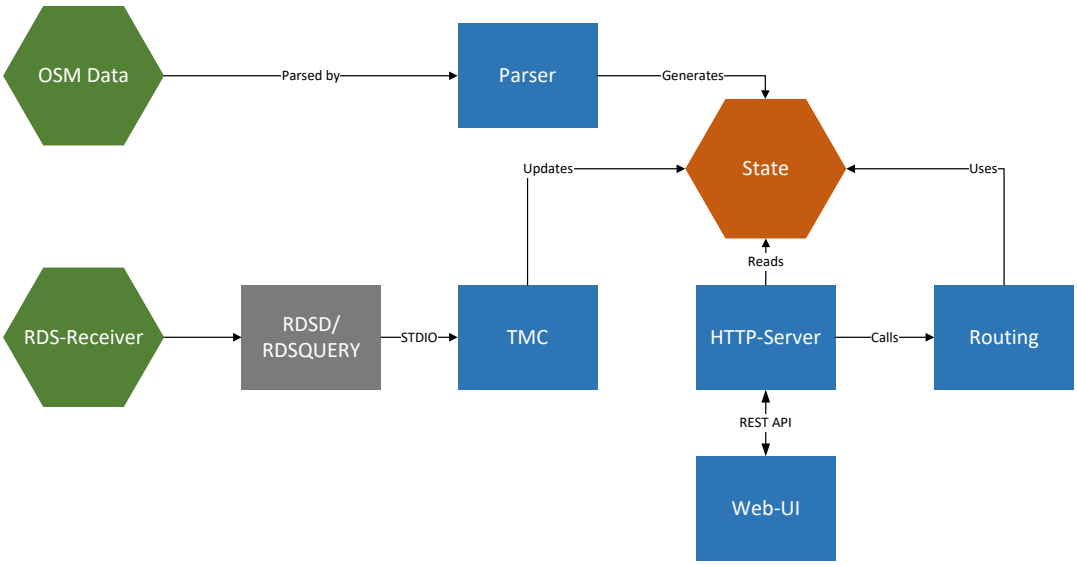


Figure 2: Architectural overview. Modules are blue, input artifacts green and the internal state orange.

These primitives are Nodes, Ways and Relations. Nodes are single point locations, ways chain multiple nodes together to form a path and relations contain meta information about multiple ways and/or nodes. These primitives have multiple Tags, key-value pairs containing additional information about the tagged entity.

osmpbfreader-rs [6] is used as wrapper around the file format. It provides a convenient API to read *.osm.pdf files. The parser module loops over the file contents in three passes: In the first pass the relevant ways are filtered out and all nodes that are contained in these ways are remembered. in the second pass, relevant node information (position) is extracted and in the third pass single edges are extracted from the ways.

After the three passes the extracted graph information is stored using the offset-array method. This allows fast access to the outgoing edges of a node while also being size-efficient. As a last step a grid is build from the nodes and their positions to allow to search for the nearest node from any given location.

3.2 Routing data representation

As already mentioned, the offset-array method is used to store the routing data. The complete set of data structures used are presented in Figure 3.

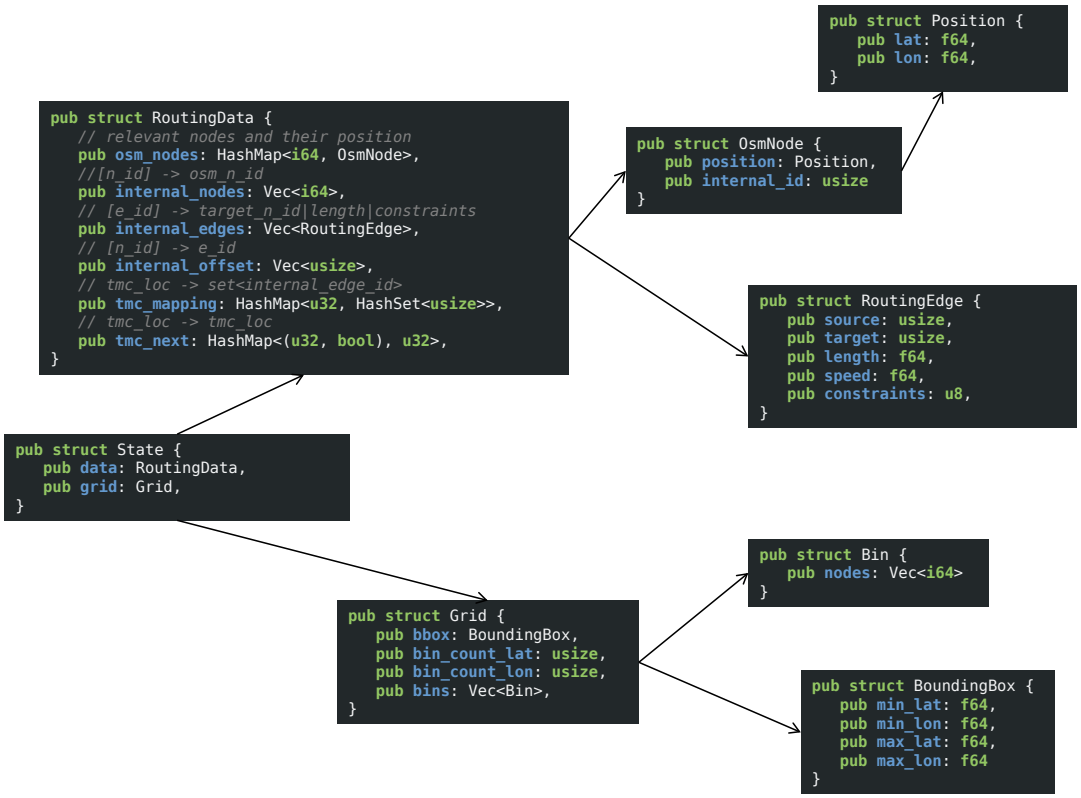


Figure 3: Internal data representation of the routing graph.

3.3 Route calculation

The shortest path is calculated using the Dijkstra algorithm. The binary heap implementation of the rust standard library is used as the data structure to keep known nodes. Depending on the vehicle (car, bike, pedestrian) edges that don't allow the chosen vehicle are skipped and depending on the metric (distance, time) and if tmc should be respected, a different cost function will be used for the edge.

3.4 TMC handling

TMC is handled at two different points in the system. First the parser will extract two static maps containing information that is needed to handle TMC messages during runtime. The first is a map from TMC-Location ID to a set of edge-ids and the second a map from a tuple of TMC-Location ID and boolean to TMC-Location ID. The first map is used to find the edges which are part of a location id and the second map is used to find neigboring locations of a given location and direction. While this information could

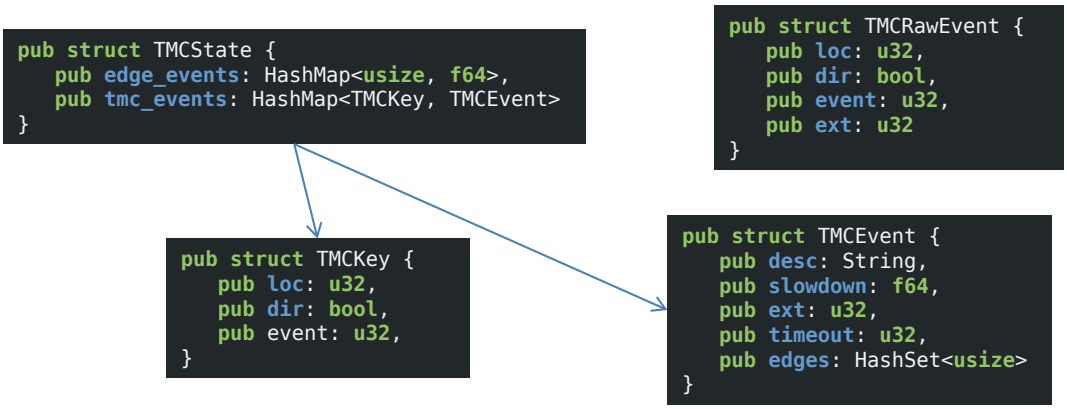


Figure 4: Internal data representation of TMC events.

event description contains	resulting slowdown
bridge closed*	1.00
tunnel closed*	1.00
stationary traffic	0.95
queuing traffic	0.80
slow traffic	0.50
heavy traffic	0.30
construction work*	0.60
traffic problem	0.10
accident	0.75

Table 1: Lookup table used to determine the slowdown factor of an event id

also be extracted from the LCL, The new tagging scheme of OSM allows to extract this information from OSM. This has the advantage that only relevant entries are stored.

The second point is the TMC module. This module will handle incoming TMC messages, use the second map to find all relevant location IDs and then use the first map to find the affected edges. It will then update the two maps in the TMCState struct (see Figure 4). To determine the delay of a event id, a lookup table similar to the one used in [2] was build by reading the ECL (see Table 1). This table is translated into an if/if else/else structure for the same reasons to those given in [2]. Fields marked with * contain omitted synonyms.

Further, an extra thread will increase a counter on each entry and, if not reset by a incoming message, remove any event after 10 minutes.

To receive TMC messages, a radio receiver is needed and from the radio

signal the RDS stream has to be extracted, from which the TMC messages can then be extracted. As receiver the InstantFM Music from ADS Tech was used. The Linux Kernel contains drivers for this device. To parse the RDS stream from the radio stream, RDSD (Radio Data System daemon) was used in combination with rdsquery. RDSD and rdsquery were developed by Hans J. Koch, but the project homepage (`rdsd.berlios.de`) given in the sources is no longer available. To make RDSD and rdsquery compile on x64 ubuntu 16.04, a new include has to be added to all source files:

```
#include <unistd.h>
```

Running rdsd is straightforward, for rdsquery the following parameters have been used:

```
rdsquery -s localhost -c 0 -t tmc
```

This causes rdsquery to only print tmc messages to STDOUT. These have the format

```
S evt=63 loc=12000 ext=3 dur=0 dir=1 div=0
```

which is parsed by the TMC module that spawns both RDSD and RDS-QUERY as child processes.

3.5 User interface

The UI is depicted in Figure 5. The user can click on the map to define the source and target location and select parameters like vehicle, metric and if tmc events should be considered during routing. A route is presented as polyline on the map and the length and travel duration is listed on the sidebar on the right. TMC events are shown as red polylines with a marker and popup containing the event description and also be listed on the right.

The UI is realized in JavaScript and uses leaflet to render the map and jquery to ease the communication with the API endpoints of the HTTP-Server module.

4 Implementation

The system is implemented in about 2753 lines of Rust code and about 200 lines of HTML, CSS and JavaScript for the UI[1]. Parsing the Baden-Würrtemberg osm.pbf file (about 400MB) takes about 3 minutes, a route calculation for this dataset typically under 2 seconds.

[1] counted with CLOC [7]

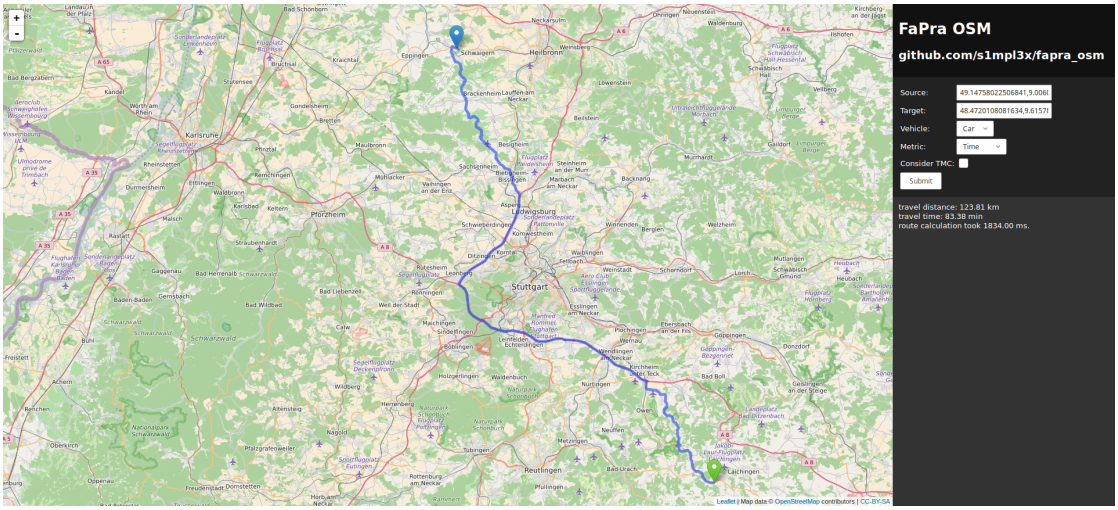


Figure 5: Web-based graphical user interface

4.1 Comparision with "Verkehrsabhängiges Routing basierend auf TMC-Nachrichten"

As already mentioned the systems differ in their implementation language. Sanwald Is also using the old TMC tagging scheme. The systems also differ in the way the TMC events are handled. While Sanwald directly modified the routing graph, This system keeps the TMC information separate from the graph while minimizing overhead during route calculation. Also Sanwald uses a node delay that is applied to all incoming edges while this system directly maps the delays to the edges. While Sanwald specifies delays in seconds, this system specifies delays as percent of the allowed max speed. The systems also differ in their data formats. While Sanwald uses a textual representation, this system uses a binary format.

4.2 Used third party libraries/software

Several open source libraries were used to build this system:

4.2.1 Rust libraries

Iron [8] HTTP-Framework for Rust.

Iron-staticfile [9] Iron component to serve static content.

Iron-mount [10] Iron middleware to map different URL-paths to different endpoints/static content.

osmpbfreader-rs [6] Wrapper/Reader library around the pdb file format for Rust.

rustc-serialize [11] Enables the compiler to generate serialization code to JSON/Binary representations of structs.

flate2 [12] Compression algorithms, used for compression/decompression of the state file.

bincode [13] Used to read/write the State struct from/to disk.

4.2.2 JavaScript libraries

leaflet [14] JavaScript map library.

jquery [15] General helper functions.

4.3 License and source code

The source code of the presented application is available at [16] and is licensed under the MIT-License:

LICENSE

```
The MIT License MIT

Copyright c 2016 Jan Strauss

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files the "Software", to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

5 Conclusion and future work

In this report an application was presented that provides traffic-dependent routing. OpenStreetMap data is used as the base for the routing graph and TMC messages are used as source of current traffic information. While the application works reliably, there are several areas where improvement is

possible: The parsing could be sped up by using concurrency, The coverage of TMC events could be improved and cache locality could be increased by storing nodes and edges of the graph in the built grid.

A major limiting factor is the sparse coverage of TMC tags in the OSM data. Out of 1457361 ways in the Baden-Württemberg extract, only 2666 contained a TMC tag. In the OSM Wiki it is stated that 2906 of 3226 TMC Objects in Baden-Württemberg are tagged [17], however it is unclear how dated that information is. Due to this fact, the majority of received TMC messages are ignored as they can't be mapped to edges in the routing graph.

The Rust programming language proved to be suited for building this application, only the compile times and limited tooling support are minor negative points.

Possible future work includes to improve the UI with more details about TMC events and the found route, display a list of driving directions similar to popular routing applications. Other possibilities include to change the routing algorithm or to improve the memory efficiency during parsing.

References

[1] OpenStreetMap contributors. (2016). OpenStreetMap, [Online]. Available: `https://www.openstreetmap.org` (visited on 09/28/2016).

[2] T. Sanwald, "Verkehrsabhängiges Routing basierend auf TMC-Nachrichten", Bachelorthesis, Universität Stuttgart, 2013.

[3] Rust Project. (2016). The Rust Programming Language, [Online]. Available: `https://www.rust-lang.org/en-US/` (visited on 09/28/2016).

[4] OpenStreetMap contributors. (2016). OpenStreetMap Statistics, [Online]. Available: `https://www.openstreetmap.org/stats/data_stats.html` (visited on 09/28/2016).

[5] OpenStreetMap contributors. (2016). OpenStreetMap Wiki DE:Proposed features/New TMC scheme, [Online]. Available: `https://wiki.openstreetmap.org/wiki/DE:Proposed_features/New_TMC_scheme` (visited on 09/28/2016).

[6] G. P. (2016). osmpbfreader-rs, [Online]. Available: `https://github.com/TeXitoi/osmpbfreader-rs` (visited on 09/28/2016).

[7] Al Danial. (2016). Cloc count lines of code, [Online]. Available: `https://github.com/AlDanial/cloc` (visited on 09/28/2016).

[8] Iron Core Team. (2016). Iron - Rust web application framework, [Online]. Available: `http://ironframework.io/` (visited on 09/28/2016).

[9] Iron Core Team. (2016). Iron - staticfile, [Online]. Available: `https://github.com/iron/staticfile` (visited on 09/28/2016).

[10] Iron Core Team. (2016). Iron - mount, [Online]. Available: https://github.com/iron/mount (visited on 09/28/2016).

[11] Rust Programming Language. (2016). rustc-serialize, [Online]. Available: https://github.com/rust-lang-nursery/rustc-serialize (visited on 09/28/2016).

[12] A. Crichton. (2016). flate2-rs, [Online]. Available: https://github.com/alexcrichton/flate2-rs (visited on 09/28/2016).

[13] T. Overby. (2016). rustc-serialize, [Online]. Available: https://github.com/TyOverby/bincodee (visited on 09/28/2016).

[14] V. Agafonkin. (2015). Leaflet - a JavaScript library for interactive maps, [Online]. Available: http://leafletjs.com/ (visited on 09/28/2016).

[15] The jQuery Foundation. (2016). jQuery, [Online]. Available: https://jquery.com/ (visited on 09/28/2016).

[16] J. Strauß. (2016). Git Repository, [Online]. Available: https://github.com/s1mpl3x/fapra_osm (visited on 09/28/2016).

[17] OpenStreetMap contributors. (2016). OpenStreetMap Wiki DE:TMC, [Online]. Available: https://wiki.openstreetmap.org/wiki/DE:TMC (visited on 09/28/2016).