Lukas Frauenlob, B.Sc.

Optimierte Modellierung von Energieinfrastrukturen mittels automatisierter Datenselektion

Masterarbeit

zur Erlangung des akademischen Grades
Diplom Ingenieur

eingereicht an der
Technischen Universität Graz

Betreuer
Assoz.Prof. Dipl.-Ing. Dr.techn. Udo Bachhiesl

Mitbetreuer
Dipl.-Ing. Robert Gaugl, B.Sc.

Institut für Elektrizitätswirtschaft und Energieinnovation

Fakultät für Elektrotechnik und Informationstechnik

Graz, Oktober 2019

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Datum	Unterschrift

Kurzfassung

Am *Institut für Elektrizitätswirtschaft und Energieinnovation* der *Technischen Universität Graz* wird seit dem Jahr 2002 *ATLANTIS* entwickelt.

ATLANTIS ist ein Modell der europäischen Elektrizitätswirtschaft und ermöglicht die Untersuchung und Simulation verschiedenster energiewirtschaftlicher Fragestellungen, wie etwa Szenarioanalysen für die Integration erneuerbarer Energien, Entwicklung der Netzinfrastrukturen, Auswirkungen von Neu- oder Rückbau von Leitungen und Kraftwerken, Entwicklung regionaler Strompreise unter verschiedensten Regulierungen wie CO_2-Regelungen, oder etwa Stresstests zur Simulation von Energieverknappung.

Um diese Fragestellungen realitätsnah simulieren zu können, ist ein genaues Modell des Höchstspannungsnetzes notwendig. Bisher wurden Hochspannungsleitungen in mühevoller Handarbeit unter hohem Personalaufwand händisch in die *ATLANTIS*-Datenbank eingetragen.

Da mittelfristig in *ATLANTIS* zusätzlich zum europäischen Energienetz jenes des gesamten afrikanischen Kontinents simulierbar sein soll, stößt diese manuelle Vorgehensweise an ihre Grenzen.

Eine mögliche Lösung dafür ist die Nutzung der Datenbasis des *OpenStreetMap* Projekts. Diese hat zudem den Vorteil, dass sie für viele Länder die einzige öffentlich frei zugängliche Datenbasis über das jeweilige Energienetz eines Landes darstellt.

In dieser Masterarbeit wird die Erstellung eines Programms beschrieben, welches Daten aus *OpenStreetMap* automatisiert so aufbereitet und selektiert, dass diese in einem für *ATLANTIS* nutzbaren Format zur Verfügung stehen.

Beginnend mit einer Analyse der Daten aus *OpenStreetMap* und deren Problemstellen bzw. Unzulänglichkeiten, wird die Struktur des über 3.000 Zeilen langen Programms erklärt und im weiteren Verlauf genau beschrieben. Die größte Herausforderung besteht darin, den sehr umfangreichen Datensatz drastisch, aber dennoch sinnvoll zu reduzieren und dabei sicherzustellen, dass keine wichtigen Daten unabsichtlich entfernt werden.

Schlussendlich werden in dieser Arbeit der exportierte Datensatz mit dem bereits in *ATLANTIS* verwendeten, händisch eingetragenen Datensatz verglichen. Der Vergleich zeigt eine hohe Übereinstimmung beider Datensätze und somit die Verwendungsfähigkeit des Programms.

Die zukünftige intensive Verwendung dieses Programms wird den notwendigen Zeitaufwand für das Hinzufügen eines neuen Landes oder einer neuen Spannungsebene drastisch reduzieren und bei gleichem Personaleinsatz die Anzahl der simulierbaren Ländern stark erhöhen.

Abstract

Since 2002 *ATLANTIS* has been developed at the *Institute of Electricity Economics and Energy Innovation* of *Graz University of Technology*.

ATLANTIS is a model of the European electricity industry and allows the investigation and simulation of various questions in the energy industry, such as scenario analyzes for the integration of renewable energies, development of network infrastructures, effects of construction or deconstruction of lines and power plants, development of regional electricity prices under various regulations such as CO_2 regulations, or stress tests to simulate energy shortages.

In order to be able to realistically investigate these questions, an exact model of the high voltage grid is required. Up to now, power lines have been added to the *ATLANTIS* database tediously and manually with great personal effort.

Since *ATLANTIS* is intended to simulate not only the European energy grid but also the energy grid of the entire African continent in the medium term, this manual approach reaches its limits.

One possible solution would be to use the database of the *OpenStreetMap* project. This too has the advantage that for many countries *OpenStreetMap* is the only publicly accessible database of the energy grid of the respective country.

This master's thesis describes the creation of a program that automatically processes, selects and exports data from *OpenStreetMap* so that it is available in an *ATLANTIS* compatible format.

Starting with an analysis of the data from *OpenStreetMap* and its problems and inadequacies, the structure of the more than 3,000-line program is explained and described in detail. The biggest challenge is a significant but reasonable reduction the very large original data set, with great care taken that no important data is inadvertently lost.

Finally, the exported data set is compared with the original, manually added data set currently used in *ATLANTIS*. The comparison shows a high degree of similarity between the two data sets and thus high usefulness of the program.

Future intensive use of this program will radically reduce the time required to add new voltage levels, or even new countries, to the database of *ATLANTIS* and significantly increase the number of available countries for *ATLANTIS* given the same personal expense.

Inhaltsverzeichnis

1 Einleitung

Elektrizität ist aus unserem heutigem Leben nicht mehr wegzudenken. Die Selbstverständlichkeit, jederzeit Zugriff auf Strom zu haben, ist in unseren Köpfen und Gewohnheiten fest verankert.

Durch den stetig steigenden Energieverbrauch und die Herausforderungen, die die Energiewende mit sich bringt, steigen die Anforderungen an unsere Versorgungsnetze und damit auch deren Komplexität. Unter diesen Voraussetzungen wird immer mehr Wert auf Szenarienanalysen und aussagekräftige Zukunftsprognosen gelegt.

Am *Institut für Elektrizitätswirtschaft und Energieinnovation* der *Fakultät für Elektrotechnik und Informationstechnik* der *Technischen Universität Graz* wird bereits seit dem Jahr 2002 *ATLANTIS* entwickelt.

ATLANTIS ist ein Modell der europäischen Elektrizitätswirtschaft. Anfangs umfasste *ATLANTIS* hauptsächlich europäische Länder, in den letzten Jahren konnte jedoch der *Mittelmeerring* geschlossen werden, was nun bedeutet, dass inzwischen auch ausführliche Analysen in Europa und dem Mittelmeerraum möglich sind.

Um sich ein besseres Bild von ATLANTIS zu bekommen, ist in Abb. 1.1 das Höchstspannungsnetz von Österreich dargestellt, genau genommen die 220 kV und 380 kV Spannungsebene sowie verschiedene Kraftwerkstypen und Netzknoten.

1 Einleitung

In über 50 Personenjahren Entwicklungseinsatz aus den Disziplinen Energiewirtschaft, Elektrotechnik, Maschinenbau, Kraftwerkstechnik, Betriebswirtschaft, Volkswirtschaft, Informatik und Recht ist so ein leistungsstarkes Modell mit folgenden Kenndaten entstanden [1]:

- 52 Länder,
- 9.700 Leitungen bzw. Transformatoren,
- 6.500 Netzknoten,
- 26.400 Kraftwerke von 27 unterschiedlichen Typen sowie
- Inklusion der 100 größten Elektrizitätsunternehmen.

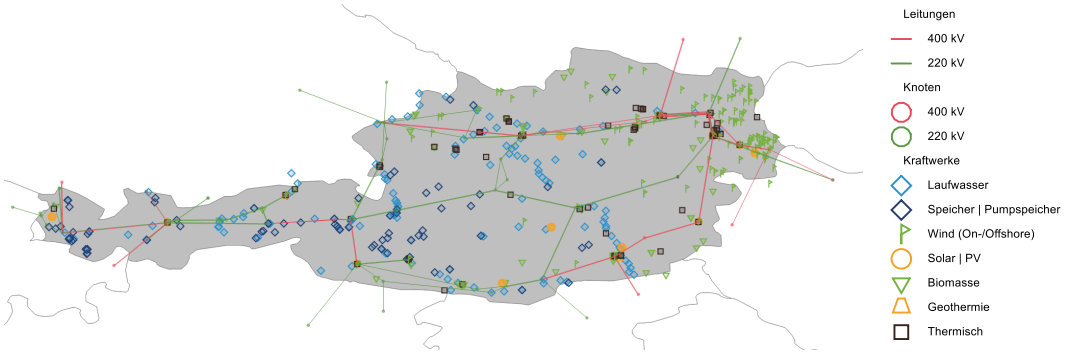
Mit *ATLANTIS* lässt sich eine Vielzahl von Forschungsfragen untersuchen, unter anderem die

- Entwicklung regionaler Strompreise,
- Auswirkungen von Neubau von Leitungen oder Kraftwerken,
- Auswirkungen von Rückbau von Leitungen oder Kraftwerken,
- Quantifizierung des volkswirtschaftlichen Nutzens von Investitionen,
- Szenarioanalysen für die Integration erneuerbarer Energien,
- Systemgrenzkosten erneuerbarer Energien,
- Vorabanalysen von verschiedenen Regulierungen und Marktorganisationen wie neue Richtlinien, CO_2-Regelungen, etc.,
- Stresstests zur Simulation von Energieverknappungen,
- uvm.

Mit diesem Datensatz ist es unter anderem möglich Lastflusssimulationen durchzuführen. In diesen wird untersucht, welchen Auslastungen einzelne Leitungen zum jetzigen Zeitpunkt oder in ferner Zukunft unterliegen werden. In Abb. 1.2 ist eine Lastflusssimulation der österreichischen Höchstspannungsebene zum Zeitpunkt Juni 2019 zu sehen.

Anhand Lastflusssimulationen wie dieser können potentielle Engstellen im Übertragungsnetz, wie sie zum Beispiel derzeit gut an der 220 kV *Salzburgleitung* zu sehen sind, frühzeitig erkannt werden.

1 Einleitung



(a) Darstellung in *VISU* (b) Legende

Abbildung 1.1: *ATLANTIS* - Leitungsnetz von Österreich [2]

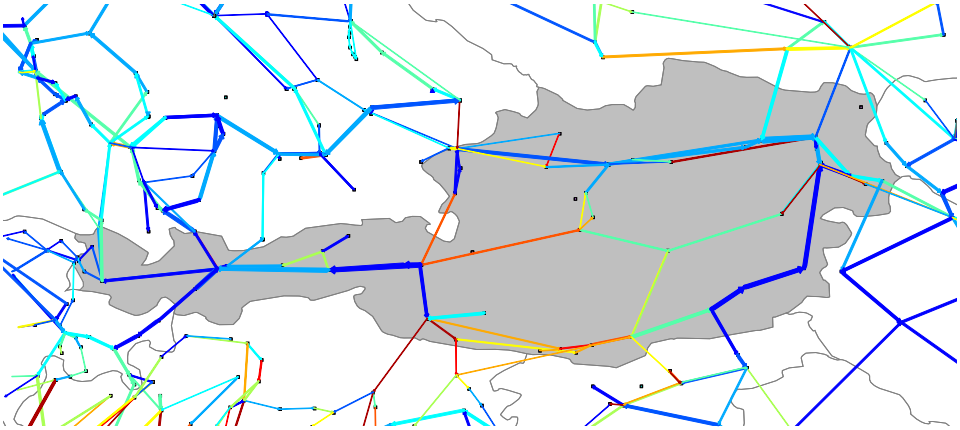


Abbildung 1.2: *ATLANTIS* - Lastflusssimulation von Österreich [2]

2 Motivation

Im vorherigen Kapitel 1 wurde in Abb. 1.2 das österreichische Höchstspannungsnetz gezeigt. Nicht näher wurde darauf eingegangen, *wie* die Daten des österreichischen Höchstspannungsnetzes in *ATLANTIS* implementiert wurden.

Die Datenbank von *ATLANTIS* arbeitet intern mit Tabellen im Excel-Dateiformat. In mühevoller händischer Kleinarbeit wurde jede einzelne Leitung, Zeile für Zeile, in diese Datenbank eingetragen. Die Leitungsdaten stammen aus unterschiedlichen Quellen, zu einem großem Teil jedoch aus der Netzkarte der *ENTSO-E*, dem Verband Europäischer Übertragungsnetzbetreiber.

In Abb. 2.1 ist die Netzkarte, wie sie auf der Website der *ENTSO-E* [3] öffentlich zugänglich ist, dargestellt. Einzelne Leitungen wurden manuell herausgesucht, die Start- und Endknotenkoordinaten händisch in *Google Maps* gesucht und so in die Datenbank eingetragen.

Diese Vorgehensweise ist aus zwei Gründen verbesserungswürdig:

- Der Zeitaufwand und damit der Personaleinsatz ist sehr hoch.
- Für viele Länder gibt es keine öffentlich verfügbare Netzkarte.

Vor allem für das mittelfristige Ziel, den afrikanischen Kontinent als gesamtes in *ATLANTIS* zu integrieren, sind diese beiden Punkte sehr bedeutend.

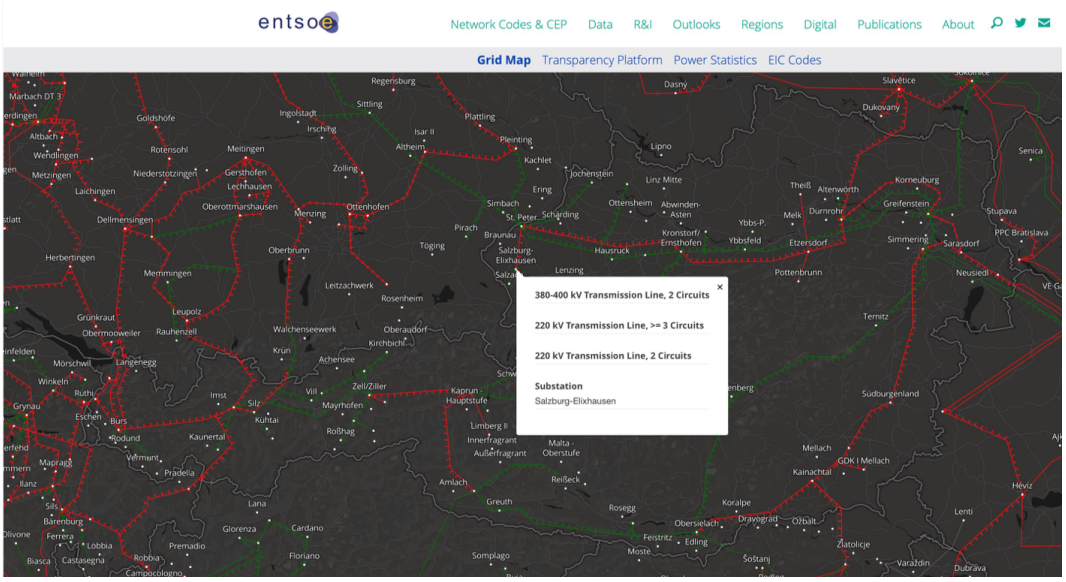


Abbildung 2.1: Netzkarte der *ENTSO-E* [3]

Eine alternative Quelle für Leitungsdaten ist die Datenbasis von *OpenStreetMap*. *OpenStreetMap* lässt sich prägnant wie eine Mischung von *Google Maps* und der *Wikipedia* beschreiben. Dies bedeutet, dass der Datensatz von verschiedenen interessierten Personen erstellt wird, sehr oft aber auch aus automatisierter Analyse von Satellitenbildern.

In Abb. 2.2 ist der *OpenStreetMap* Datensatz des elektrischen Übertragungsnetzes von Österreich abgebildet. Wie man auf den ersten Blick erkennen kann, ist dieser um einiges ausführlich als der der *ENTSO-E* Netzkarte. Jedoch ist dieser Datensatz nicht ohne weiteres in *ATLANTIS* nutzbar. Dies ist aus mehreren Gründen so, zwei davon sind in Abb. 2.3 zu erkennen:

Zum einen sind manche Leitungsstücke ohne sinnvolle Informationen und mit keiner anderen Leitung verbunden (siehe kurze Leitung im Bild links mittig), zum anderen ist das gesamte Leitungsnetz sehr detailliert und realitätsnah. Dies ist von einem Kartendienst natürlich zu erwarten, für *ATLANTIS* sind jedoch nur die Endpunkte einer Leitung und eine allgemein vereinfachte, schematische Darstellung von Interesse.

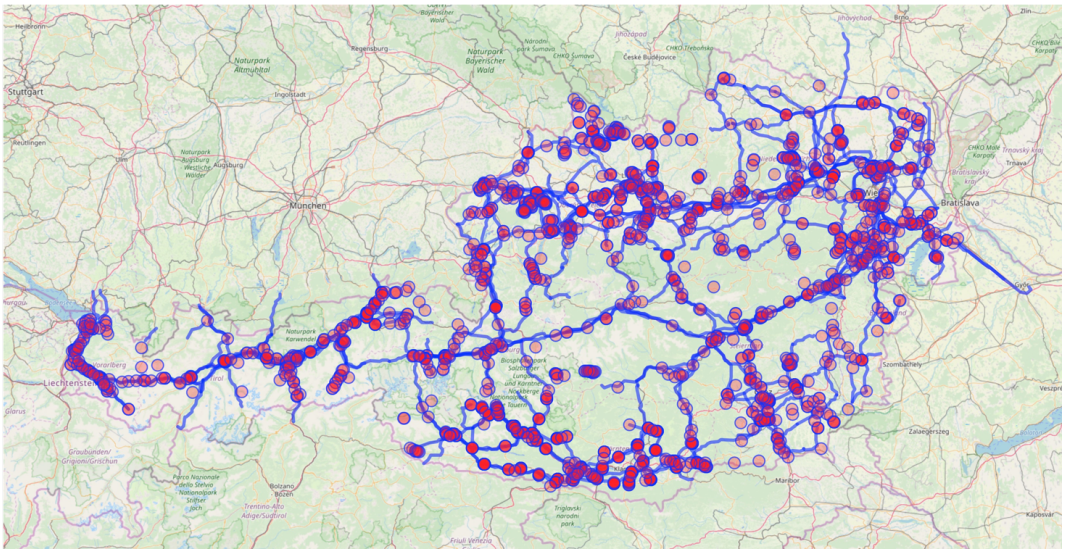


Abbildung 2.2: *OpenStreetMap* - das österreichische Energienetz [4]

In Abb. 2.4 (a) ist zu erkennen, dass vor allem in Umspannwerken die Leitungen einfach enden, jedoch so gut wie nie miteinander verbunden sind. In einer schematischen Darstellung wie in *ATLANTIS* sind diese Knotenpunkte aber essentiell.

In Abb. 2.4 (b) ist die Problematik der Segmentierung und Stichleitungen dargestellt: Auf diesem Ausschnitt sind fünf Leitungen - die Hauptleitung besteht aus drei einzelnen Leitungen - sowie zwei Stichleitungen mit insgesamt vier Knoten abgebildet. Tatsächlich soll die Vereinfachung in diesem Beispiel nur aus zwei Leitungen, welche sich in einem Knotenpunkt treffen, bestehen.

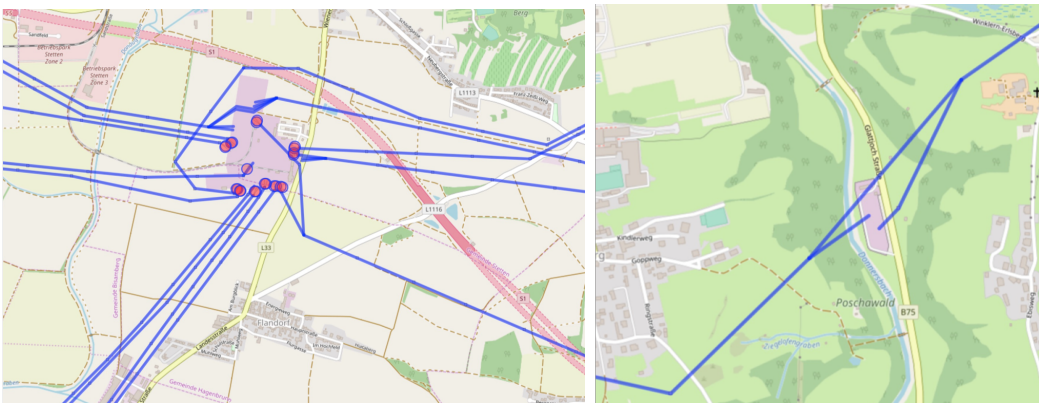
Die Datenbasis von *OpenStreetMap* kann unter anderem im *.json* Dateiformat exportiert werden. Dies ist ein einfaches, kompaktes Dateiformat zum Datenaustausch strukturierter Daten.

Abbildung 2.3: *OpenStreetMap* - Beispiel Detailgrad [4]

.json wurde ursprünglich für *JavaScript* entwickelt, daher auch der volle Name *JavaScript Object Notation*, jedoch sind für praktisch alle verbreiteten Programmiersprachen *Parser* verfügbar, sprich volle Kompatibilität zum Importieren und Exportieren ist fast immer gegeben. In Abb. 2.5 ist ein Ausschnitt aus so einer *.json* Datei dargestellt.

Die Datei ist als Array von Objekten aufgebaut. Ein Objekt ist zwischen zwei geschwungenen Klammern ({}) verschachtelt und enthält verschiedene Eigenschaften wie *type* oder *id*. In der Eigenschaft *tags* ist oft eine Vielzahl nützlicher Zusatzinformationen zu finden, jedoch variiert dies von Datensatz zu Datensatz.

In einem Way-Element Objekt sind immer die einzelnen *nodes* aufgezählt, welche Koordinaten von Knoten des Way-Elements enthalten. Jedes Node-Element besteht wiederum selbst aus einer *id* und den Längen- und Breitengradkoordinaten.



(a) unterschiedliche Endknoten (b) Segmentierung und Stichleitungen

Abbildung 2.4: *OpenStreetMap* - Beispiel Problemstellen [4]


```
{
  "type": "way",
  "id": 23049205,
  "nodes": [
    248719559,
    248719560,
    384720866,
    248719561
  ],
  "tags": {
    "cables": "6",
    "name": "Bisamberg – Gerasdorf",
    "operator": "Wien Energie GmbH",
    "power": "line",
    "ref": "100/51;100/61",
    "voltage": "110000",
    "wires": "single"
  }
},
{
  "type": "way",
  "id": 23049236,
  "nodes": [
    248719562,
    248719793,
    248719794,
    248719795,
    248719796,
    248719797,
    248719798,
```


(a) Way-Elemente


```
{
  "type": "node",
  "id": 954609926,
  "lat": 48.6759654,
  "lon": 16.8468103
},
{
  "type": "node",
  "id": 954609950,
  "lat": 48.6732778,
  "lon": 16.8469028
},
{
  "type": "node",
  "id": 954609979,
  "lat": 48.6704539,
  "lon": 16.8469943
},
{
  "type": "node",
  "id": 954609899,
  "lat": 48.6790593,
  "lon": 16.8466974
},
{
  "type": "node",
  "id": 954609983,
  "lat": 48.680958,
  "lon": 16.8431096
},
```


(b) Node-Elemente

Abbildung 2.5: *OpenStreetMap* - Export im *.json* Dateiformat

Zusammengefasst kann gesagt werden, dass der Datensatz von *OpenStreetMap* für eine Verwendung in *ATLANTIS*

- zu umfangreich und detailliert,
- voll von unwichtigen Daten wie Koordinaten von Freileitungsmasten,
- zu stark segmentiert,
- nicht immer zusammenhängend,
- für den Menschen nur schlecht lesbar und
- gar nicht lesbar für *ATLANTIS* ist, sowie
- Leitungen in Knotenpunkte unterschiedliche Endpunkte aufweisen.

Um dennoch diese wertvolle Datenbasis nutzen zu können, war die Notwendigkeit gegeben, ein Programm zu entwickeln, welches eben jenen Datensatz für *ATLANTIS* nutzbar macht. An dieses Programm wurden folgende Anforderungen gestellt:

- Universal auf sämtliche Netzebenen aller Länder einsetzbar,
- gute Performance auch bei sehr großen Datenmengen,
- Robustheit bei variierenden Daten,
- Abfangen und Lösung verschiedenster auftretender Fehler,
- einfache Fehlersuche,
- übersichtliche Struktur für zukünftige Erweiterungen,
- sinnvolle Visualisierungen der Daten sowie
- klare Warnungen und Bereitstellung wichtiger Zusatzinformationen.

Als Programmiersprache wurde MATLAB gewählt. Das Programm MATLAB ist für die Verarbeitung von großen Datenmengen sehr gut geeignet, weist zahlreiche Visualisierungsmöglichkeiten auf und ist durch die weite Verbreitung im wissenschaftlichen Bereich gut akzeptiert.

3 MATLAB - Hauptprogramm

Das MATLAB Programm besteht im Eigentlichen aus 6 Modulen. Der Ablauf dieser Module ist in Abb. 3.1 schematisch dargestellt.

Diese 6 Module sind wiederum in insgesamt 23 Funktionen unterteilt. Dies stellt sicher, dass folgende Anforderungen zutreffen:

- Übersichtlichkeit im Programmablauf
- Übersichtlichkeit im Variablen-Workspace
- Einfache Wartbar- und Erweiterbarkeit
- Visualisierungen optional ein- oder ausschaltbar

Um diese Anforderungen zu erreichen, wurde Wert auf folgende Eigenschaften gelegt:

- Ausführliche Kommentare und Beschreibungen
- Programmcode und Kommentare in Englisch
- selbsterklärende Variablennamen
- nur eine einzelne Datei, keine externen Funktionen o.ä.
- Verzicht auf komplexes GUI mit unverständlichem Programmcode
- informativer Output in der Konsole mit aktuellem Zwischenstand des Programms

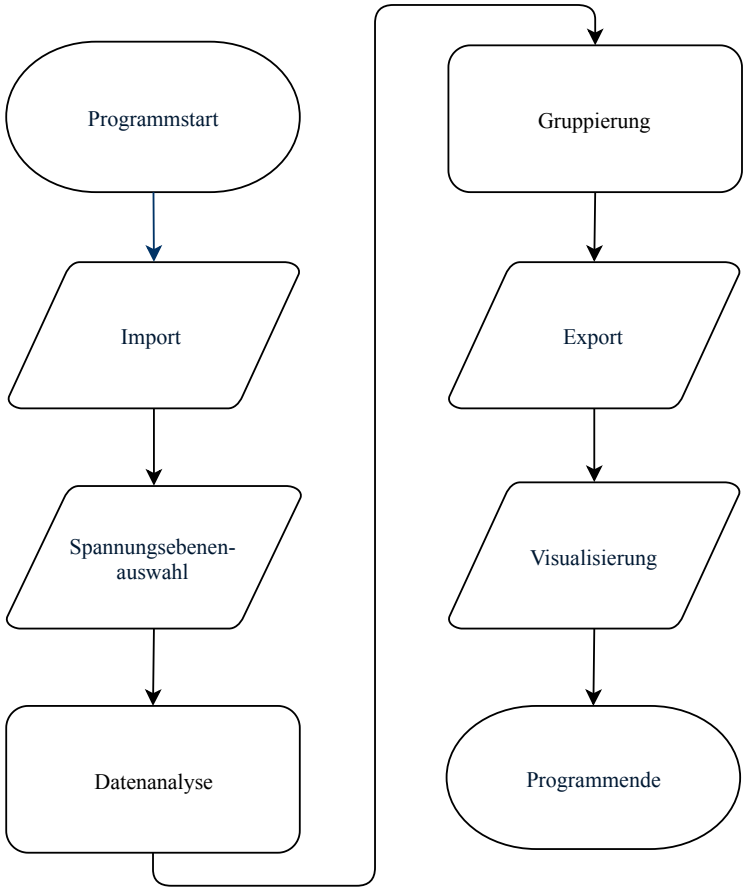


Abbildung 3.1: Schematischer Aufbau des gesamten MATLAB Skripts

Vor den eigentlichen 6 Modulen wird das MATLAB Skript initialisiert und einige Parameter können angepasst werden.

Im ersten Modul werden die Daten importiert, im zweiten werden lediglich die gewünschten Spannungsebenen ausgewählt, damit kann sich die zu verarbeitende Datenmenge bereits erheblich reduzieren.

Im dritten Modul werden die Daten nach bestimmten Aspekten analysiert, sowohl um die Daten nach wichtigen Informationen zu sortieren, als auch den Umfang erneut zu reduzieren.

Im vierten Modul, dem Herzstück, werden Endpunkte der jeweiligen Leitungen sinnvoll zusammen gruppiert und damit verbunden.

Im vorletzten Modul werden die Daten für den Export aufbereitet und letztendlich auch in Excel-Dateien exportiert.

Ebenso ist es möglich, im letzten Modul sämtliche Visualisierungen an- bzw. abzuschalten.

3.1 Initialisierung

Im Teil *Initialisierung* wird das Skript initialisiert, indem sämtliche bereits geöffneten Fenster geschlossen, der Inhalt der Konsole entfernt und bis auf wenige Ausnahmen die Variablen im Workspace gelöscht werden.

```
%% OSM to ATLANTIS Importool - V1.0
%
% ABSTRACT
% This skript uses map data downloaded from the open street map project,
% imports it to matlab, processes, visualizes and cleans up the data and
% exports it in ATLANTIS-readable Excel files. The data contains information
% and coordinates of power lines of the electric grid of any selected country.
%
% INSTRUCTIONS TO IMPORT DATA
% Go to https://overpass-turbo.eu, click "Wizard" and create a search query
% like "power=line and voltage=* in "Austria"". The first part specifies
% overland power transmission lines, the second the voltage level ("*" means
% that any voltage level is accepted) and the third the country code.
% Click "run" and if applicable "continue anyway". If necessary, raise the
% "[timeout: xx]" limit. Then export data as "raw OSM data" to a "*.json" file.
%
% DESCRIPTION
% The ".json" file will be imported as "data_raw", all elementes will be
% seperated in "data_ways_all" and a "data_nodes_all".
% For easiser visualisations, x/y coordinates in [km] will be calculated from
% the given lat/lon coordinates, the origin will be the meanvalue of all
% coordinates, aka "center of gravity".
% Coordinates (x/y and lat/lon) of endnodes will be added from "data_nodes_all"
% to "data_ways_all" and the length of each line will be calculated and added.
% If a line has more than one voltage level, it will be cloned accordingly.
```

```
26 % The user than can select voltage levels, this data ("data_ways_selected")
27 % will be processed from now on.
28 % Lines which are short busbars in substations will be deleted and lines
29 % which could be DC lines will be marked. If a line contains more than three
30 % cables, it may be cloned accordingly.
31 % Endpoints which have the same coordinates will be fused, endpoints which are
32 % in a certain radii to each other will be grouped together - for each occuring
33 % voltage level there will be one seperate unique node (NUID), but they all
34 % have the same coordinates.
35 % All remaining valid ways will be exported in two Excel files.
36 %
37 %
38 % CREDITS
39 % (C) created as master thesis by Lukas Frauenlob, October 2019, IEE, TU Graz
40 %
41 % CHANGELOG:
42 % V0.1 - 0.9: Creationprocess while working on master thesis
43 % V1.0 - 15. 10. 2019, final version as described in master thesis
44 % V1.X - Bugfixes/added features, please describe them here
45
46 %% Initialize script
47 close all
48 clc
49 clearvars -except data_raw voltage_levels_selected
50 overallruntime = tic;
```

Matlab Code 3.1: Hauptprogramm - Initialisierung

In den ersten Zeilen des Skript wird in Form von Kommentaren in einer groben Kurzfassung der Funktionsumfang des Skripts umrissen (»ABSTRACT«), eine Erklärung geliefert, wie die Daten aus *OpenStreetMap* exportiert werden können (»INSTRUCTIONS TO IMPORT DATA«), eine genauere Beschreibung des Programmablaufes (»DESCRIPTION«) sowie Hinweise auf den Ersteller (»CREDITS«) und Versionsverlauf (»CHANGELOG«).

Mit *close all* werden sämtliche offenen Visualisierungen geschlossen, *clc* leert die Konsole und mit *clearvars* werden fast alle Variablen aus dem Workspace gelöscht, Ausnahmen sind lediglich *data_raw* sowie *voltage_levels_selected*. Dies wird in Abschnitt 3.4 sowie 3.5 näher erklärt.

Die erste Ausnahme *data_raw* sorgt dafür, dass die Auswahl der *.json* Datei von *OpenStreetMap* beim erneuten Durchlauf des Programms nicht erneut ausgewählt werden muss, und die zweite Ausnahme *voltage_levels_selected* nicht zu einer erneuten Auswahl der Spannungsebenen führt.

Die MATLAB Befehle *tic* und *toc* messen die Zeit der Ausführung von Programmcode zwischen diesen beiden Befehlen. Dies wird in jeder Funktion angewendet und die Laufzeit je Funktion immer in der Konsole dargestellt.

Um auch die gesamte Laufzeit des Programms messen zu können, wird der Wert von *tic* in die Variable *overallruntime* gespeichert und am Schluss des Programms in die Konsole geschrieben.

3.2 Einstellungen

In den *Einstellungen* werden wichtige Parameter für das Programm festgelegt.

```
%% Settings
% Two character country code, according to ISO-3166-1, of current country
export_excel_country_code = "AT";

% Set neighbourhood threshold radius to determine, how close endnodes have
% to be together to get grouped
neighbourhood_threshold = 0.5;

% Max. length of a line which can be a type 'busbar', in km
busbar_max_length = 1;

% Display all numbers (up to 15 digits) in console without scientific notation
format long
```

Matlab Code 3.2: Hauptprogramm - generelle Einstellungen

In der Variable *export_excel_country_code* wird der zweistellige Ländercode nach ISO 3166-1 eingestellt. Dieser Ländercode wird genutzt, um im Modul Export (siehe Kapitel 8) die *LtgsID* (siehe MATLAB Code 8.3) und die *NUID* (siehe MATLAB Code 8.4) zu erstellen, welche dazu verwendet werden, Leitungen beziehungsweise Knoten eindeutig in *ATLANTIS* zu identifizieren.

Des Weiteren wird der Ländercode dazu genutzt, den exportierten Excel-Dateien einen eindeutigen Dateinamen in der Funktion *my_export_excel()* (beschrieben in Abschnitt 8.4), zuzuweisen.

In der Variable *neighbourhood_threshold* wird der Gruppierungsradiusschwellwert eingestellt. Dieser Radius wird im Modul Gruppierung (siehe Kapitel 7) genutzt um Endknoten zu gruppieren und ist somit der wichtigste Parameter dieses MATLAB Programms.

Das Kommando *format long* sorgt dafür, dass in der MATLAB Konsole Zahlenwerte bis zu 15 Stellen als Dezimalzahl dargestellt werden. Dies ist notwendig, da die *IDs* der Way-Elemente länger als die in der Standardeinstellung *format short* erlaubten 4 Stellen sind, bevor Zahlen in wissenschaftlicher Notation dargestellt werden [5].

In der Variable *busbar_max_length* wird für die Funktion *my_delete_busbars()* die maximal zugelassen Länge einer Busbar oder einer Bay eingestellt. Die genauere Funktionsweise ist in Abschnitt 6.1 beschrieben. Ein Wert von 1 km hat sich als praktikabel erwiesen.

3.3 Visualisierungseinstellungen

In den *Visualisierungseinstellungen* lassen sich die Visualisierungen ein- oder ausschalten.

```
%%% Recommended visualisations
% Visualize all selected ways, hence the original dataset
bool.plot_ways_original = true;

% Visualize all selected ways, while they are being grouped. This plot
% includes the original and the new ways, including the threshold-circles
bool.plot_ways_grouping = true;

% Visualize all selected ways on map, final dataset with endnodes grouped
bool.plot_ways_final = true;

% Visualize distances between all endnodes to easier set neighbourhood_treshold
bool.histogram_distances_between_endpoints = true;


%%% Optional visualisations, for debugging purposes and in-depth-research
% Visualize length of busbars to set busbar_max_length
bool.histogram_length_busbars = false;

% Visualize how many endnodes are stacked on top of each other
bool.histogram_stacked_endnodes = false;

% Visualize all stacked endnodes on map
bool.plot_stacked_endnodes = false;

% Visualize how many neighouring endnodes are grouped together
bool.histogram_neighbouring_endnodes = false;

% visualize all neighbouring endnodes on map
bool.plot_neighbouring_endnodes = false;
```

Matlab Code 3.3: Hauptprogramm - Visualisierungseinstellungen

Gerade beim Import eines neuen Datensatzes ist es erwünscht, den noch unbekannten Datensatz visuell auf verschiedensten Weisen darzustellen. Gegen Ende des Selektierungsprozesses steht mehr die Laufzeit des MATLAB Programms im Vordergrund. Eine Deaktivierung bereits bekannter Visualisierungen ist daher sinnvoll.

Die insgesamt 9 möglichen Visualisierungen lassen sich in zwei Gruppen einteilen, empfohlene und optionale Visualisierungen.

Die empfohlene Visualisierungen beinhalten die drei geographischen Visualisierungsfunktionen

1. *my_plot_ways_original()*, beschrieben in Abschnitt 9.1,
2. *my_plot_ways_grouping()*, beschrieben in Abschnitt 9.2 und
3. *my_plot_ways_final()*, beschrieben in Abschnitt 9.3

sowie die Visualisierung der Distanzen zwischen Endpunkten, welche näher in Abschnitt 7.1 beschrieben wird.

Am wichtigsten davon sind die Visualisierungen *bool.plot_ways_grouping* sowie *bool.histogram_distances_between_endpoints* um den Wert der Variable *neighbourhood_threshold* einstellen zu können.

Die optionalen Visualisierungen enthält mit *bool.histogram_length_busbars* eine Darstellung der Länge aller als solcher deklarierten *busbars* und *bays*, wie näher in Funktion *my_delete_busbars()* in Abschnitt 6.1 beschrieben wird. Dies dient dazu den Wert der Variable *busbar_max_length* adjustieren zu können.

Die restlichen vier Visualisierungen stellen Endpunkte, welche als übereinander- bzw. nebeneinanderliegend im Modul *Gruppierung* (siehe Kapitel 7) erkannt werden.

3.4 Modul 1/6 - Import

Das Hauptprogramm startet mit dem erstem Modul, dem Modul *Import*. Zu Beginn wird eine Begrüßung und Informationsnachricht in der Konsole ausgegeben.

```
%% Main Program
% Print welcome message, selected country code and neighbouring_threshold
fprintf(['WELCOME to OpenStreetMap to ATLANTIS Importtool V1.0! \n' ...
         '(C) created by Lukas Frauenlob, IEE, TU Graz, October 2019 ' ...
         '\n\n\n--- Info ---\n' ...
         '    ... to restart data import, please delete variable' ...
             '"data_raw". \n' ...
         '    ... to restart voltage level selection, delete ' ...
             '"voltage_levels_selected". \n' ...
         '    ... please check if visualisations are toggled on/off for ' ...
             'either \n' ...
         '        performance improvements or additional information!\n\n\n' ...
         '--- Settings --- \n' ...
         '    ... Country code for Excel output: "%s" \n' ...
         '    ... Neighbouring (=grouping circle) threshold: %5.2f km ' ...
         '\n\n\n'], export_excel_country_code, neighbourhood_threshold)
```

Matlab Code 3.4: Hauptprogramm - Information

In dieser Nachricht werden generelle Information, sowie die aktuelle Version des MATLAB Programms, dargestellt.

Eine kurze Bedienungsanleitung wird im Bereich *Info* angezeigt, unter *Settings* werden die wichtigsten Einstellungen der Variablen *export_excel_country_code* sowie *neighbourhood_threshold* dargestellt.

Das eigentlichen Modul Import besteht aus drei Funktionen und sorgt dafür, dass der von *OpenStreetMap* exportierte Datensatz in MATLAB genutzt werden kann und im MATLAB Workspace in einem sinnvollen Format zur Verfügung steht.

```
%%% Import Data
fprintf('--- Import data (Step 1/6) --- \n')

% If data wasn't imported yet, open UI, select json.file and import it
if not(exist('data_raw', 'var'))
    [data_raw, file_name, file_path] = my_import_json();

    % When importing new data (possibly from another country),
    % delete old voltage_levels_selected to force new vlevel selection
    clearvars voltage_levels_selected
end

% Separate all 'node' and 'way' elements to seperate variables and add UID
[data_nodes_all, data_ways_all] ...
    = my_seperate_raw_data_add_UID(data_raw);

% Add the lat/lon & X/Y coordinates and way lengths to all ways
[data_ways_all, degrees_to_km_conversion] ...
    = my_add_coordinates(data_ways_all, data_nodes_all);
```

Matlab Code 3.5: Hauptprogramm - Modul 1/6

Die Funktionsweise der drei einzelnen Funktionen wird genauer in Kapitel 4 beschrieben.

Die Funktion *my_import_json()* (siehe Abschnitt 4.1), wird jedoch nur beim ersten Durchlauf des Programms aufgerufen, damit nicht jedes Mal die *.json* Datei neu ausgewählt werden muss. Siehe dazu auch Abschnitt 3.1.

3.5 Modul 2/6 - Spannungsebenenauswahl

Im zweiten Modul des MATLAB Programms, dem Modul *Spannungsebenenauswahl*, wird dem Benutzer ermöglicht, nur Spannungsebenen, welche von Interesse sind, auszuwählen.

```
%%% Select voltage levels
fprintf('\n--- Select voltage levels (Step 2/6) --- \n')

% Count the number of lines with a specific voltage level, display and add it
[data_ways_all, voltage_levels_sorted] ...
    = my_count_voltage_levels(data_ways_all);

% Open a dialog to ask the user to select voltage levels
if not(exist('voltage_levels_selected', 'var'))
    voltage_levels_selected ...
        = my_ask_voltage_levels(voltage_levels_sorted);
end

% Save all ways which match selected voltage levels
data_ways_selected ...
    = my_select_ways(data_ways_all, voltage_levels_selected)
```

Matlab Code 3.6: Hauptprogramm - Modul 2/6

Die Funktionsweise der drei einzelnen Funktionen wird genauer in Kapitel 5 beschrieben.

Die Funktion *my_ask_voltage_levels()* (siehe Abschnitt 5.2), wird jedoch nur beim ersten Durchlauf des Programms aufgerufen, damit nicht jedes Mal die Spannungsebenen neu ausgewählt werden müssen. Siehe dazu auch Abschnitt 3.1.

3.6 Modul 3/6 - Datenanalyse

Im dritten Modul des MATLAB Programms, dem Modul *Datenanalyse*, wird der Datensatz nach bestimmten Kriterien untersucht. Die Funktionsweise der drei einzelnen Funktionen wird genauer in Kapitel 6 beschrieben.

```
%%% Analyse data
fprintf('\n--- Analyse data (Step 3/6) --- \n')

% Find all ways with type busbars, extract them and delete them
[data_ways_selected, data_busbars] ...
    = my_delete_busbars(data_ways_selected, bool, busbar_max_length);

% Detect all possible DC lines
[data_ways_selected, dc_candidates] ...
    = my_count_possible_dc(data_ways_selected);

% Count how many cables a way has (needed to double or triple a way), add flags
data_ways_selected ...
    = my_count_cables(data_ways_selected);
```

Matlab Code 3.7: Hauptprogramm - Modul 3/6

3.7 Modul 4/6 - Gruppierung

Im vierten Modul des MATLAB Programms, dem Modul *Gruppierung*, werden Endpunkte nach bestimmten Kriterien zusammengefasst.

```
%%% Group nodes
fprintf('\n--- Group nodes (Step 4/6) --- \n')

% Calculate distances between all endpoints
distances_between_nodes ...
    = my_calc_distances_between_endpoints(data_ways_selected, ...
                                          degrees_to_km_conversion, bool);
```

```
11 % Calculate all stacked nodes
12 [data_ways_selected, nodes_stacked_pairs] ...
13     = my_calc_stacked_endnodes(data_ways_selected, distances_between_nodes, ...
14                                bool);
15
16 % Calculate neighbouring nodes
17 [data_ways_selected, nodes_neighbouring_pairs] ...
18     = my_calc_neighbouring_endnodes(data_ways_selected, ...
19                                     distances_between_nodes, ...
20                                     neighbourhood_threshold, bool);
21 % Group stacked nodes
22 nodes_stacked_grouped ...
23     = my_group_nodes(nodes_stacked_pairs);
24
25 % Group neighbouring nodes
26 nodes_neighbouring_grouped ...
27     = my_group_nodes(nodes_neighbouring_pairs);
28
29 % Add coordinates of stacked endnodes
30 data_ways_selected ...
31     = my_group_stacked_endnodes(data_ways_selected, nodes_stacked_grouped);
32
33 % Add coordinates of neighbouring endnodes
34 [data_ways_selected, grouped_xy_coordinates] ...
35     = my_group_neighbouring_endnodes(data_ways_selected, ...
36                                      nodes_neighbouring_grouped, ...
37                                      degrees_to_km_conversion);
38
39 % Add final coordinates, hence select from original or grouped coordinates
40 data_ways_selected ...
41     = my_add_final_coordinates(data_ways_selected);
```

Matlab Code 3.8: Hauptprogramm - Modul 4/6

Die Funktionsweise der sieben einzelnen Funktionen wird genauer in Kapitel 7 beschrieben.

Die Funktion *my_group_nodes()* (siehe Abschnitt 7.4) wird zwei Mal aufgerufen. Einmal werden darin übereinander- und einmal nebeneinanderliegende Endpunkte zusammengefasst.

3.8 Modul 5/6 - Export

Im fünften Modul des MATLAB Programms, dem Modul *Export*, wird der Datensatz für den Export als *Excel*-Dateien vorbereitet und schlussendlich auch als solche abgespeichert. Die Funktionsweise der vier einzelnen Funktionen wird genauer in Kapitel 8 beschrieben.

```
%%% Export
fprintf('\n--- Export (Step 5/6) ---\n')

% Delete ways which have identical endpoints
data_ways_selected ...
    = my_delete_singular_ways(data_ways_selected);

% Copy all tags of all ways into a seperate variable
data_ways_selected_tags ...
    = my_get_tags(data_ways_selected);

% Add LtgsID and duplicate ways if necessary
data_ways_selected ...
    = my_add_LtgsID_clone_ways(data_ways_selected, export_excel_country_code);

% Export data to excel files, add NUID
data_ways_selected ...
    = my_export_excel(data_ways_selected, export_excel_country_code, ...
                      data_ways_selected_tags);
```

Matlab Code 3.9: Hauptprogramm - Modul 5/6

3.9 Modul 6/6 - Visualisierungen

Im sechsten und letzten Modul des MATLAB Programms, dem Modul *Visualisierungen*, werden die Daten im Originalzustand, während dem Gruppieren und im Finalzustand visuell dargestellt.

```
%%% Visualisations
fprintf('\n--- Visualisations (Step 6/6) ---\n')

% Plot original ways
my_plot_ways_original(data_ways_selected, data_busbars, ...
                      voltage_levels_selected, bool);

% Plot ways while grouping endnodes
my_plot_ways_grouping(data_ways_selected, data_busbars, ...
                       grouped_xy_coordinates, neighbourhood_threshold, bool);

% Plot final ways
my_plot_ways_final(data_ways_selected, voltage_levels_selected, bool);


fprintf(['\n\nOverall runtime of program: %f seconds. \n' ...
         'GOOD BYE! :) \n \n'], toc(overallruntime))
clear overallruntime
```

Matlab Code 3.10: Hauptprogramm - Modul 6/6

Die Funktionsweise der drei einzelnen Funktionen wird genauer in Kapitel 9 beschrieben.

Die Funktion *my_plot_ways_original()* (siehe Abschnitt 9.1) wird bewusst erst am Schluss dieses Programms aufgerufen.

Die eigentlichen Intention, den Datensatz »so wie er ist« darzustellen, stößt bei umfangreichen Datensätzen schnell an die Grenze der Visualisierungsfunktionen von MATLAB. MATLAB ist primär dazu entwickelt worden, mit großen Datenmengen zu rechnen, nicht um sehr große Datenmengen visuell effizient darzustellen.

Da die Funktion *my_delete_busbars()*, beschrieben in Abschnitt 6.1, sowie die Funktion *my_delete_singular_ways()*, beschrieben in Abschnitt 8.1, bereits Modul 3 respektive 5 aufgerufen wurde, wurde der Datensatz bereits beträchtlich von kleinen, kurzen Way-Elementen bereinigt, womit eine Visualisierung mit vertretbaren Rechenaufwand ermöglicht wird.

4 Funktionen - Modul Import

Der erste von sechs Programmmodulen ist das *Import-Modul*. In diesem werden die Daten, welche von *OpenStreetMap* exportiert wurden, in MATLAB geladen, im Workspace in geeignete Dateiformate konvertiert, miteinander kombiniert und als Variable *data_ways_all* dem nächsten Modul übergeben.

Der Aufbau des *Import-Moduls* ist schematisch in Abb. 4.1 dargestellt.

Das *Import-Modul* besteht aus drei Funktionen:

1. *my_import_json()*, beschrieben in Abschnitt 4.1,
2. *my_seperate_raw_data_add_UID()*, beschrieben in Abschnitt 4.2 und
3. *my_add_coordinates()*, beschrieben in Abschnitt 4.3.

Die erste Funktion, *my_import_json()*, öffnet ein Fenster zur Dateiauswahl der *.json* Exportdatei und konvertiert die Daten als Variable *data_raw* in den MATLAB Workspace. Ebenso wird der Dateipfad und Dateiname in den Workspace übergeben.

Die Funktion *my_seperate_raw_data_add_UID()* vergibt jedem Way-Element eine einzigartige Identifikationsnummer, die *UID - unique identifier*.

Jedes Way-Element wird dadurch eindeutig benannt und ist somit im gesamten Programmablauf einwandfrei rückverfolgbar. Zudem teilt die Funktion die gemischten Way- und Node-Elemente in zwei Variablen, nämlich *data_nodes_all* und *data_ways_all*, auf.

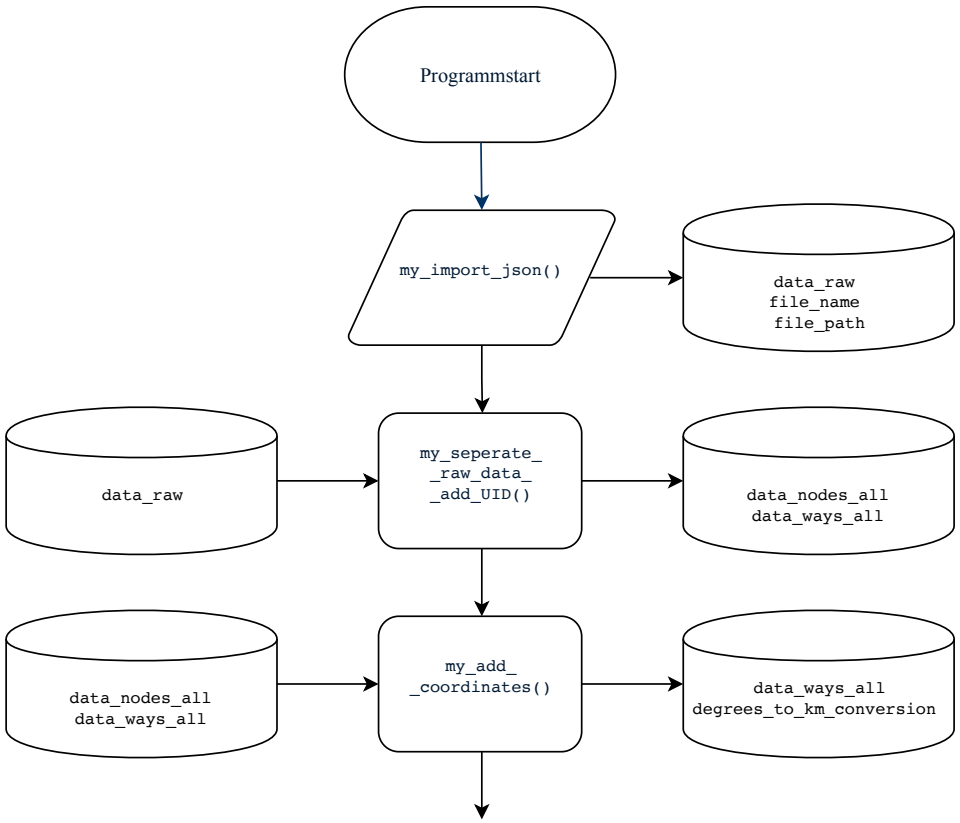
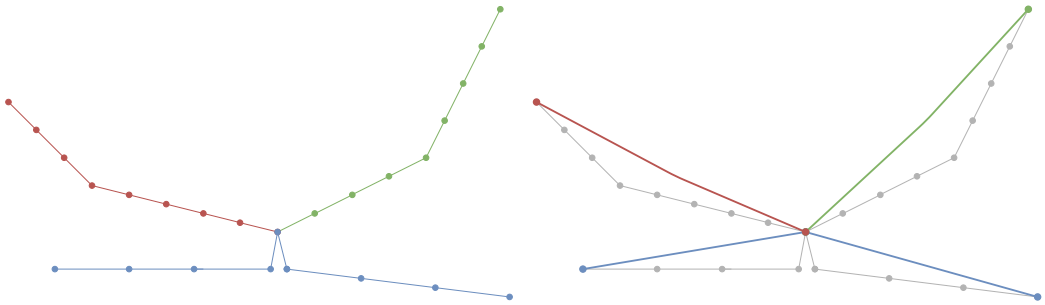


Abbildung 4.1: Schematischer Aufbau Modul 1 von 6

Diese beiden Variablen werden von der Funktion *my_add_coordinates()* übernommen und lediglich die relevanten Koordinaten aus *data_nodes_all* eben jenen Way-Elementen in *data_ways_all* als neue *fields* zugewiesen.

Da für *ATLANTIS* die Koordinaten der einzelnen Hochspannungsmasten einer Leitung irrelevant sind, werden diese ignoriert und nur die relevanten Anfangs- und Endknotenkoordinaten erhalten. Dies ist anschaulich in Abb. 4.2 dargestellt.

Da im Zuge dessen sämtliche Längen- und Breitengradinformationen als in *x/y* konvertierte Distanzwerte hinzugefügt werden, werden die dazu notwendigen Informationen in der Variable *degrees_to_km_conversion* ebenfalls



(a) Ausgangsdatensatz (b) Datensatz nur mit Endpunkten

Abbildung 4.2: Schematische Visualisierung der Auswahl der Endpunkte

exportiert, um später in anderen Funktionen erneut genutzt werden zu können.

4.1 my_import_json()

Die Funktion *my_import_json()* importiert die Daten aus *OpensStreetMap* in den MATLAB Workspace.

```
1  function [data_raw, file_name, file_path] ...
2      = my_import_json()
3      % DESCRIPTION
4      % This function opens an UI to select a *.json file. With the given
5      % file name and file path the *.json file will be converted to a cell
6      % object. Unnecessary header files, which will be created by overpass,
7      % will be deleted.
8      %
9      % INPUT
10     % (none)
11     %
12     % OUTPUT
13     % data_raw ... all data from the imported *.json file as cell array
14     % file_name ... name of file
15     % file_path ... path of file
16
17     disp('Start importing Data (*.json file)...')
```

```
18      % Open UI to select file
19      [file_name, file_path] = uigetfile('*.json');
20      tic
21
22      % Print file path and filename to console
23      fprintf('   ... file path: %s \n   ... file name: %s \n', ...
24              file_path, file_name)
25
26      % Import and decode selected .json file into workspace
27      data_raw_jasonimport = jsondecode(fileread([file_path, file_name]));
28
29      % Strip unnecessary header data from export file,  keep relevant elements
30      data_raw = data_raw_jasonimport.elements;
31
32      fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
33  end
```

Matlab Code 4.1: Funktion *my_import_json()*

OpensStreetMap exportiert Auszüge aus der Datenbank beispielsweise über die Website [4] im *.json* Dateiformat. Dieses muss zuerst von MATLAB eingelesen werden.

Dazu wird in Zeile 19 mit dem Befehl *uigetfile* ein Dateiauswahlfenster geöffnet, welches lediglich Dateien mit der Endung *.json* akzeptiert. Der Dateipfad und Dateiname wird in Zeile 23 in der Konsole ausgegeben.

Mithilfe der MATLAB-eigenen Funktion *jsondecode* wird die *.json* Datei als *Cell-Object* in den MATLAB Workspace importiert.

Da sämtliche interessanten Objekte, sprich alle Way- und Node-Elemente, Teil des *element arrays* sind, werden lediglich diese Daten in Zeile 30 importiert und somit der überflüssige Overhead, welcher von [4] der *.json* Datei hinzugefügt wurden, entfernt.

Die Variable *data_raw* wird von der Funktion zurückgegeben.

4.2 my_seperate_raw_data_add_UID()

In der Funktion *my_seperate_raw_data_add_UID()* werden Way- und Node-Elemente separiert und die *UID* erstellt.

```
function [data_nodes_all, data_ways_all] ...
    = my_seperate_raw_data_add_UID(data_raw)

    % DESCRIPTION
    % This function importes the raw data, looks for 'node' and 'way' elements
    % and seperates them from raw data to save them in seperate variables
    % with type "struct array". If the data exported from OSM has corrupted
    % elements (hence, a field like "tags" is missing), this element will be
    % ignored. A manual review of the *.json file will then be necessary.
    % An unique identifier number (UID) will be created and added to
    % every way element.
    %
    % INPUT
    % data_raw ... imported json data as cell array
    %
    % OUTPUT
    % data_nodes_all ... all node elements as struct array
    % data_way_all ... all way elements as struct array

    tic
    disp(['Start seperating raw data into way- ' ...
          'and node-elements... (takes a few seconds)'])

    % preallocation of counter variables
    num_node_elements = 0;
    num_way_elements = 0;

    % Seperate nodes and ways elements from raw data
    for i_raw_element = 1 : numel(data_raw)

        % check if current element is a node element
        if strcmp(data_raw{i_raw_element, 1}.type, 'node')

              % increase node-counter
              num_node_elements = num_node_elements + 1;

              % copy it to "data_nodes"
              data_nodes_cell{num_node_elements, 1} ...
                  = data_raw{i_raw_element, 1};


```

```
43          % check if element is a way-element
44          elseif strcmp(data_raw{i_raw_element, 1}.type, 'way')
45
46              % increase ways-counter
47              num_way_elements = num_way_elements + 1;
48
49              % copy it to "data_ways"
50              data_ways_cell{num_way_elements, 1} = data_raw{i_raw_element, 1};
51          end
52      end
53
54      % Try to convert the two cell variables to struct variables
55      try
56          % Convert from class cell to struct
57          data_ways_all = cell2mat(data_ways_cell);
58          data_nodes_all = cell2mat(data_nodes_cell);
59
60      % if error (fields dont match) remove structs which have more/less fields
61      catch
62
63          % Count number of fields for each struct in data_ways_cell
64          % and save it in a lookup table
65          for i_ways_cell = 1 : numel(data_ways_cell)
66              num_of_fields_ways(i_ways_cell) ...
67                  = numel(fieldnames(data_ways_cell{i_ways_cell})');
68          end
69
70          % Count number of fields per struct in data_nodes_cell
71          % and save it in a lookup table
72          for i_nodes_cell = 1 : numel(data_nodes_cell)
73              num_of_fields_nodes(i_nodes_cell) ...
74                  = numel(fieldnames(data_nodes_cell{i_nodes_cell})');
75          end
76
77          % create boolean index, which fields have "average" number of fields
78          b_num_of_fields_ok_ways ...
79              = num_of_fields_ways == median(num_of_fields_ways);
80          b_num_of_fields_ok_nodes ...
81              = num_of_fields_nodes == median(num_of_fields_nodes);
82
83          % If at least one field is not ok, hence has more or less fields
84          if any(not(b_num_of_fields_ok_ways))
85
86              % Print error message
87              fprintf(['   ATTENTION! There is at least one way element which'...
88                       ' has a different amount of fields. \n              ' ...
89                       'It wont be imported, please debug this function to '...
90                       'find out more! \n'])
91
92              % delete all these elements and continue
93              data_ways_cell = data_ways_cell(b_num_of_fields_ok_ways);
94          end
```

```
95          % Do the same with nodes
96          if any(not(b_num_of_fields_ok_nodes))
97
98              % Print error message
99              fprintf(['   ATTENTION! There is at least one node element which'...
100                      ' has a different amount of fields. \n              ' ...
101                      'It wont be imported, please debug this function to '...
102                      'find out more! \n'])
103
104             % delete all these elements and continue
105             data_nodes_cell = data_nodes_cell(b_num_of_fields_ok_nodes);
106         end
107
108         % Convert from class cell to struct
109         data_ways_all = cell2mat(data_ways_cell);
110         data_nodes_all = cell2mat(data_nodes_cell);
111     end
112
113     % Create unique ID (UID) and add it
114     for i_way_element = 1 : numel(data_ways_all)
115         data_ways_all(i_way_element).UID = i_way_element;
116     end
117
118     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
119 end
```

Matlab Code 4.2: Funktion *my_seperate_raw_data_add_UID()*

Von der vorhergehenden Funktion *my_import_json()* wird die Variable *data_raw* importiert. Zählvariablen werden ab Zeile 25 initialisiert und ab Zeile 29 beginnt der Hauptteil der Funktion: Mit der Zählvariable *i_raw_element* wird jedes einzelne Element aus *data_raw* aufgerufen.

In der *if*-Abfrage in Zeile 32 wird geprüft, ob das aktuelle Element im *field* »type« den Wert »node« hat, sprich ein Node-Element ist. Ist dies der Fall, wird die Zählvariable *num_node_elements* inkrementiert und das Element dem Cell-Array *data_nodes_cell* hinzugefügt.

Schlägt diese Überprüfung fehl, wird das selbe ab Zeile 44 mit dem Wert »way« überprüft. Im positiven Fall wird dieses Element dem Cell-Array *data_ways_cell* hinzugefügt.

Da der MATLAB Dateityp *struct array* in diesem Anwendungsfall besser handhabbar ist als der Dateityp *cell array*, werden die beiden Variablen *data_nodes_cell* und *data_ways_cell* in den Zeilen 57 und 58 in *struct arrays* umgewandelt.

In einigen wenigen Datensätzen trat der Fehler auf, dass nicht jedes Way- bzw. Node-Element immer die selben *fields* aufweist. Dies führt in den Zeilen 57 oder 58 zu einer Fehlermeldung und damit zu einem Abbruch des Programms. Ab Zeile 61 wird diese Fehler abgefangen:

Zuerst werden ab Zeile 65 für jedes Element die Anzahl der *fields* gezählt und in den Tabelle *num_of_fields_ways* und *num_of_fields_nodes* gespeichert. Zahlenwerte, die dem Medianwert, also die häufigst vorkommende Anzahl an Feldern, entsprechen, werden ab Zeile 78 berechnet und entsprechende Elemente in *num_of_fields_ways* und *num_of_fields_nodes* damit indiziert.

Sollte auch nur ein Element eine inkorrekte Anzahl an Feldern haben, wird dies ab Zeile 84 bzw. 96 mit einer Fehlermeldung quittiert und sämtliche fehlerhaften Elemente in Zeile 93 bzw. 105 aus dem Datensatz entfernt und die Operationen aus den Zeilen 57 und 58 in 109 und 110 wiederholt.

Die *UID* wird in der For-Schleife ab Zeile 114 berechnet und einem jedem Way-Element im *field* »UID« zugewiesen.

Die Funktion *my_seperate_raw_data_add_UID()* gibt die Variablen *data_nodes_all* und *data_ways_all* zurück.

4.3 my_add_coordinates()

In der Funktion *my_add_coordinates()* werden den Way-Elemente die Koordinaten der Endknoten hinzugefügt.

```
function [data, degrees_to_km_conversion] ...
    = my_add_coordinates(data, data_nodes_all)

    % DESCRIPTION
    % The first and last node IDs, hence the endpoints, will be extracted
    % from every way element and the correspondending lon/lat coordinates
    % will be added to every way element. Since lon/lat coordniates don't
    % give an intutive feeling of distances in a plot, x/y coordnates in km
    % will be calculated. This will be done by a rough (but sufficient)
    % approximation: The midpoint (COG - center of gravity) of all lon/lat
    % coordinates will be calculated and will be the 0-origin of the x/y plane.
    % An approxmation formula calculates the km-per-degree-conversion on this
    % point on earth. From every endpoint the latitudinal/longitudinal distance
    % to the midpoint will be converted to the x/y km distance, this x/y
    % value will be added to every way element. Using that information,
    % the distance between the endpoints will be calculated and added too.
    %
    % INPUT
    % data ... dataset off all way elements
    % data_nodes_all ... dataset of all node elements
    %
    % OUTPUT
    % data ... the updated dataset off all way elements: IDs of endnodes,
    %          lat/lon coordinates, x/y coordinates, length of line
    % degrees_to_km_conversion ... the necessary information to convert lon/lat
    %                              to x/y coordinates for further use of
    %                              grouped endnodes in another function.

    tic
    disp('Start adding coordinates to each way... (takes a few seconds)')

    % Create a list of all node ids
    list_all_node_IDs = [data_nodes_all(:).id]';

    % Add all endnode coordinates to data
    for i_ways = 1 : numel(data)

        % Add first and last endnode IDs as seperate elements to data
        data(i_ways).ID_node1 = data(i_ways).nodes(1, 1);
        data(i_ways).ID_node2 = data(i_ways).nodes(end, 1);


```

```
43          % Find the position of the endnode id in the list_all_node_IDs
44          position_node1 = find(data(i_ways).ID_node1 == list_all_node_IDs);
45          position_node2 = find(data(i_ways).ID_node2 == list_all_node_IDs);
46
47          % use this position to assign the lon/lat coordinates to data_ways
48          data(i_ways).lon1 = data_nodes_all(position_node1).lon;
49          data(i_ways).lat1 = data_nodes_all(position_node1).lat;
50          data(i_ways).lon2 = data_nodes_all(position_node2).lon;
51          data(i_ways).lat2 = data_nodes_all(position_node2).lat;
52      end
53
54      % Calculate latitudinal/longitudinal midpoint of all coordinates
55      mean_lat = mean([[data.lat1], [data.lat2]]);
56      mean_lon = mean([[data.lon1], [data.lon2]]);
57
58      % Determine if we are on North/South Hemisphere ...
59      if mean_lat > 0
60          fprintf('   INFO: Majority of nodes are on the NORTH and ')
61      else
62          fprintf('   INFO: Majority of nodes are on the SOUTH and ')
63      end
64
65      % ... and East/West Hemisphere, then print this information to console
66      if mean_lon > 0
67          fprintf('EASTERN hemisphere \n')
68      else
69          fprintf('WESTERN hemisphere \n')
70      end
71
72      disp('   ... start calculating and adding x/y coordinates...')
73
74      % at this mean position, calculate how many km approx. equal one degree
75      % source: https://gis.stackexchange.com/questions/75528/...
76      %         understanding-terms-in-length-of-degree-formula/75535#75535
77
78      radians = mean_lat * pi / 180;
79
80      km_per_lon_deg = (111132.954 * cos(1 * radians) ...
81                          - 93.55 * cos(3 * radians) ...
82                          + 0.118 * cos(5 * radians)) / 1000;
83
84      km_per_lat_deg = (111132.92 ...
85                        - 559.82 * cos(2 * radians) ...
86                        +  1.175 * cos(4 * radians) ...
87                        - 0.0023 * cos(6 * radians)) / 1000;
88
89      % calculate the difference in degree for each point from midpoint
90      delta_lon1 = [data.lon1]' - mean_lon;
91      delta_lon2 = [data.lon2]' - mean_lon;
92      delta_lat1 = [data.lat1]' - mean_lat;
93      delta_lat2 = [data.lat2]' - mean_lat;
94
```

```
95      % convert the delta_degree into delta_kilometer, as x1/x2/y1/y2
96      x1 = delta_lon1 * km_per_lon_deg;
97      x2 = delta_lon2 * km_per_lon_deg;
98      y1 = delta_lat1 * km_per_lat_deg;
99      y2 = delta_lat2 * km_per_lat_deg;
100
101     % convert the delta_kilometer into cell arrays to process them in batch
102     x1_cell = num2cell(x1);
103     x2_cell = num2cell(x2);
104     y1_cell = num2cell(y1);
105     y2_cell = num2cell(y2);
106
107     % save delta_kilometer to data_ways
108     [data.x1] = x1_cell{:};
109     [data.y1] = y1_cell{:};
110     [data.x2] = x2_cell{:};
111     [data.y2] = y2_cell{:};
112
113     disp('   ... calculate length of each line and add it...')
114
115     % Calculate distances between endpoints and add it
116     length = num2cell(sqrt((x1 - x2).^2 + (y1 - y2).^2)');
117     [data.length] = length{:};
118
119     % Return the conversion data to use it later again for grouped nodes
120     degrees_to_km_conversion = [km_per_lon_deg, km_per_lat_deg, ...
121                                 mean_lon, mean_lat];
122
123     fprintf('   ... finished! (%5.3f seconds) \n \n' , toc)
124 end
```

Matlab Code 4.3: Funktion *my_add_coordinates*()

Von der vorhergehenden Funktion *my_seperate_raw_data_add_UID()* werden die Variablen *data_nodes_all* und *data_ways_all* importiert.

Um lediglich die Koordinaten der Endpunkte den Way-Elementen hinzuzufügen, werden in Zeile 33 zuerst sämtliche Node-IDs in eine separate Variable *list_all_nodes_IDs* kopiert. Die *for*-Schleife ab Zeile 36 speichert in jedes Way-Element die erste (Zeile 39) und letzte (Zeile 40) Node-ID.

Diese beiden Node-IDs werden in Zeile 44 und 45 in der Variable *list_all_nodes_IDs* gesucht und deren Positionen in *position_node1* bzw. *position_node2* gespeichert.

Mithilfe dieser Positionen können in den Zeilen 49 bis 51 die Längen- und Breitengrade als *lon1, lat1* für den ersten Endknoten, sowie als *lon2, lat2* für den zweiten Endknoten in das jeweilige Way-Element gespeichert werden.

Um nun die *x/y*-Koordinaten berechnen zu können, wird in den Zeilen 55 und 56 der Mittelwert aller Breiten- bzw. Längengrade ermittelt. Dieser *mean_lat* und *mean_lon* Punkt entspricht somit dem Schwerpunkt dieser Fläche und liegt mit hoher Wahrscheinlichkeit nicht auf einer bereits bestehenden Koordinate.

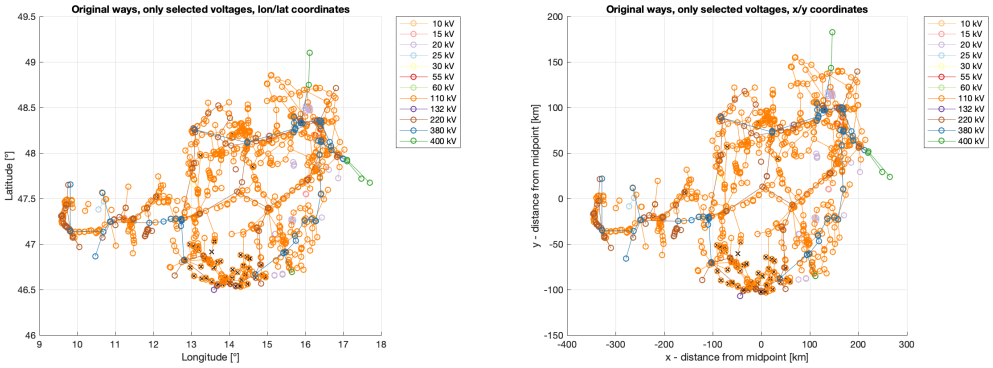
In den Abfragen von Zeile 59 bis 70 wird in der Konsole die Information ausgegeben, ob sich dieser Mittelpunkt auf Nord- bzw. Süd- und West- bzw. Ost-Hemisphäre befindet.

In den Formeln in Zeilen 78 bis 87 wird in Abhängigkeit des geographischen Breitengrads, eben *mean_lat*, mittels einer Approximation berechnet, wie viele km ein Längengrad ° (*km_per_lon_deg*) bzw. Breitengrad ° (*km_per_lon_deg*) an dieser Position entspricht.

In den Zeilen 90 bis 91 wird der Unterschied in Längen- bzw. Breitengrad eines jeden einzelnen Endpunktes zu *mean_lat* und *mean_lon* berechnet und in den Variablen *delta_lat1*, *delta_lat2*, *delta_lon1* und *delta_lon2* gespeichert.

Dieser Gradunterschied wird in den Zeilen 96 bis 99 nach [6] in Kilometerdistanzen umgerechnet, und danach in den Zeilen 102 bis 111 in *Cell-Arrays* umgewandelt um dem Datensatz hinzugefügt werden zu können.

Diese Approximation von ° in km ist an den Koordinaten von *mean_lat* und *mean_lon* sehr genau, je weiter man sich von diesen entfernt, vor allem in nördliche bzw. südliche Richtung, naturgemäß ungenauer.



(a) Längen-/ Breitengradkoordinaten (b) *x/y*-Koordinaten

Abbildung 4.3: Vergleich Längen-/Breitengrad- und *x/y*-Koordinaten

Da diese Daten im weiteren Verlauf des Programms aber lediglich die Berechnung der Leitungslänge sowie die Gruppierung von Endnodes beeinflussen, und letzteres sowieso maßgeblich von der einzustellenden Variable *neighbourhood_threshold* abhängig ist, ist eine eventuell auftretende Ungenauigkeit bei sehr großen Distanzen zum Mittelpunkt vernachlässigbar.

Der Unterschied zwischen der Darstellungen eines Datensatzes in Längen- und Breitengrad verglichen zu einer Darstellung in *x/y*-Koordinaten ist in Abb. 4.3 dargestellt.

In den Zeilen 115 und 116 wird die Länge einer Leitung mit dem Satz des Pythagoras berechnet und einem jedem Way-Element hinzugefügt.

Da diese Konvertierung von geographischen Längen- bzw. Breitengrad in späteren Funktionen nochmal gebraucht wird, werden die dazu notwendigen Informationen in den Variablen *degrees_to_km_conversion* zurückgegeben.

5 Funktionen - Modul Spannungsebenenauswahl

Das zweite von sechs Programmmodulen ist das Modul *Spannungsebenenauswahl*.

In diesem wird dem Benutzer die Möglichkeit geboten, aus sämtlichen im Datensatz verfügbaren Spannungsebenen die Spannungsebenen von Interesse auszuwählen und alle weiteren Berechnungen und den finalen Export nur mit diesen durchzuführen.

Der Aufbau des *Spannungsebenenauswahl-Moduls* ist schematisch in Abb. 5.1 dargestellt.

Das Modul *Spannungsebenenauswahl* besteht aus drei Funktionen:

1. *my_count_voltage_levels()*, beschrieben in Abschnitt 5.1,
2. *my_ask_voltage_levels()*, beschrieben in Abschnitt 5.2 und
3. *my_select_ways()*, beschrieben in Abschnitt 5.3.

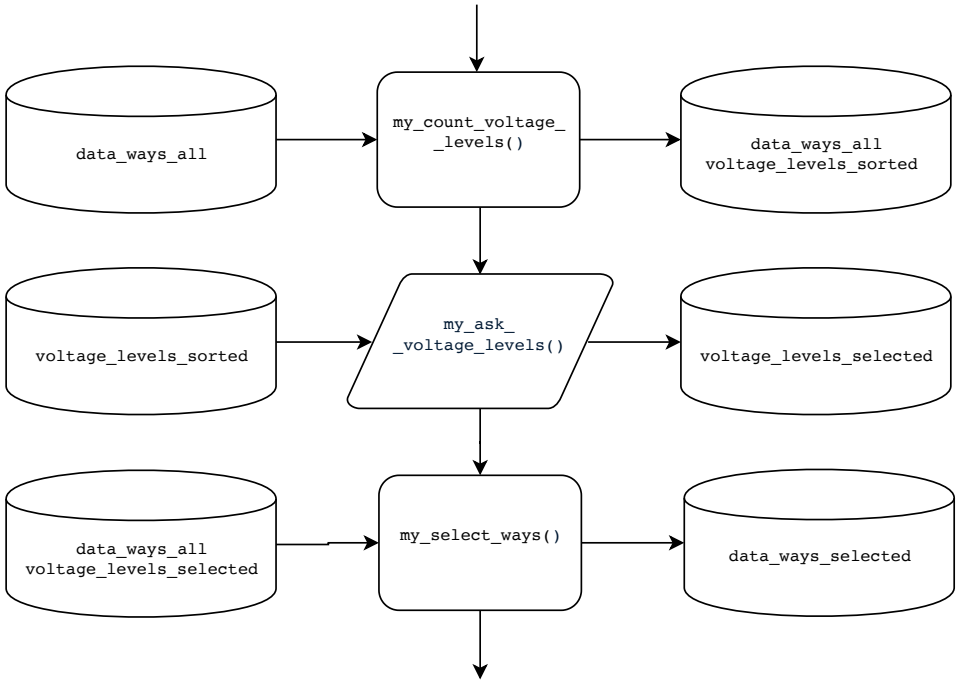


Abbildung 5.1: Schematischer Aufbau Modul 2 von 6

5.1 my_count_voltage_levels()

Die erste Funktion, *my_count_voltage_levels()*, extrahiert von jedem Way-Element aus *data_ways_all* den Wert des *fields* »tags / voltage« und fügt es als eigenständiges *field* dem Element hinzu.

```
1  function [data, voltage_levels_unique] ...
2      = my_count_voltage_levels(data)
3
4      % DESCRIPTION
5      % This function reads the tag information about the voltage level and
6      % adds that information to every way element. If a way has two or three
7      % different voltage levels, the corresponding way will be
8      % doubled/tripled automatically. A list of all voltage levels will be
9      % displayed to the console.
10     %
11     %
```

```
12      % INPUT
13      % data ... dataset of all way elements
14      %
15      % OUTPUT
16      % data ... updated dataset off all way elements: ways with multiple
17      %          voltage levels got cloned and "number of voltage levels" and
18      %          the volage level got added to every way element
19      % voltage_levels_unique ... a list of all voltage levels in the dataset
20
21      tic
22      disp('Start counting all voltage levels...')
23
24      %%% Extract all voltage levels and add them to every way element
25      for i_ways = 1 : numel(data)
26
27          % Check if there is even a voltage field
28          if not(isfield(data(i_ways).tags, 'voltage'))
29              % print warning to console
30              fprintf(['   ATTENTION! Way element UID %d does not ' ...
31                       'contain a field "voltage".' ...
32                       'This way wont be selected. \n'], data(i_ways).UID)
33
34              % cancel that element, skip rest of for-loop, go to next i_ways
35              continue
36          end
37
38          % clear voltage level variable (error if old values are still there)
39          voltage_levels = [];
40
41          % save the voltage in temporary variable
42          voltage_levels = str2double(data(i_ways).tags.voltage);
43
44          % If there is more than one voltage level, check if there are two
45          if isnan(voltage_levels)
46
47              % Split the two voltage levels, sperated by ";", up
48              voltage_levels ...
49                  = str2double(strsplit(data(i_ways).tags.voltage, ';'));
50          end
51
52          % if there is still a invalid voltage level, print a message to
53          % console and skip that element
54          if any(isnan(voltage_levels))
55
56              % print warning to console
57              fprintf(['   ATTENTION! UNKNOWN voltage level ("%s") ' ...
58                       'in UID %d. This way wont be selected. \n'], ...
59                        data(i_ways).tags.voltage, data(i_ways).UID)
60
61              % cancel that element, skip rest of for-loop, go to next i_ways
62              continue
63          end
```

```
64          % if it's just one voltage level, add voltage level to curr. way
65          if numel(voltage_levels) == 1
66
67              data(i_ways).voltage = voltage_levels;
68              data(i_ways).vlevels = 1;
69
70          % if there are two voltage levels, add flag and print to console
71          elseif numel(voltage_levels) == 2
72
73              data(i_ways).voltage = [];
74              data(i_ways).vlevels = 2;
75              fprintf(['   ATTENTION! Two voltage levels ("%s") ' ...
76                       'in UID %d. This way will be duplicated. \n'], ...
77                       data(i_ways).tags.voltage, data(i_ways).UID)
78
79          % if there are three voltage levels, add flag and print to console
80          elseif numel(voltage_levels) == 3
81
82              data(i_ways).voltage = [];
83              data(i_ways).vlevels = 3;
84              fprintf(['   ATTENTION! Three voltage levels ("%s") ' ...
85                       'in UID %d. This way will be tripled. \n'], ...
86                       data(i_ways).tags.voltage, data(i_ways).UID)
87
88          % if there is not voltage level entry at all, print message to console
89          else
90              data(i_ways).voltage = [];
91              data(i_ways).vlevels = [];
92              fprintf(['   ATTENTION! Unkown voltage levels ("%s") ' ...
93                       'in UID %d. This way wont be selected. \n'], ...
94                       data(i_ways).tags.voltage, data(i_ways).UID)
95          end
96      end
97
98
99      %%% Clone ways with two or three different voltage levels
100     % Initialize counter variables
101     num_of_cloned_ways = 0;
102     iterations_to_skip = 0;
103
104     % Go throuh every way element (add cloned ways to reach last way too)
105     for i_ways = 1 : numel(data) + num_of_cloned_ways
106
107         % Skip iterations if a way got cloned in a previous iteration
108         if iterations_to_skip > 0
109
110             % Skip once and decrease to_skip_counter by 1
111             iterations_to_skip = iterations_to_skip - 1;
112             continue;
113         end
114
115
```

```
116         % If there are two voltage leveles, duplicate a way
117         if data(i_ways).vlevels == 2
118
119             % Get the two voltage levels of current way
120             voltage_levels ...
121                 = str2double(strsplit(data(i_ways).tags.voltage, ';'));
122
123             % copy the current way two times
124             way_to_clone_a = data(i_ways);
125             way_to_clone_b = data(i_ways);
126
127             % Add the two voltage levels to one way each
128             way_to_clone_a.voltage = voltage_levels(1);
129             way_to_clone_b.voltage = voltage_levels(2);
130
131             % Duplicate the way
132             data = [data(1 : (i_ways - 1)); ...
133                     way_to_clone_a; way_to_clone_b;
134                     data((i_ways + 1) : end)];
135
136             % run the for-loop one iteration longer to reach the last way
137             num_of_cloned_ways = num_of_cloned_ways + 1;
138
139             % Skip next interation to ignore the cloned way
140             iterations_to_skip = 1;
141         end
142
143         % if there are three voltage levels,
144         if data(i_ways).vlevels == 3
145
146             % Get the three voltage levels of current way
147             voltage_levels ...
148                 = str2double(strsplit(data(i_ways).tags.voltage, ';'));
149
150             % copy the current way three times
151             way_to_clone_a = data(i_ways);
152             way_to_clone_b = data(i_ways);
153             way_to_clone_c = data(i_ways);
154
155             % Add the three voltage levels to one way each
156             way_to_clone_a.voltage = voltage_levels(1);
157             way_to_clone_b.voltage = voltage_levels(2);
158             way_to_clone_c.voltage = voltage_levels(3);
159
160             % Triple the way
161             data = [data(1 : (i_ways - 1)); ...
162                     way_to_clone_a; way_to_clone_b; way_to_clone_c;
163                     data((i_ways + 1) : end)];
164
165             % run the for-loop two iterations longer to reach the last way
166             num_of_cloned_ways = num_of_cloned_ways + 2;
167
```

```
168             % Skip the next two interation to ignore the cloned ways
169             iterations_to_skip = 2;
170         end
171     end
```

Matlab Code 5.1: Funktion *my_count_voltage_levels*()

Eine Aufzählung der im Datensatz vorhanden Spannungsebenen wird in der Variable *voltage_levels_selected* zurückgegebenen und zusätzlich in der Konsole angezeigt, siehe Abb. 5.2.

Darin erkennt man, dass im diesem beispielhaften Datensatz 195 mal eine 380 kV, 1.814 mal eine 110 kV und nur eine Leitung mit nicht identifizierbarer Spannungsebene vorhanden sind.

Command Window

voltage_level	number_of_ways
400000	8
380000	195
220000	311
132000	4
110000	1814
60000	3
55000	4
30000	5
25000	6
20000	43
15000	11
10000	3
0	2

```
... there are 1 way(s) with unknown voltage level.
... finished! (1.627 seconds)
```

Abbildung 5.2: Ausgabe von Funktion *my_count_voltage_levels()*

Sollte das *field* »tags / voltage« nicht existieren oder ungültige Werte besitzen, so wird in der Konsole ein Hinweis ausgegeben und das zugehörige Element übersprungen.

Sollten auf einer Leitung zwei oder drei unterschiedliche Spannungsebenen vorhanden sein, so wird diese Leitung verdoppelt bzw. verdreifacht und die zwei bzw. drei Spannungsebenen jeweils den Kopien zugeordnet.

In der ausgegebenen *Excel*-Datei müssen im Nachhinein dennoch diese Duplikate händisch überprüft werden um weitere Information, wie z.B. die Anzahl der Leitungen, anzupassen.

In Abb. 5.3 ist die Ausgabe eines beispielhaften Datensatzes zu sehen. Die Leitung mit der *UID* 1.589 trägt Leiterseile der Spannungsebenen 380 kV und als auch 110 kV.

Während bei der Leitung mit der *UID* 1.672 offensichtlich ein fehlerhafter und unvollständiger Datensatz vorhanden ist, müsste bei der Leitung mit der *UID* 1.288 manuell überprüft werden, warum als dritte Spannungsebene 0 kV angeben ist.

Möglicherweise handelt es sich hierbei um ein nicht angeschlossenes System, welches in Zukunft genutzt werden könnte, ein Leiterseil welches rein aus statischen Stabilitätsgründen verbaut wurde oder einfach nur um ein Blitzschutzseil.

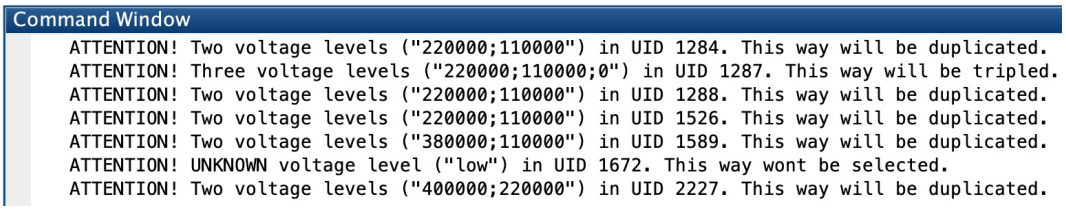


Abbildung 5.3: Ausgabe von Funktion *my_count_voltage_levels()*

5.2 my_ask_voltage_levels()

Die Funktion *my_ask_voltage_levels()* lässt den Benutzer die gewünschten Spannungsebenen auswählen.

```
function voltage_levels_selected ...
    = my_ask_voltage_levels(voltage_levels_sorted)

    % DESCRIPTION
    % This function opens an UI which displays all found voltage
    % levels of the dataset. The user can select one / multiple / all
    % voltage levels, this information will be returned as list. If the user
    % cancels the dialog, all voltage levels will be selected.
    %
    % INPUT
    % voltage_levels_sorted ... a list of all unique voltage levels of dataset
    %
    % OUTPUT
    % voltage_levels_selected ... a list of all selected voltage levels

    % Settings for the dialog
    vlevels_default = num2str(voltage_levels_sorted(:, 1));
    dialog_title = 'Voltage Level Selection';
    dialog_description = 'Plese select one or multiple voltage levels';
    dialog_window_size = [250, 300]; % matlab default: [160 300]

    % Create dialog to select voltagelevels
    [index_selected, b_select_ok] = listdlg('ListString', vlevels_default, ...
                                            'Name', dialog_title, ...
                                            'ListSize', dialog_window_size, ...
                                            'PromptString',dialog_description);

    % return selected voltage levels, if user made a selection
    if b_select_ok
        voltage_levels_selected = voltage_levels_sorted(index_selected, 1);
    else % If user didn't select anything, return all voltages
        voltage_levels_selected = voltage_levels_sorted(:, 1);
    end
end
```

Matlab Code 5.2: Funktion *my_ask_voltage_levels*()

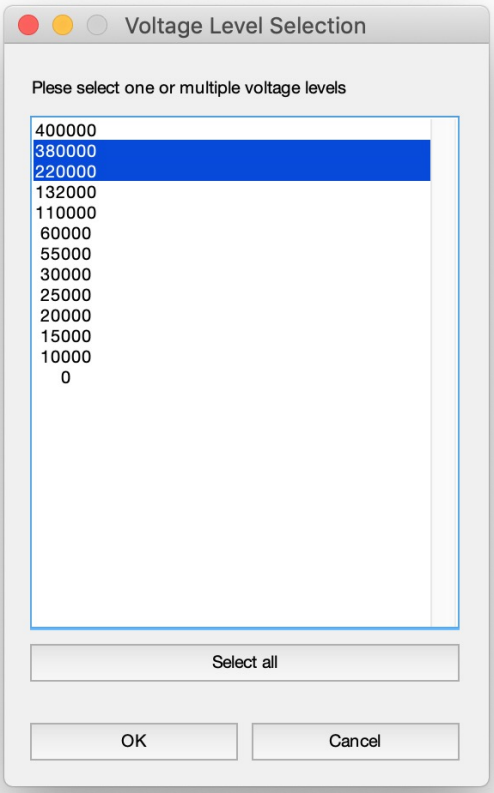


Abbildung 5.4: Spannungsebenenauswahlfenster von *my_ask_voltage_levels()*

Dazu wird dem Benutzer ein Auswahlfenster, siehe Abb. 5.4, angezeigt, mit welchem er eine, mehrere oder alle Spannungsebenen aus der Variable *voltage_levels_sorted()* auswählen kann. In diesem Beispiel wurden die Spannungsebenen 380 kV und 220 kV ausgewählt.

Die ausgewählten Spannungsebenen werden in der Variable *voltage_levels_selected()* zurückgegeben.

5.3 my_select_ways()

In der dritten Funktion *my_select_ways()* werden mithilfe der in *my_ask_voltage_levels()* ausgewählten Spannungsebenen aus den Originaldaten in *data_ways_all* lediglich die in *voltage_levels_selected* gespeicherten Spannungsebenen ausgewählt und in das neue *struct array data_ways_selected* kopiert.

```
function data_ways_selected ...
    = my_select_ways(data_ways_all, vlevels_selected)

    % DESCRIPTION
    % Copies all ways, which have a selected voltage level to a new struct
    %
    % INPUT
    % data_ways_all ... dataset of all ways
    % vlevels_selected ... list of selected voltage levels
    %
    % OUTPUT
    % data_ways_selected ... dataset of all ways with selected vlevel
    tic
    disp('Start selecting ways according to their voltage level...')

    % Initialize ID Counter for the new 'data_ways_selected'
    i_ways_selected = 1;

    % Go through every way element of all ways
    for i_ways_org = 1 : numel(data_ways_all)

        % If voltage level of current way got selected
        if any(ismember(data_ways_all(i_ways_org).voltage, vlevels_selected))

            % copy current way to new struct
            data_ways_selected(i_ways_selected) = data_ways_all(i_ways_org);

            % Increase element counter of new struct
            i_ways_selected = i_ways_selected + 1;
        end
    end

    % Transpose the new struct to match other dimensions
    data_ways_selected = data_ways_selected';

    fprintf('   ... finished! (%5.3f seconds) \n \n' , toc)
end
```

Matlab Code 5.3: Funktion *my_select_ways*()

6 Funktionen - Modul Datenanalyse

Das dritte von sechs Programmmodulen ist das *Datenanalyse-Modul*. In diesem wird der Datensatz nach verschiedenen Kriterien untersucht mit den Zielen, den Datenumfang zu verringern, Gleichspannungsleitungen zu erkennen und notwendige Leitungen zu vervielfachen.

Der Aufbau des *Datenanalyse-oduls* ist schematisch in Abb. 6.1 dargestellt.

Das Modul *Datenanalyse* besteht aus drei Funktionen:

1. *my_delete_busbars()*, beschrieben in Abschnitt 6.1,
2. *my_count_possible_dc()*, beschrieben in Abschnitt 6.2 und
3. *my_count_cables()*, beschrieben in Abschnitt 6.3.

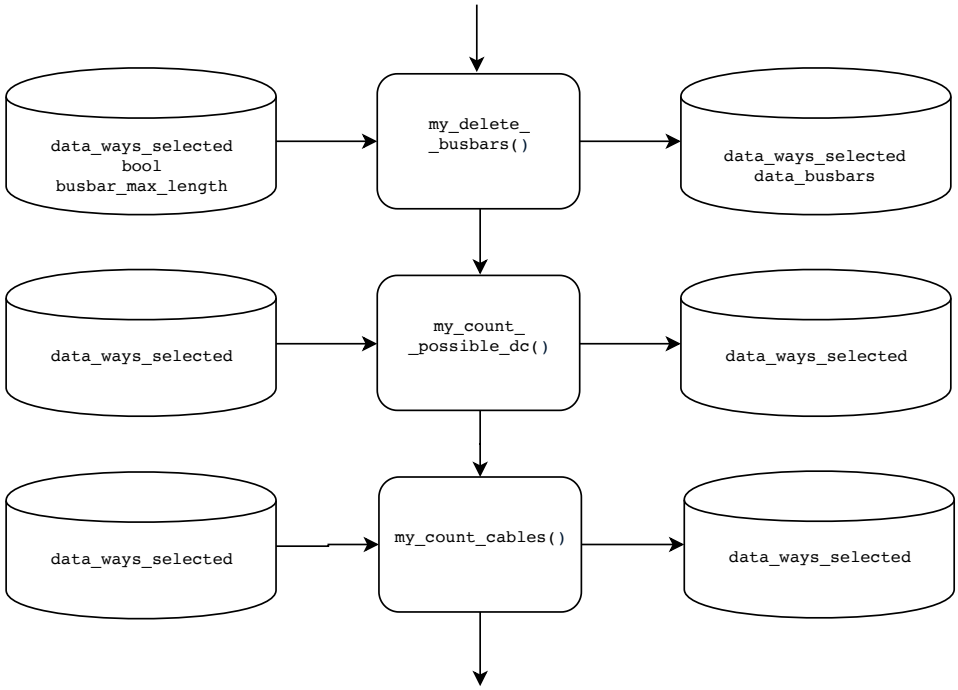


Abbildung 6.1: Schematischer Aufbau Modul 3 von 6

6.1 my_delete_busbars()

Die erste Funktion, *my_delete_busbars()*, reduziert den Datensatz durch Entfernung von sehr kurzen Leitungsstücken, welche innerhalb von Umspannwerken vorkommen.

```
1  function [data, data_busbars] ...
2      = my_delete_busbars(data, bool, busbar_max_length)
3
4      % DESCRIPTION
5      % This function checks if a way is declared as a busbar, if so, it
6      % checks if its length is less than the max treshhold and adds a flag
7      % to that way element. The lenght of a busbar will be saved in a
8      % seperate variable, which can optionally be plotted in a histogram to
9      % set the max. busbar lenght accordingly. All busbars will be extraced to
10     % a seperate variable and then deleted from the original dataset.
11     %
```

```
12      % INPUT
13      % data ... dataset of selected ways
14      % bool ... boolean struct if the histogram should be plotted or not
15      % busbar_max_length ... the maximal length a busbar can have
16      %
17      %
18      % OUTPUT
19      % data ... updated dataset with all busbars deleted
20      % data_busbars ... all way elements which are busbars
21
22      tic
23      disp('Start deleting ways with type "busbar"...')
24
25      % Initialize counter for busbars
26      i_busbars = 0;
27
28      % go through all way-elements
29      for i_ways = 1 : numel(data)
30
31          % Condition if the tag field "line" exists
32          b_line_exists = isfield(data(i_ways).tags, 'line');
33
34          % Condition if length of current way is less then max busbar length
35          b_length_ok = data(i_ways).length < busbar_max_length;
36
37          % if "line" field exists and if it's value is "busbar"
38          if b_line_exists && strcmp(data(i_ways).tags.line, 'busbar')
39
40              % and if its length isn't too long
41              if b_length_ok
42
43                  % Set flag that current way is a busbar
44                  data(i_ways).busbar = true;
45
46                  % Increase counter of found busbars
47                  i_busbars = i_busbars + 1;
48
49                  % Save its length for optionally histogram
50                  lengths_of_busbars(i_busbars) = data(i_ways).length;
51
52              % but if its length is too long, skip it and print message
53              else
54                  fprintf(['   ATTENTION!  Way Element UID %d has type ' ...
55                  '"busbar", but is too long. \n               ' ...
56                  'Lenght: %5.2f km of max. %3.1f km \n              ' ...
57                  'This way wont be added to the ' ...
58                  '"busbar" exception list. \n'], ...
59                  data(i_ways).UID, data(i_ways).length, busbar_max_length)
60                  data(i_ways).busbar = false;
61              end
62
63
```

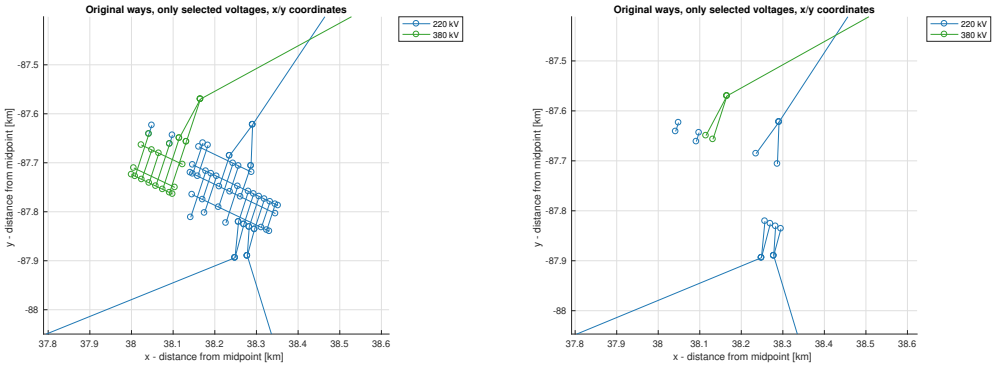
```
64          % If it's not a busbar...
65          else
66              % ... set flag accordingly
67              data(i_ways).busbar = false;
68          end
69      end
70
71      % extract all busbars to a seperate variable
72      data_busbars = data([data.busbar]);
73
74      % delete all busbars from original dataset
75      data([data.busbar]) = [];
76
77      % Optional: Histogram of busbar lengths, to set max busbar length
78      if bool.histogram_length_busbars
79          figure
80          histogram(lengths_of_busbars, 200)
81          title('Lengths of busbars below busbar-max-length-treshold')
82          xlabel('Length [km]'), ylabel('Number of busbars with that length')
83      end
84
85      fprintf(['   ... %d busbars have been deleted\n' ...
86               '   ... finished! (%5.3f seconds) \n \n'], ...
87               i_busbars, toc)
88  end
```

Matlab Code 6.1: Funktion *my_delete_busbars*()

In einigen Way-Elementen befindet sich das *field* »tags / line«, und einige davon haben den Wert »busbar« oder »bay« und bezeichnen Bauteile in einem Umspannwerk.

Da die Längen dieser Leitungen im Normalfall sehr kurz sind und ein Umspannwerk im Laufe des MATLAB Programms sowieso in ein einzelnes Node-Element umgewandelt wird, können diese Elemente bereits an früher Stelle im Programm entfernt werden und damit die Rechenzeit verkürzen.

Da es jedoch manchmal vorkommt, dass teils kilometerlange Leitungen fälschlicherweise als »busbar« deklariert werden, wird kontrolliert, ob sich die Leitungslänge einer vermeintlichen Busbar unter einem bestimmten Maximalwert befindet.



(a) Datensatz mit Busbars und Bays (b) Datensatz ohne Busbars und Bays

Abbildung 6.2: Auswirkung der Funktion *my_delete_busbars()*

Dieser Maximalwert wird in der Variable *busbar_max_length*, beschrieben in Kapitel 3.2, eingestellt. Ein Wert von 1 km hat sich als praktikable Obergrenze erwiesen.

Die Funktion übernimmt zudem die Variable *bool*, mit welcher optionale Visualisierung der Länge erkannter Busbars erstellt werden können. Diese Visualisierungen sind hilfreich um den Wert der Variable *busbar_max_length* anzupassen.

In der Variable *data_busbars* werden sämtliche als Busbars- oder Bays- identifizierten Elemente abgespeichert um in einer späteren Funktion dennoch visualisiert werden zu können.

In Abb. 6.2 wird das Entfernen von Busbars- und Bays-Elementen in einem Umspannwerk anschaulich dargestellt.

6.2 my_count_possible_dc()

In der zweiten Funktion, *my_count_possible_dc()*, wird der Datensatz nach möglichen Gleichspannungsleitungen untersucht.

```
function [data, dc_candidates] ...
    = my_count_possible_dc(data)

    % DESCRIPTION
    % This function checks every way element, if it could potentially be a
    % DC line. There are three hints that a line may be a DC line: It has
    % only 1 cable, the frequency is "0" or name contains somewhere the two
    % letter "dc". If one or more of those checks are correct, the UID,
    % reason and voltage level will be copied to a seperate variable for
    % later manuell checks.
    %
    % INPUT
    % data ... the dataset of selected ways
    %
    % OUTPUT
    % data ... updated dataset, including a flag if a way may be a DC line
    % dc_candidates ... list of all UIDs which may be a DC line

    tic
    fprintf('Start detecting lines which could be DC lines... \n')

    % Initialize the DC_candidate struct
    dc_candidates(1).UID = [];

    % Go through every way element
    for i_ways = 1 : numel(data)

         % if field "frequency" exists AND its value is 0
        if isfield(data(i_ways).tags, 'frequency') ...
                && str2double(data(i_ways).tags.frequency) == 0

            % Set boolean tag 'b_candidate_dc' true
            data(i_ways).dc_candidate = true;

            % Add the UID of that way
            dc_candidates(end + 1).UID = data(i_ways).UID;

            % Add voltage level of that way
            dc_candidates(end).voltage_level = data(i_ways).voltage;

            % Add the reason
            dc_candidates(end).reason = 'tag "frequency" has value "0"';
```

```
43          else
44              % Set boolean tag false (next condition may change that)
45              data(i_ways).dc_candidate = false;
46          end
47
48
49          % if field "name" exists AND contains the case insensitive value 'DC'
50          if isfield(data(i_ways).tags, 'name') ...
51                 && any(strfind(lower(data(i_ways).tags.name), 'dc'))
52
53              % Set boolean tag 'b_candidate_dc' true
54              data(i_ways).dc_candidate = true;
55
56              % Add the UID of that way
57              dc_candidates(end + 1).UID = data(i_ways).UID;
58
59              % Add the reason
60              dc_candidates(end).reason = 'tag "name" contains "DC"';
61
62              % Add voltage level of that way
63              dc_candidates(end).voltage_level = data(i_ways).voltage;
64
65          else
66               % Set boolean tag false (next condition may change that)
67              data(i_ways).dc_candidate = false;
68
69          end
70
71          % if field "cables" exists AND its value is 1
72          if isfield(data(i_ways).tags, 'cables') ...
73                 && str2double(data(i_ways).tags.cables) == 1
74
75              % Set boolean tag 'b_candidate_dc' true
76              data(i_ways).dc_candidate = true;
77
78              % Add the UID of that way
79              dc_candidates(end + 1).UID = data(i_ways).UID;
80
81              % Add voltage level of that way
82              dc_candidates(end).voltage_level = data(i_ways).voltage;
83
84              % Add the reason
85              dc_candidates(end).reason = 'tag "cables" has value "1"';
86
87          else
88              % Set boolean tag false
89              data(i_ways).dc_candidate = false;
90          end
91      end
92
93      % Delete the first (and empty) entry of that cable struct
94      dc_candidates(1) = [];
```

```
95      % Output information depending if candidates were found or not
96      if size(dc_candidates, 2) == 0
97
98          % Add information to variable
99          dc_candidates(1).UID = ['No possible DC candidate in all ways of ' ...
100                                  'those selected voltage levels found!'];
101
102         % Print information to console
103         disp('   ... no potentially DC lines found.')
104
105     else
106         % Print how many ways do not contain information about cables
107         fprintf(['   ... %d ways could pontentially be a DC line.  \n' ...
108                  '   ... Please refer in workspace to variable DC ' ...
109                      'candidates \n ' ...
110                  'to manually check them if necessary! \n'], ...
111                  numel(unique([dc_candidates(:).UID])))
112     end
113
114     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
115 end
```

Matlab Code 6.2: Funktion *my_count_possible_dc*()

Da der Datensatz in *OpenStreetMap* keine einheitliche Möglichkeit bietet, Gleichspannungsleitungen als solche zu kennzeichnen, sucht dieses Programm nach drei möglichen Kriterien für eine Gleichspannungsleitung:

1. Das *field* »tags / frequency« existiert und hat den Wert »0«.
2. Das *field* »tags / cables« existiert und hat den Wert »1«.
3. Das *field* »tags / name« existiert und enthält die Buchstabenkombination »DC«.

Trifft mindestens eines dieser Kriterien zu, wird das Way-Element in der Variable *dc_candidates* gespeichert und eine Information dazu in der Konsole ausgegeben.

Eine manuelle Kontrolle dieser möglichen Gleichspannungsleitungen ist danach notwendig, um mögliche Schema zu erkennen und so im Datensatz Gleichspannungsleitungen identifizieren zu können.

Mögliche Merkmale können sein, dass Beispielsweise alle Way-Elemente mit einem bestimmten Spannungslevel Gleichspannung führen oder dass Leitungen, welche im selben Umspannwerk enden, Gleichspannungsleitungen sind.

In Abb. 6.3 ist der Inhalt der Variable *dc_candidates* eines Datensatzes dargestellt. In diesem ist zu sehen, dass 17 mal einer der drei Hinweise zugetroffen hat.

In der Spalte *UID* ist die Identifikationsnummer des jeweiligen Way-Elementes beschrieben, in der Spalte *reason* der Grund für die Klassifizierung als mögliche Gleichspannungsleitung und in der Spalte *voltage_level* die Spannungsebene des jeweiligen Elements.

Variables – dc_candidates

dc_candidates

1x17 struct with 3 fields

Fields	UID	reason	voltage_level
1	617	'tag "name" contains "DC"'	0
2	725	'tag "frequency" has value "0"'	533000
3	725	'tag "name" contains "DC"'	533000
4	725	'tag "cables" has value "1"'	533000
5	4466	'tag "cables" has value "1"'	132000

Abbildung 6.3: Inhalt der Variable *dc_candidates*

6.3 my_count_cables()

In der Funktion *my_count_cables()* wird der Datensatz dahingehend untersucht, wie viele *cables*, also Leiterseile, ein Way-Element besitzt.

```
function data ...
    = my_count_cables(data)

    % DESCRIPTION
    % This function checks for every way element the number of cables, adds
    % them to the dataset and to a sepearte variable "cabels_per_way". If a
    % line obviously carries 2, 3 or 4 systems, a flag will be set
    % accordingly and that way will be doubled, tripled or quadrupled in a
    % later function.
    %
    % INPUT
    % data ... dataset of selected ways
    %
    % OUTPUT
    % data ... updated dataset with new fields "num_of_cables" and "systems"


    tic
    fprintf('Start counting cables per way... \n')

    % Initialize the cables struct
    cables_per_way(1).UID = [];

    % Go through every way
    for i_ways = 1 : numel(data)

        % check if "cable" field even exists
        if isfield(data(i_ways).tags, 'cables')

            % Some ways have different number of cables for different voltage
            % levels, catch that NaN error, if a cable is for example "3;6".
            if isnan(str2double(data(i_ways).tags.cables))
                fprintf(['   ATTENTION! Unknown cable number ("%s") in ' ...
                         'UID %d. This way wont be cloned ' ...
                         'automatically.\n'], ...
                         data(i_ways).tags.cables, data(i_ways).UID);
                continue;
            end

            % Add the UID of that way
            cables_per_way(end + 1).UID = data(i_ways).UID;

```

```
43              % Add the number of cables as field to dataset
44              data(i_ways).cables = str2double(data(i_ways).tags.cables);
45
46              % Add the number of cables of that way to a seperate variable
47              cables_per_way(end).num_of_cables ...
48                  = str2double(data(i_ways).tags.cables);
49
50              % If it's a double system (2x3 cables), set flag accordingly
51              if cables_per_way(end).num_of_cables == 6
52                  data(i_ways).systems = 2;
53
54              %If it's a triple system (3x3 cables), set flag accordingly
55              elseif cables_per_way(end).num_of_cables == 9
56                  data(i_ways).systems = 3;
57
58              % if it's a quadruple system (4x3 cables), set flag accordingly
59              elseif cables_per_way(end).num_of_cables == 12
60                  data(i_ways).systems = 4;
61
62              % if none of the above, leave it empty
63              else
64                  data(i_ways).systems = [];
65              end
66
67          % there is no information regarding cable number
68          else
69              % leave flag empty
70              data(i_ways).systems = [];
71          end
72      end
73
74      % Delete the first (and empty) entry of that cable struct
75      cables_per_way(1) = [];
76
77      if size(cables_per_way, 2) == 0
78          % Print that in this selection no cables per way info was found
79          fprintf(['   ... the ways in this voltage level selection \n     ' ...
80                   '   dont provide information about number of cables...\n'])
81
82          % Save that information in output variable too
83          cables_per_way(1).UID = ['   ... the ways in this voltage level ' ...
84                                   'selection dont provide any information ' ...
85                                   'about number of cables! \n'];
86
87      else
88          % Calculate how many ways have a certain cable count
89          [cables_occurance, cables_unique] ...
90              = hist([cables_per_way.num_of_cables], ...
91                     unique([cables_per_way.num_of_cables]));
92
93          % print that information to console
94          fprintf('\n')
```

```
95          disp(array2table([cables_unique', cables_occurance'], ...
96                           'VariableNames', {'cables_per_way', 'number_of_ways'}))
97
98          % Print how many ways do not contain information about cables
99          fprintf('   ... %d ways with unknown number of cables. \n', ...
100                 numel(data) - sum(cables_occurance))
101
102         % Print little explanation to console
103         fprintf(['   ... ways with 6 cables will be doubled, ways with 9 ' ...
104                  'cables tripeled \n       and ways with 12 cables ' ...
105                  'quardupled.\n   ... Please refer in workspace to '...
106                  'variable "cables_per_way"   \n       to manually check ' ...
107                  'other ways (DC? Traction Current? Unused cable?) ' ...
108                  'if necessary. \n'])
109     end
110
111     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
112 end
```

Matlab Code 6.3: Funktion *my_count_cables*()

Die Funktion liest von jedem Way-Element, sofern vorhanden, das *field* »tags / cables« aus, speichert es in die lokale Variable *cables_per_way* und fügt diese Information als neues *field* »cables« einem jedem Way-Element des Datensatzes hinzu.

Es muss klargestellt werden, dass sich der Wert des Feldes *cable* nicht auf mögliche Bündelleiterkonfigurationen bezieht. Diese Information ist in einigen Datensätzen mit dem *field* »tags / wires« und möglichen Werten wie »single«, »double«, »triple« oder »quadruple« abgedeckt.

Die Leiterseilanzahl und die vorkommende Häufigkeit wird in der Konsole ausgegeben, siehe dazu Abb. 6.4. In diesem Beispieldatensatz ist zu sehen, dass 620 Leitungen verdoppelt werden, da sie sechs Leiterseile und damit zwei Systeme aufweisen. Die 120 Leitungen mit je vier Leiterseilen sind wahrscheinlich Bahnstromleitungen.

Eine nähere Untersuchung an der einen Leitung mit 10 Leiterseilen ergab in diesem Beispiel, dass es sich um eine drei-systemige Drehstromleitung mit einem zusätzlichem Blitzschutzleiterseil handelt.

```
Command Window
  Start counting cables per way...

      cables_per_way    number_of_ways
      ______________    ______________

       2                104
       3                880
       4                120
       6                620
       8                  9
       9                 24
      10                  1
      12                 69

     ... 62 ways with unknown number of cables.
     ... ways with 6 cables will be doubled, ways with 9 cables tripeled
         and ways with 12 cables quardupled.
     ... Please refer in workspace to variable "cables_per_way"
         to manually check other ways (DC? Traction Current? Unused cable?) if necessary.
     ... finished! (0.402 seconds)
```

Abbildung 6.4: Ausgabe von Funktion *my_count_cables()*

Die Information über die Anzahl der Leiterseile ist sehr hilfreich um das mögliche Vorkommen von Bahnstromleitungen (2, 4, oder 8 Leiterseile) oder Gleichspannungsleitungen (1 oder 2 Leiterseile) zu erkennen, sowie bei bei einigen konkreten Fällen nähere Untersuchungen zu starten (z.B. unbeschaltene Leiterseile aus Stabilitätsgründen, kombinierte Drehstrom- und Bahnstromleitungen, Verwendung von Blitzschutzleiterseilen, ...).

In dieser Funktion ist konkret von Interesse, ob es sich bei einem Way-Element ggf. um eine mehr-systemige Leitung handelt, sprich ob die Anzahl der Leiterseile Vielfache von 3, wie 6, 9, oder 12 entspricht. In diesen Fällen wird davon ausgegangen, dass es sich um zwei-, drei- oder vier-systemige Leitungen handelt.

Sofern dies der Fall ist, wird das jeweilige Way-Element mit einem entsprechenden *field* versehen und die jeweiligen Way-Elemente in der Funktion *my_add_LtgsID_clone_ways()*, beschrieben in Abschnitt 8.3, vervielfacht. Dieser Vorgang ist zudem anschaulich in Abb. 8.2 dargestellt.

7 Funktionen - Modul Gruppierung

Das vierte von sechs Programmmodulen ist das *Gruppierung-Modul*. In diesem werden Endpunkte nach bestimmten Kriterien zusammengefasst.

Der Aufbau des *Gruppierung-Moduls* ist schematisch in Abb. 7.1 dargestellt.

Das Modul *Gruppierung* besteht aus sieben Funktionen:

1. *my_calc_distances_between_endpoints()*, beschrieben in Abschnitt 7.1,
2. *my_calc_stacked_endnodes()*, beschrieben in Abschnitt 7.2,
3. *my_calc_neighbouring_endnodes()*, beschrieben in Abschnitt 7.3,
4. *my_group_nodes()*, beschrieben in Abschnitt 7.4,
5. *my_group_stacked_endnodes()*, beschrieben in Abschnitt 7.5,
6. *my_group_neighbouring_endnodes()*, beschrieben in Abschnitt 7.6 und
7. *my_add_final_coordinates()*, beschrieben in Abschnitt 7.7.

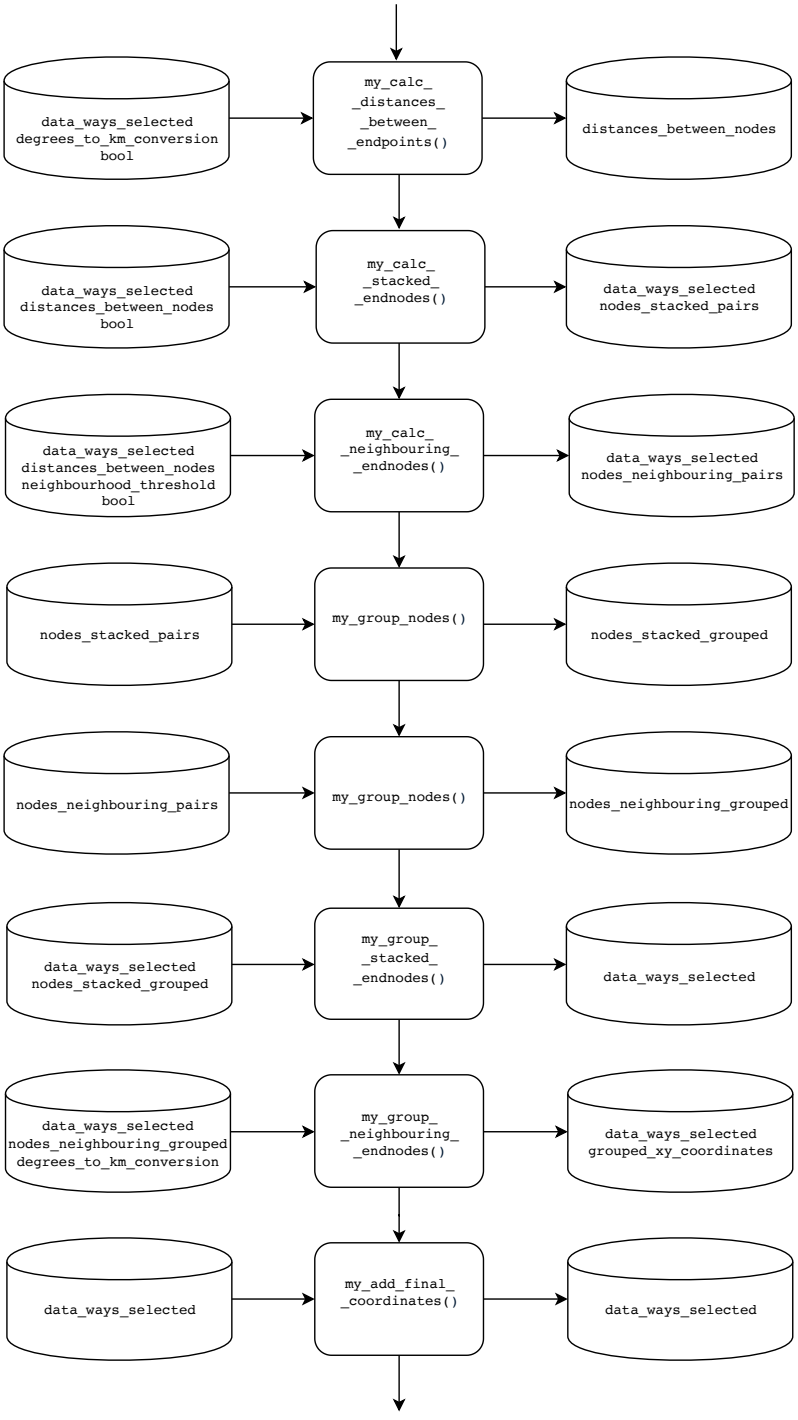


Abbildung 7.1: Schematischer Aufbau Modul 4 von 6

7.1 my_calc_distances_between_endpoints()

Die erste Funktion, *my_calc_distances_between_endpoints()*, berechnet die Distanz zwischen allen Endpunkten.

```
function M ...
    = my_calc_distances_between_endpoints(data, degrees_to_km_conversion, bool)

    % DESCRIPTION
    % This function creates a Matrix "M" or "distances" whith all distances
    % between all endpoints. "M" would be a diagonal symmetrical Matrix
    % (distances A to B is equal to distances B to A), so all elements in
    % the south-west diagonal half will be set to "NaN". Distances
    % between the same element (distance A to A or B to B) will be set to
    % "-1" since this is an impossible distance value and therefore
    % distinguishable. The correct value would be "0", but since we are
    % looking speficially for stacked endnodes (distance A to B equals 0)
    % the true value ("0") will not be used. Optionally a histogram of all
    % distances can be plotted - this can be very useful to set the
    % neighbouring threshold value.
    %
    % INPUT
    % data ... dataset of all selected ways
    % bool ... boolean selector to optionally plot a histogram.
    %
    % OUTPUT
    % M ... matrix with all distances between all endpoints


    tic
    disp(['Start calculating distances between all endpoints' ...
          '... (takes a few seconds)'])

    % preallocate the distance matrix with NaN-elements
    M = NaN(numel(data)*2);

    % Extract degrees to km conversion data into variables
    km_per_lon_deg = degrees_to_km_conversion(1);
    km_per_lat_deg = degrees_to_km_conversion(2);

    % Fetch coordinates for the horizontal row which contains
    % every 4x4 block of endnodes.
    all_lon1 = [data(1:end).lon1];
    all_lon2 = [data(1:end).lon2];
    all_lat1 = [data(1:end).lat1];
    all_lat2 = [data(1:end).lat2];

```

```
43      % go through each row of distance matrix
44      for i_row = 1 : numel(data)
45
46          % Initialize variables, delete old values before each iteration
47              % naming scheme of variables:
48              % lat_deltas... = all latitudinal deltas to the values of data_row
49              % lon_deltas... = all longitudinal deltas to the values of data_row
50              % ...to_xxxx... = refering to lon1/lat1/lon1/lon2 of data_column
51              % ...deg/km = value in degrees or in x/y km
52          lon_deltas_to_lon1_deg = [];
53          lon_deltas_to_lon2_deg = [];
54          lat_deltas_to_lat1_deg = [];
55          lat_deltas_to_lat2_deg = [];
56          lon_deltas_to_lon1_km = [];
57          lon_deltas_to_lon2_km = [];
58          lat_deltas_to_lat1_km = [];
59          lat_deltas_to_lat2_km = [];
60
61          % Create the 4x4 field of the current row, which will calculate
62          % distances to all other endnodes
63          data_column = [data(i_row).lon1, data(i_row).lat1; ...
64                         data(i_row).lon2, data(i_row).lat2];
65
66          % Every iteration this row gets smaller by one 4x4 block. Therefore
67          % delete the first coordinates from previous run
68          all_lon1 = all_lon1(2:end);
69          all_lon2 = all_lon2(2:end);
70          all_lat1 = all_lat1(2:end);
71          all_lat2 = all_lat2(2:end);
72
73          % Preallocate the row vector
74          data_row = zeros(2, numel(all_lon1) * 2);
75
76          % Copy all coordinates in alternating order to the row
77          data_row(1, 1:2:end) = all_lon1(1:end);
78          data_row(1, 2:2:end) = all_lon2(1:end);
79          data_row(2, 1:2:end) = all_lat1(1:end);
80          data_row(2, 2:2:end) = all_lat2(1:end);
81
82          % Calc absolute distance in degree between lon/lat coordinates
83          lon_deltas_to_lon1_deg = data_column(1) - data_row(1:2:end);
84          lon_deltas_to_lon2_deg = data_column(2) - data_row(1:2:end);
85          lat_deltas_to_lat1_deg = data_column(3) - data_row(2:2:end);
86          lat_deltas_to_lat2_deg = data_column(4) - data_row(2:2:end);
87
88          % Convert the delta_degree to delta_kilometer
89          lon_deltas_to_lon1_km = lon_deltas_to_lon1_deg * km_per_lon_deg;
90          lon_deltas_to_lon2_km = lon_deltas_to_lon2_deg * km_per_lon_deg;
91          lat_deltas_to_lat1_km = lat_deltas_to_lat1_deg * km_per_lat_deg;
92          lat_deltas_to_lat2_km = lat_deltas_to_lat2_deg * km_per_lat_deg;
93
94
```

```
95          % Use Pythagoras to calculate distances between endpoints
96          M_new_row = [];
97          M_new_row(1, :) ...
98              = sqrt(lon_deltas_to_lon1_km.^2 + lat_deltas_to_lat1_km.^2);
99          M_new_row(2, :) ...
100             = sqrt(lon_deltas_to_lon2_km.^2 + lat_deltas_to_lat2_km.^2);
101
102         % Apply the newly calculated distance row to the distance matrix
103         M([i_row*2 - 1, i_row*2], i_row*2 - 1 : end) = [-ones(2), M_new_row];
104     end
105
106     % Plot a Histogram of all the distances
107     if bool.histogram_distances_between_endpoints
108         disp('   ... start visualizing all distances in a histogram ...')
109
110         h = figure;
111         % Set windows size double the standard length
112         set(gcf, 'Position', [h.Position(1:3), h.Position(4) * 2])
113
114         subplot(5,1,1)
115         histogram(M, 200, 'BinLimits', [0, max(max(M))])
116         title('Distances between all endnodes')
117         ylabel('number of pairs'), xlabel('distance [km]')
118
119         subplot(5,1,2)
120         histogram(M, 200, 'BinLimits', [0, 10])
121         ylabel('number of pairs'), xlabel('distance [km]')
122
123         subplot(5,1,3)
124         histogram(M, 400, 'BinLimits', [-1.5, 2])
125         ylabel('number of pairs'), xlabel('distance [km]')
126
127         subplot(5,1,4)
128         histogram(M, 300, 'BinLimits', [0, 0.3])
129         ylabel('number of pairs'), xlabel('distance [km]')
130
131         subplot(5,1,5)
132         histogram(M, 300, 'BinLimits', [0 + eps, 0.3])
133         ylabel('number of pairs'), xlabel('distance [km]')
134     end
135     fprintf('   ... finished! (%5.3f seconds) \n \n' , toc)
136 end
```

Matlab Code 7.1: Funktion *my_calc_distances_between_endpoints*()

Sämtliche Distanzen werden in einer Distanzmatrix gespeichert. Dass sämtliche Endknoten als Teile von Way-Elementen gespeichert sind, verkompliziert den Aufbau dieser Distanzmatrix erheblich.

Doch um die Funktionsweise von *my_calc_distances_between_endpoints()* überhaupt verstehen zu können, lohnt sich die Erklärung an Hand eines Miniaturbeispiels, nämlich anhand der Berechnung der Distanzen zwischen den vier Endknoten von zwei Way-Elementen, wie in Abb. 7.2 dargestellt.

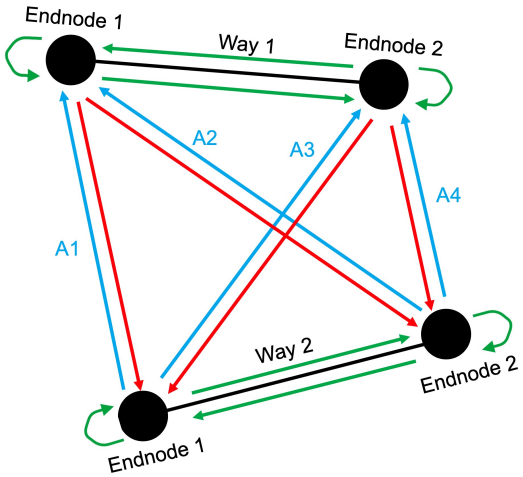


Abbildung 7.2: Schematischer Zusammenhang der Distanzmatrix

Die vier Distanzen zwischen den vier Endknoten *Endnode 1* und *Endnode 2* der beiden Way-Elementen *Way 1* und *Way 2* sind mit den blauen Pfeilen *A1*, *A2*, *A3* sowie *A4* dargestellt.

Intuitiv ist für uns Menschen die Distanz von *Objekt A* zu *Objekt B* natürlich die gleiche wie von *Objekt B* zu *Objekt A*. Ein Computer unterscheidet jedoch zwischen diesen beiden Distanzen, auch wenn sie identische Zahlwerte aufweisen. Aus diesem Grund sind die Distanzen *-A1*, *-A2*, *-A3* sowie *-A4* mit roten Pfeilen in gegenläufiger Richtung dargestellt.

Die Distanzen zwischen den beiden Endknoten eines Way-Elements lassen sich auch wieder als vier gegenläufige Pfeile darstellen, ebenso die Distanzen von einem Endknoten zu sich selbst. Diese vier Distanzen eines Endknotens zu sich selbst, sowie die vier Distanzen, welche letztendlich die Länge einer Leitung beschreiben, sind in grün dargestellt.

Von den nun dargestellten 16 Pfeilen sind jedoch nur 4 von tatsächlichem Interesse. Aus diesem Grund werden die gegenläufigen Distanzpfeile, welche in rot dargestellt wurden, in der Distanzmatrix in MATLAB als *NaN* belegt [7]. *NaN* (»not a number«) dient damit als Platzhalter für einen ungültigen Wert, welcher im weiteren Programmverlauf nicht beachtet wird.

Die acht grünen Pfeile enthalten nun zum einen unsinnige Informationen, wie die Distanz eines Endpunktes zu sich selbst, sowie zum anderen bereits bekannte Informationen, wie die Länge einer Leitung, welche schon in der Funktion *my_add_coordinates()* in Abschnitt 4.3 berechnet wurde.

Aus diesem Grund wird diesen Werten die physikalische unmögliche Distanz *-1* zugewiesen. Dies hat den Vorteil, dass sofort ersichtlich ist, dass hier bewusst ein unplausibler Wert gesetzt wurde, welche sich jedoch von den *NaN* der roten, redundanten Pfeile, unterscheiden soll.

Die eigentliche Distanzmatrix ist nun aus Hauptreihen und Hauptspalten der jeweiligen Way-Elementen aufgebaut. Diese Hauptreihen bzw. Hauptspalten unterteilen sich wiederum in zwei weitere Reihen bzw. Spalten, nämlich die der jeweiligen Endknoten *Endnode 1* und *Endnode 2*.

Die nun eigentlich gesuchte Distanz zwischen den vier Endpunkten zweier Way-Elemente besteht somit aus einer eigenen *2x2* Matrix. Ein jedes dieser vier Matrixfelder wird wiederum aus vier weiteren Variablen berechnet: Den Koordinaten des ersten und den Koordinaten des zweiten Endknoten.

In Abb. 7.3 ist die Distanzmatrix, welche in der Funktion *my_calc_distances_between_endpoints()* berechnet wird, schematisch dargestellt. Die Farben der jeweiligen Felder (blau, rot, grün) stimmen inhaltlich zu den Farben der soeben beschriebenen Pfeile überein.

Um nun die Distanz *A1* zu berechnen, werden die *lon/lat* Elemente des *Endnode 1* des *Way 1* sowie die *lon/lat* Elemente des *Endnode 1* des *Way 2* benötigt. Ebenso werden für die Distanz *Y4* die *lon/lat* Elemente des *Endnode*

				Way 1		Way 2		Way 3		Way 4		Way 5		Way 6	
				Endnode 1	Endnode 2	Endnode 1	Endnode 2	Endnode 1	Endnode 2	Endnode 1	Endnode 2	Endnode 1	Endnode 2	Endnode 1	En
				lon	lon	lon	lon	lon	lon	lon	lon	lon	lon	lon	
				lat	lat	lat	lat	lat	lat	lat	lat	lat	lat	lat	
Way 1	Endnode 1	lon	lat	-1	-1	A1	A2	B1	B2	C1	C2	D1	D2	E1	
	Endnode 2	lon	lat	-1	-1	A3	A4	B3	B4	C3	C4	D3	D4	E3	
Way 2	Endnode 1	lon	lat	NaN	NaN	-1	-1	Z1	Z2	Y1	Y2	X1	X2	W1	
	Endnode 2	lon	lat	NaN	NaN	-1	-1	Z3	Z4	Y3	Y4	X3	X4	W3	
Way 3	Endnode 1	lon	lat	NaN	NaN	NaN	NaN	-1	-1	AA1	AA2	BB1	BB2	CC1	
	Endnode 2	lon	lat	NaN	NaN	NaN	NaN	-1	-1	AA3	AA4	BB3	BB4	CC3	
Way 4	Endnode 1	lon	lat	NaN	NaN	NaN	NaN	NaN	NaN	-1	-1	ZZ1	ZZ2	YY1	

Abbildung 7.3: Schematischer Aufbau der Distanzmatrix

2 des *Way 2* sowie die *lon/lat* Elemente des *Endnode 2* des *Way 4* benötigt.

Um die Berechnung effizient durchführen zu können, wird die Distanzmatrix zuerst mit *NaN* Elementen initialisiert und danach hauptzeilenweise in einer *for*-Schleife zusammengesetzt. Die simultane Berechnung einer Hauptzeile nutzt den Vorteil der Vektorisierung von MATLAB aus [8], welche besagt, dass Rechenoperationen in Vektorform von MATLAB effizienter durchgeführt werden als dieselbe Berechnung iterativ in Skalarform.

Die Funktion *my_calc_distances_between_endpoints()* übernimmt den Datensatz aller Way-Elemente in der Variabel *data* und gibt die berechnete Distanzmatrix *M* retour. Mithilfe der Variabel *bool* können optional Histogramme der Längen der Distanzmatrix *M* dargestellt werden.

7.2 my_calc_stacked_endnodes()

Die zweite Funktion im Modul *Gruppierung*, *my_calc_stacked_endnodes()*, sucht Endpunktpärchen (♡), welche übereinander liegen.

```
function [data, nodes_stacked_pairs] ...
    = my_calc_stacked_endnodes(data, distances, bool)

    % DESCRIPTION
    % This function searches every distance combination between all
    % endpoints which have the value "0", which means that two endpoints
    % have the same coordinates and are stacked on top of each other. (This
    % is easy to do and drastically increases computing performance in
    % upcoming functions). Since in the distance Matrix M every endnode
    % needs two rows/columns, the original "ID" will be recalculate to get
    % the right way element. To the dataset a boolean flag will be added to
    % determine if endnode1/2 is stacked. A list of all pairs of stacked
    % endnodes will be return for further grouping. Optionally data of all
    % stacked endnodes can be plotted and also a histogram of how many
    % endnodes are stacked can be shown.
    %
    % INPUT
    % data ... input dataset
    % distances ... distance Matrix M with distances between all endnodes
    % bool ... boolean selector variable to toogle on/off the visualisations
    %
    % OUTPUT
    % data ... updated dataset, new flag: endnode1/2_stacked
    % nodes_stacked_pairs ... a raw list of all pairs of stacked endnodes

    tic
    disp('Start finding all stacked endnodes...')

    %%% Get the way ids of stacked elements
    % Create boolean logical index of all distance combinations witch equal 0
    b_dist_is_zero = distances == 0;

    % if no distance element has value 0, cancel that function since no two
    % endpoints are stacked
    if not(any(any(b_dist_is_zero)))

        % Set all boolean flags to false
        [data(:).node1_stacked] = deal(false);
        [data(:).node2_stacked] = deal(false);

        % Create empty pseudo output
        nodes_stacked_pairs = [];
```

```
43          % Print this information to console
44          fprintf('   ... no endnode is stacked! \n')
45
46          % End that function
47          fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
48          return
49      end
50
51      % Get the indices of this boolean matrix, hence the row/column IDs
52      [nodes_stacked_indices.raw_column, nodes_stacked_indices.raw_row] ...
53          = find(b_dist_is_zero);
54
55      % Combine the row(y)- and column(x)-indices in one list and sort them
56      nodes_stacked_indices.raw_combined ...
57          = sort([nodes_stacked_indices.raw_column; ...
58                  nodes_stacked_indices.raw_row]);
59
60      % remove duplicates: extract unique ids and calculate their occurances
61      [nodes_stacked_indices.unique_occurance, nodes_stacked_indices.unique] ...
62          = hist(nodes_stacked_indices.raw_combined, ...
63                 unique(nodes_stacked_indices.raw_combined));
64
65      % match dimensions, hence transpose "unique_occurance"
66      nodes_stacked_indices.unique_occurance ...
67          = nodes_stacked_indices.unique_occurance';
68
69      fprintf('   ... %d endnodes are stacked! \n', ...
70          size(nodes_stacked_indices.unique, 1))
71
72      % Create new table, first column: unique indices
73      nodes_stacked = table(nodes_stacked_indices.unique, ...
74                            'VariableNames', {'index'});
75
76      % Convert indices to Wayelement ID
77      nodes_stacked.way_ID = ceil(nodes_stacked.index / 2);
78
79      % Convert indices to boolean indicator if it's endnode1 (true) or 2 (false)
80      nodes_stacked.endnode1 = mod(nodes_stacked.index, 2);
81
82      % return all pairs, to group them later in another function
83      nodes_stacked_pairs ...
84          = [nodes_stacked_indices.raw_row, nodes_stacked_indices.raw_column];
85
86
87
88      %%% Add stacked information to dataset
89      % Start with first index
90      i_stacked_nodes = 1;
91
92      % Initialize frequent used variabel
93      numel_way_IDs = numel(nodes_stacked.way_ID);
94      % go through all ways in data_ways_selected
```

```
95      for i_ways = 1 : size(data, 1)
96
97          % Catch out-of-index-error if very last index (last way, endnode 2)
98          % is stacked: Then break the loop
99          if i_stacked_nodes > size(nodes_stacked, 1)
100
101             % change variable so the next if check fails
102             i_stacked_nodes = i_stacked_nodes - 1;
103         end
104
105         % Does current way (from data_ways_selected) contain a stacked endnode?
106         if i_ways == nodes_stacked.way_ID(i_stacked_nodes)
107         % Yes, at least one endnode is stacked
108
109             % Are both endnodes stacked?
110             % Check if it's not the last way_ID AND next way_ID is identical
111             if (i_stacked_nodes < numel_way_IDs) ...
112                     && ((nodes_stacked.way_ID(i_stacked_nodes) ...
113                         == nodes_stacked.way_ID(i_stacked_nodes + 1)))
114
115                 % Yes, both endnodes are stacked
116                 data(i_ways).node1_stacked = true;
117                 data(i_ways).node2_stacked = true;
118
119                 % Skip one index, since we just set two nodes
120                 i_stacked_nodes = i_stacked_nodes + 1;
121
122             % No, not both. So only one. Is endnode 1 stacked?
123             elseif nodes_stacked.endnode1(i_stacked_nodes)
124
125                 % Yes, endnode 1 is stacked
126                 data(i_ways).node1_stacked = true;
127                 data(i_ways).node2_stacked = false;
128
129             % No, endnode 1 is not stacked
130             else
131
132                 % So endnode 2 must be stacked
133                 data(i_ways).node1_stacked = false;
134                 data(i_ways).node2_stacked = true;
135             end
136
137             % select next index to compare against way_ID
138             i_stacked_nodes = i_stacked_nodes + 1;
139
140         else
141         % No, none of both endnodes are stacked
142             data(i_ways).node1_stacked = false;
143             data(i_ways).node2_stacked = false;
144         end
145     end
146     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
```

```
147
148     % Visualize this stacked data
149     if bool.plot_stacked_endnodes
150         tic
151         disp('Start visualizing all stacked endnodes (takes a few seconds) ...')
152
153         % Extract all nodes
154         x = [[data.x1]; [data.x2]];
155         y = [[data.y1]; [data.y2]];
156
157         % Extract node1 if it is stacked, else ignore it
158         x_node1_stacked = x(1, [data.node1_stacked]);
159         y_node1_stacked = y(1, [data.node1_stacked]);
160
161         % Extract node2 if it is stacked, else ignore it
162         x_node2_stacked = x(2, [data.node2_stacked]);
163         y_node2_stacked = y(2, [data.node2_stacked]);
164
165         % Plot all nodes, highlight node1 and node2 if stacked
166         figure
167         hold on
168         title('All ways with endnodes STACKED on XY-Map'), grid on
169         xlabel('x - distance from midpoint [km]')
170         ylabel('y - distance from midpoint [km]')
171
172         plot(x, y, 'ok-')
173         plot(x_node1_stacked, y_node1_stacked, 'xr');
174         plot(x_node2_stacked, y_node2_stacked, '+b');
175
176         fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
177     end
178
179
180     % plot histogram how many endnodes are stacked
181     if bool.histogram_stacked_endnodes
182         figure
183         histogram(nodes_stacked_indices.unique_occurance + 1)
184         title('Stacked endnodes: If stacked, how many are stacked?')
185         xlabel('Nodes stacked on top of each other')
186         ylabel('Number of different positions this occurs in')
187     end
188 end
```

Matlab Code 7.2: Funktion *my_calc_stacked_endnodes*()

Dazu wird die Distanzmatrix aus Funktion *my_calc_distances_between_endpoints()* mittels *logical indexing* nach Distanzen, welche exakt dem Wert *0* entsprechen, durchsucht.

Die Position dieser Felder wird gespeichert und den jeweiligen Endknoten als *field node1_stacked* bzw. *node2_stacked* hinzugefügt.

Eine Liste mit den jeweiligen Pärchen, sprich welcher Endknoten liegt genau über welchem Endknoten, wird erstellt und von der Funktion zurück gegeben.

Optional kann ein Histogramm der Anzahl übereinanderliegenden Endpunkte erstellt werden. Ebenso können auf einer Karte Endpunkte, auf die eben dies zutrifft, visualisiert werden.

7.3 my_calc_neighbouring_endnodes()

Die dritte Funktion, *my_calc_neighbouring_endnodes()*, findet alle Endpunktpärchen welche ein einem bestimmten Radius zueinander liegen.

```
1  function [data, nodes_neighbouring_pairs] ...
2      = my_calc_neighbouring_endnodes(data, distances, ...
3                                      neighbourhood_threshold, bool)
4      % DESCRIPTION
5      % This function searches every distance combination between all
6      % endpoints which have a distance value bigger than "0" (the "0" case
7      % was covered before) and lower then the treshold in
8      % "neighbourhood_treshhold", which means that two endpoints
9      % are in the vicinity, aka neighbourhood, to each other.
10     % Since in the distance Matrix M every endnode needs two rows/columns,
11     % the original "ID" will be recalculate to get the right way element.
12     % To the dataset a boolean flag will be added to determine if endnode1/2
13     % is in a neighbourhood. A list of all pairs of neighbouring endnodes will
14     % be return for further grouping. Optionally data of all
15     % neighbouring endnodes can be plotted and also a histogram of how many
16     % endnodes are in a neighbourhood can be shown.
17     %
18     % INPUT
19     % data ... input dataset
20     % distances ... distance Matrix M which contains distances between all
21     %               endnodes
22     % neighbourhood_threshold ... threshold-radius to determine if a
23     %                             endnode is in a neighbourhood or not
```

```
24      % bool ... boolean selector variable to toogle on/off the visualisations
25      %
26      % OUTPUT
27      % data ... updated dataset, new flag: endnode1/2_neighbour
28      % nodes_neighbouring_pairs ... list of all pairs of neighbouring endnodes
29
30      tic
31      disp('Start finding all neighbouring endnodes...')
32
33      %%% Get IDs of all neighbouring endnodes
34      % Create boolean logical index of all combinations which are in
35      % neighourhood, but still not stacked
36      b_dist_neighbourhood = distances < neighbourhood_threshold & distances > 0;
37
38      % if no element is in neighbourhood region, cancel that function
39      if not(any(any(b_dist_neighbourhood)))
40
41          % Set all boolean flags to false
42          [data(:).node1_neighbour] = deal(false);
43          [data(:).node2_neighbour] = deal(false);
44
45          % Create empty pseudo output
46          nodes_neighbouring_pairs = [];
47
48          % Print this information to console
49          fprintf('   ... no endnode is in a neighbourhood! \n')
50
51          % End that function
52          fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
53          return
54      end
55
56      % Get the indices of this boolean matrix, hence the ids of elements
57      [nodes_neighbour_indices.raw_column, nodes_neighbour_indices.raw_row] ...
58          = find(b_dist_neighbourhood);
59
60      % Combine the row(y)- and column(x)-indices in one list and sort them
61      nodes_neighbour_indices.raw_combined ...
62          = sort([nodes_neighbour_indices.raw_column; ...
63                  nodes_neighbour_indices.raw_row]);
64
65      % remove duplicates: extract unique ids and calculate their occurances
66      [nodes_neighbour_indices.unique_occurance, nodes_neighbour_indices.unique]
67          = hist(nodes_neighbour_indices.raw_combined, ...
68                 unique(nodes_neighbour_indices.raw_combined));
69
70      % match dimensions, hence transpose "unique_occurance"
71      nodes_neighbour_indices.unique_occurance ...
72          = nodes_neighbour_indices.unique_occurance';
73
74      fprintf('   ... %d endnodes are in same neighbourhood! \n', ...
75          size(nodes_neighbour_indices.unique, 1))
```

```
76      % Create new table, first column: unique occuring indices
77      nodes_neighbouring = table(nodes_neighbour_indices.unique, ...
78                            'VariableNames', {'index'});
79
80      % Convert indices to Wayelement ID
81      nodes_neighbouring.way_ID = ceil(nodes_neighbouring.index / 2);
82
83      % Convert indices to boolean indicator if it's endnode1 (true) or 2 (false)
84      nodes_neighbouring.endnode1 = mod(nodes_neighbouring.index, 2);
85
86      % return all pairs, to group them later
87      nodes_neighbouring_pairs ...
88          = [nodes_neighbour_indices.raw_row, nodes_neighbour_indices.raw_column];
89
90
91
92      %%% Add neighbouring information to dataset
93      % Start with first index
94      i_neighbouring_nodes = 1;
95
96      % Initialize frequent used variabel
97      numel_way_IDs = numel(nodes_neighbouring.way_ID);
98
99      % go through all ways in data_ways_selected
100     for i_ways = 1 : size(data, 1)
101
102         % Catch out-of-index-error if very last index (last way, endnode 2)
103         % is a neighbour: Then break the loop
104         if i_neighbouring_nodes > size(nodes_neighbouring, 1)
105
106             % change variable so the next if check fails
107             i_neighbouring_nodes = i_neighbouring_nodes - 1;
108         end
109
110         % Contains current way a neighbouring endnode?
111         if i_ways == nodes_neighbouring.way_ID(i_neighbouring_nodes)
112         % Yes, at least one endnode is in neighbourhood
113
114             % Are both endnodes neighbours?
115             % Check if it isnt the last way_ID AND upcoming way_ID is identical
116             if i_neighbouring_nodes < numel_way_IDs ...
117                     && (nodes_neighbouring.way_ID(i_neighbouring_nodes) ...
118                          == nodes_neighbouring.way_ID(i_neighbouring_nodes+1))
119
120                 % Yes, both endnodes are neighbours
121                 data(i_ways).node1_neighbour = true;
122                 data(i_ways).node2_neighbour = true;
123
124                 % Skip one index, since we just set two nodes
125                 i_neighbouring_nodes = i_neighbouring_nodes + 1;
126
127
```

```
128             % No, not both. So only one. Is endnode 1 a neighbour?
129             elseif nodes_neighbouring.endnode1(i_neighbouring_nodes)
130 
131                 % Yes, endnode 1 is a neighbour
132                 data(i_ways).node1_neighbour = true;
133                 data(i_ways).node2_neighbour = false;
134 
135             % No, endnode 1 is not a neighbour
136             else
137 
138                 % So endnode 2 must be a neighbour
139                 data(i_ways).node1_neighbour = false;
140                 data(i_ways).node2_neighbour = true;
141             end
142 
143             % select next index to compare against way_ID
144             i_neighbouring_nodes = i_neighbouring_nodes + 1;
145 
146         else
147         % No, none of both endnodes is a stacked one
148             data(i_ways).node1_neighbour = false;
149             data(i_ways).node2_neighbour = false;
150         end
151     end
152 
153     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
154 
155     % Visualize this neighbouring data
156     if bool.plot_neighbouring_endnodes
157         tic
158         disp(['Start visualizing all neighbouring endnodes ' ...
159               '(takes a few seconds) ...'])
160 
161         % Extract all nodes
162         x = [[data.x1]; [data.x2]];
163         y = [[data.y1]; [data.y2]];
164 
165         % Extract node1 if it is in a neighbourhood, else ignore it
166         x_node1_neighbour = x(1, [data.node1_neighbour]);
167         y_node1_neighbour = y(1, [data.node1_neighbour]);
168 
169         % Extract node2 if it is in a neighbourhood, else ignore it
170         x_node2_neighbour = x(2, [data.node2_neighbour]);
171         y_node2_neighbour = y(2, [data.node2_neighbour]);
172 
173         % Plot all nodes, highlight node1 and node2 if in neighourhood
174         figure
175         hold on
176         title('All ways with endnodes NEIGHBOURING on XY-Map'), grid on
177         xlabel('x - distance from midpoint [km]')
178         ylabel('y - distance from midpoint [km]'),
179 
```

```
180         plot(x, y, 'ok-');
181         plot(x_node1_neighbour, y_node1_neighbour, '*g');
182         plot(x_node2_neighbour, y_node2_neighbour, '*g');
183 
184         fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
185     end
186 
187     % plot histogram how many endnodes are in the neighbourhood
188     if bool.histogram_neighbouring_endnodes
189         figure
190         histogram(nodes_neighbour_indices.unique_occurance + 1, ...
191                   max(nodes_neighbour_indices.unique_occurance))
192         title('Neighbouring endnodes: How many will be in one group?')
193         xlabel('Number of nodes which will be grouped together')
194         ylabel('Number of different positions this occurs in')
195     end
196 end
```

Matlab Code 7.3: Funktion *my_calc_neighbouring_endnodes*()

Dazu wird die Distanzmatrix aus Funktion *my_calc_distances_between_endpoints()* mittels *logical indexing* nach Distanzen, welche zwischen dem Wert *0* und der maximalen Gruppierungsdistanz aus Variable *neighbourhood_threshold* entsprechen, durchsucht.

Der Wert von *neighbourhood_threshold* wird am Anfang des Programms, beschrieben in Abschnitt 3.2 eingestellt.

Die Position dieser Felder wird gespeichert und den jeweiligen Endknoten als *field node1_neighbour* bzw. *node2_neighbour* hinzugefügt.

Eine Liste mit den jeweiligen Pärchen, sprich welcher Endknoten liegt im Einzugsradius neben welchem Endknoten, wird erstellt und von der Funktion zurück gegeben.

Optional kann ein Histogramm der Anzahl nebeneinanderliegenden Endpunkte erstellt werden. Ebenso können auf einer Karte Endpunkte, auf die eben dies zutrifft, visualisiert werden.

7.4 my_group_nodes()

Die vierte Funktion, *my_group_nodes()*, fasst die verschiedenen über- oder nebeneinanderliegenden Endpunktpärchen in Gruppen zusammen.

```
function list ...
    = my_group_nodes(pairs_input)

    % DESCRIPTION
    % This function takes as a input a list of pairs (stacked_pairs or
    % neighbouring_pairs) to group them: If A is a pair with B, C a pair
    % with D and B a pair with D, that means that there will be one group with
    % A, B, C and D. This function checks all cases, hence creates new
    % groups, adds elements to an existing group and even concentate groups:
    % If there is allready a group1 with A, B, C, D; and another
    % group2 with X, Y, Z; and one pair C with Y comes up, then the new group
    % will be A, B, C, D, X, Y, Z. So group2 will be added to group1 and
    % group2 then deleted.
    %
    % INPUT
    % pairs_input ... list of pairs
    %
    % OUTPUT
    % list ... a list of groups made out of the pairs from pairs_input

    tic
    fprintf(['Start grouping all pairs from "%s" ' ...
             '(may take a few seconds)... \n'], inputname(1))

    % Initialize empty list
    list = [];

    % sort pairs horizontally
    pairs_sorted_horizontally = sort(pairs_input, 2);

    % sort pairs vertically in regards to 1st column
    data_paired = sortrows(pairs_sorted_horizontally);

    % Go through every pair, which consists of "partner1" and "partner2"
    for i_pairs = 1 : size(data_paired, 1)

        % Create "partner1" and "partner2 from current pair
        partner1 = data_paired(i_pairs, 1);
        partner2 = data_paired(i_pairs, 2);

        % If partner1 is already in any group, save its row, otherwise return 0
        [row_partner1, ~] = find(list == partner1);
```

```
43          % If partner2 is already in any group, save its row, otherwise return 0
44          [row_partner2, ~] = find(list == partner2);
45
46          % So, is partner1 already in any group?
47          if row_partner1
48          % Yes, partner1 is in a group
49
50              % So, Is partner2 in any group too?
51              if row_partner2
52              % Yes, both partner1 and 2 are in the same or in seperate groups
53
54                  % Are both in the same group?
55                  if row_partner2 == row_partner1
56                  % Yes, nothing to do
57
58                      % Return to next iteration of for-loop
59                      continue
60
61                  else
62                  % No, special case:
63                      % Two subgroups have formed, and need to be concatenated.
64                      % So the whole group of partner 2 ("line2") will be copied
65                      % to the end of the group of partner1 ("line1")
66
67                      % Determine the values and number of values of line2
68                      src_nonzero_values = nonzeros(list(row_partner2, :))';
69                      src_num_nonzero_values = numel(src_nonzero_values);
70
71                      % Calculate the exact position in line1 to where the
72                      % nonzero values of line2 will be copied too
73                      dest_num_nonzero_values = nnz(list(row_partner1, :));
74                      dest_start_pos = dest_num_nonzero_values + 1;
75                      dest_end_pos ...
76                          = dest_num_nonzero_values + src_num_nonzero_values;
77
78                      % Copy the values of line2 to the end of line1
79                      list(row_partner1, dest_start_pos : dest_end_pos) ...
80                          = src_nonzero_values;
81
82                      % Sort line1
83                      sorted_values = sort(nonzeros(list(row_partner1, :))', 2);
84                      trailing_zeros = zeros(sum(list(row_partner1, :) == 0), 1)';
85                      list(row_partner1, :) = [sorted_values, trailing_zeros];
86
87                      % Delete line2
88                      list(row_partner2, :) = [];
89                  end
90
91              else
92              % No, partner2 is in no group yet
93
94
```

```
95                  % Add partner2 at the end of partner1's row
96                  num_values = nnz(list(row_partner1, :));
97                  list(row_partner1, num_values + 1) = partner2;
98
99                  % Sort partner1's row
100                 sorted_values = sort(nonzeros(list(row_partner1,:))', 2);
101                 trailing_zeros = zeros(sum(list(row_partner1,:) == 0), 1)';
102                 list(row_partner1,:) = [sorted_values, trailing_zeros];
103             end
104
105         % No, partner1 is in no group. Is partner2 in any group?
106         elseif row_partner2
107         % Yes, at least partner2 is in a group
108
109             % Add partner1 at the end of partner2's row
110             num_values = nnz(list(row_partner2, :));
111             list(row_partner2, num_values + 1) = partner1;
112
113             % Sort partner2's row
114             sorted_values = sort(nonzeros(list(row_partner2, :))', 2);
115             trailing_zeros = zeros(sum(list(row_partner2, :) == 0), 1)';
116             list(row_partner2,:) = [sorted_values, trailing_zeros];
117
118         else
119         % No, neither partner1 nor partner2 are in any group
120
121            % Add new group with both partner1 and partner2
122            trailing_zeros = zeros(1, size(list, 2) - 2);
123            newrow = [[partner1, partner2], trailing_zeros];
124            list = [list; newrow];
125         end
126     end
127
128     fprintf(['   ... %d nodes will be grouped together in %d grouped nodes,'...
129              '\n       with an average of %4.2f nodes ' ...
130              'per grouped node. \n'], ...
131              sum(sum(list ~= 0, 2)), numel(sum(list ~= 0, 2)), ...
132              sum(sum(list ~= 0, 2)) / numel(sum(list ~= 0, 2)))
133
134     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
135 end
```

Matlab Code 7.4: Funktion *my_group_nodes*()

In der Funktion *my_group_nodes()* werden die über- oder nebeneinanderliegenden Pärchen aus *my_calc_stacked_endnodes()* sowie *my_calc_neighbouring_endnodes()* in gemeinsame Gruppen zusammengefasst.

Dies geschieht durch eine Reihe von *If-Else-Abfragen*. Zuerst wird geprüft, ob ein Partner eines Pärchens bereits in irgendeiner Gruppe ist.

Trifft dies nicht zu, so wird eine neue Gruppe mit eben jenem Pärchen erstellt. Trifft dies aber hingegen zu, so wird überprüft, ob der zweite Partner überhaupt in der gleichen Gruppe wie der erste Partner ist.

Ist dies der Fall, so wird dieses Pärchen ignoriert und das nächste Pärchen überprüft. Ist dies nicht der Fall, so wird der zweite Partner des Pärchens in die Gruppe, in der sich bereits der erste Partner befindet, hinzugefügt.

Vereinfacht gesagt sorgt dies dafür, dass wenn A und B, C und D, sowie B und D jeweils ein Pärchen sind, schlussendlich eine gemeinsame Gruppe mit A, B, C und D gebildet wird.

Sollten sich über die Zeit jedoch eine Gruppe mit A, B, C und D, sowie eine andere Gruppe mit X, Y und Z gebildet haben, es sich aber herausstellen, dass C mit Y ein Pärchen ist, so werden beide Gruppen zusammengelegt und enthalten damit A, B, C, D, X, Y und Z.

Die Funktion gibt eine Liste mit den finalen Gruppen zurück.

Diese Funktion wird zwei Mal aufgerufen, einmal berechnet sie die Gruppen von übereinander liegenden Pärchen, einmal von nebeneinander liegenden Pärchen.

7.5 my_group_stacked_endnodes()

Die fünfte Funktion, *my_group_stacked_endnodes()*, speichert die Information von übereinanderliegenden Endpunkte im Datensatz.

```
function data ...
    = my_group_stacked_endnodes(data, nodes_stacked_grouped)

    % DESCRIPTION
    % This function gets the ID and lon/lat/x/y coordinates of the first
    % member of a stacked group and copies it to all the other members of
    % that group, therefore giving all members the same node ID and
    % (exactly) same coordinate.
    %
    % INPUT
    % data ... original dataset
    % nodes_stacked_grouped ... list of nodes which are stacked
    %
    % OUTPUT
    % data ... dataset where all stacked nodes have identica group id

    disp('Start adding coordinates of stacked groups... ')

    % Preallocate new fields, so that they are in the right order
    [data(:).ID_node1_grouped] = deal(NaN);
    [data(:).ID_node2_grouped] = deal(NaN);
    [data(:).lon1_grouped] = deal(NaN);
    [data(:).lat1_grouped] = deal(NaN);
    [data(:).lon2_grouped] = deal(NaN);
    [data(:).lat2_grouped] = deal(NaN);
    [data(:).x1_grouped] = deal(NaN);
    [data(:).y1_grouped] = deal(NaN);
    [data(:).x2_grouped] = deal(NaN);
    [data(:).y2_grouped] = deal(NaN);

    % extract first group coordinates of all stacked groups
    for i_group = 1 : size(nodes_stacked_grouped, 1)

        % Save the node_ID of first group member
        i_node_ID = nodes_stacked_grouped(i_group, 1);

        % Convert the node_ID in the way_ID
        i_way_ID = ceil(i_node_ID / 2);

        % Extract from the node_ID the boolean information, if
        % it is node1 (true) or node2 (false)
        b_node1 = mod(i_node_ID, 2);
```

```
43          if b_node1
44               % get ID/coordinates of node 1
45               grouped_node_ID = data(i_way_ID).ID_node1;
46               grouped_lat = data(i_way_ID).lat1;
47               grouped_lon = data(i_way_ID).lon1;
48               grouped_x = data(i_way_ID).x1;
49               grouped_y = data(i_way_ID).y1;
50          else
51              % get ID/coordinates of node 2
52              grouped_node_ID = data(i_way_ID).ID_node2;
53              grouped_lat = data(i_way_ID).lat2;
54              grouped_lon = data(i_way_ID).lon2;
55              grouped_x = data(i_way_ID).x2;
56              grouped_y = data(i_way_ID).y2;
57          end
58
59          % go through every (nonzero) member of that group
60          for i_group_member = 1 : nnz(nodes_stacked_grouped(i_group, :))
61
62              % Save the node_ID of that group member
63              i_node_ID = nodes_stacked_grouped(i_group, i_group_member);
64
65              % Convert the node_ID in the way_ID
66              i_way_ID = ceil(i_node_ID / 2);
67
68              % Extract from the node_ID the boolean information, if
69              % it is node1 (true) or node2 (false)
70              b_node1 = mod(i_node_ID, 2);
71
72              if b_node1
73                  % add the new combined id/lat/lon
74                  data(i_way_ID).ID_node1_grouped = grouped_node_ID;
75                  data(i_way_ID).lat1_grouped = grouped_lat;
76                  data(i_way_ID).lon1_grouped = grouped_lon;
77                  data(i_way_ID).x1_grouped = grouped_x;
78                  data(i_way_ID).y1_grouped = grouped_y;
79              else
80                  % add the new combined id/lat/lon
81                  data(i_way_ID).ID_node2_grouped = grouped_node_ID;
82                  data(i_way_ID).lat2_grouped = grouped_lat;
83                  data(i_way_ID).lon2_grouped = grouped_lon;
84                  data(i_way_ID).x2_grouped = grouped_x;
85                  data(i_way_ID).y2_grouped = grouped_y;
86              end
87          end
88      end
89      fprintf('   ... finished! (%5.3f seconds) \n \n' , toc)
90  end
```

Matlab Code 7.5: Funktion *my_group_stacked_endnodes*()

Zu Beginn der Funktion werden einem jeden Way-Element neue Felder hinzugefügt, in denen ggf. die Koordinaten neuer, gruppierter Endpunkte gespeichert werden. Diese Felder werden vorerst mit *NaN* ausgefüllt.

Vom jeweils ersten Element einer Gruppe werden die ID des Knotens, die Längen- und Breitengradinformationen sowie die *x/y*-Koordinaten gespeichert und einem jedem nachfolgenden Gruppenmitglied zugeordnet.

Somit bekommen alle Mitglieder einer Gruppe von übereinander liegenden Endpunkten identische Koordinaten und eine identische Knoten-ID.

7.6 my_group_neighbouring_endnodes()

Die sechste Funktion, *my_group_neighbouring_endnodes()*, speichert die Information von nebeneinanderliegende Endpunkte im Datensatz.

```
function [data, grouped_xy_coordinates] ...
    = my_group_neighbouring_endnodes(data, nodes_neighbouring_grouped, ...
                                     degrees_to_km_conversion)

    % DESCRIPTION
    % This function extracts all lon/lat coordinates of all members for every
    % neighbouring group, then calculates the mean lon/lat value and copies
    % it to every group member. Then the x/y values will newly be
    % calculated and too added.
    %
    % INPUT
    % data ... origial input dataset
    % nodes_neighbouring_grouped ... list with nodes grouped
    % degrees_to_km_conversion ... conversion data to calculate x/y coordinates
    %
    % OUTPUT
    % data ... updated dataset with grouped fields
    % grouped_xy_coordinates ... list of x/y coordinates of grouped nodes,
    %                            this will be used in a plot later

    tic
    disp('Start adding grouping neighbours... ')

    % Precalculation for improved code readabilty
    num_of_groups = size(nodes_neighbouring_grouped, 1);

    % Preallocate output (otherwise error if no ways will be grouped)
    grouped_xy_coordinates = [];

    % extract all coordinates of all neighbouring group members
    for i_group = 1 : num_of_groups

        % go through every (nonzero) member of that group
        for i_group_member = 1 : nnz(nodes_neighbouring_grouped(i_group, :))

            % Save the node_ID of that group member
            i_node_ID = nodes_neighbouring_grouped(i_group, i_group_member);

            % Convert the node_ID in the way_ID
            i_way_ID = ceil(i_node_ID / 2);


```

```
43              % Extract from the node_ID the boolean information, if
44              % it is node1 (true) or node2 (false)
45              b_node1 = mod(i_node_ID, 2);
46
47              if b_node1
48                  % get coordinates of node 1
49                  lon = data(i_way_ID).lon1;
50                  lat = data(i_way_ID).lat1;
51                  x = data(i_way_ID).x1;
52                  y = data(i_way_ID).y1;
53              else
54                  % get coordinates of node 2
55                  lon = data(i_way_ID).lon2;
56                  lat = data(i_way_ID).lat2;
57                  x = data(i_way_ID).x2;
58                  y = data(i_way_ID).y2;
59              end
60
61              % Save the coordinates of that group member in alternating manner
62              % lon of member1 in column 1, lat of member1 in column 2,
63              % lon of member2 in column 3, lat of member2 in column 4, etc.
64              grouped_lonlat_coordinates(i_group, i_group_member * 2 - 1) = lon;
65              grouped_lonlat_coordinates(i_group, i_group_member * 2) = lat;
66
67              % Do the same with x/y
68              % x of member1 in column 1, y of member1 in column 2,
69              % x of member2 in column 3, y of member2 in column 4, etc.
70              grouped_xy_coordinates(i_group, i_group_member * 2 - 1) = x;
71              grouped_xy_coordinates(i_group, i_group_member * 2) = y;
72          end
73      end
74
75      % Preallocate the list of mean coordinates
76      list_coordinates_mean = zeros(size(nodes_neighbouring_grouped, 1), 2);
77
78      % calculate mean lon/lat for every group
79      for i_group = 1 : num_of_groups
80
81          % save all lon/lat
82          lon_data = grouped_lonlat_coordinates(i_group, 1:2:end);
83          lat_data = grouped_lonlat_coordinates(i_group, 2:2:end);
84
85          % remove all zeros, since they would miscalculate the mean value
86          lon_data = lon_data(lon_data ~= 0);
87          lat_data = lat_data(lat_data ~= 0);
88
89          % calculate the mean value and save it
90          list_coordinates_mean(i_group, 1:2) = [mean(lon_data), mean(lat_data)];
91      end
92
93      % Add the grouped coordinates of every group to dataset
94      for i_group = 1 : num_of_groups
```

```
95          % go through every (nonzero) member of that group
96          for i_group_member = 1 : nnz(nodes_neighbouring_grouped(i_group, :))
97
98              % Save the node_ID of that group member
99              i_node_ID = nodes_neighbouring_grouped(i_group, i_group_member);
100
101             % Convert the node_ID in the way_ID
102             i_way_ID = ceil(i_node_ID / 2);
103
104             % Extract from the node_ID the boolean information, if
105             % it is node1 (true) or node2 (false)
106             b_node1 = mod(i_node_ID, 2);
107
108             if b_node1
109                 % add the new combined id/lat/lon
110                 data(i_way_ID).ID_node1_grouped = i_group;
111                 data(i_way_ID).lon1_grouped = list_coordinates_mean(i_group, 1);
112                 data(i_way_ID).lat1_grouped = list_coordinates_mean(i_group, 2);
113             else
114                 % add the new combined id/lat/lon
115                 data(i_way_ID).ID_node2_grouped = i_group;
116                 data(i_way_ID).lon2_grouped = list_coordinates_mean(i_group, 1);
117                 data(i_way_ID).lat2_grouped = list_coordinates_mean(i_group, 2);
118             end
119         end
120     end
121
122
123     %%% Add x/y coordinates to the new groups
124     % Extract input data into varables
125     km_per_lon_deg = degrees_to_km_conversion(1);
126     km_per_lat_deg = degrees_to_km_conversion(2);
127     mean_lon = degrees_to_km_conversion(3);
128     mean_lat = degrees_to_km_conversion(4);
129
130     % calculate the difference in degree for each point from mean
131     delta_lon1 = [data.lon1_grouped]' - mean_lon;
132     delta_lon2 = [data.lon2_grouped]' - mean_lon;
133     delta_lat1 = [data.lat1_grouped]' - mean_lat;
134     delta_lat2 = [data.lat2_grouped]' - mean_lat;
135
136     % convert the delta_degree into delta_kilometer
137     x1 = num2cell(delta_lon1 * km_per_lon_deg);
138     x2 = num2cell(delta_lon2 * km_per_lon_deg);
139     y1 = num2cell(delta_lat1 * km_per_lat_deg);
140     y2 = num2cell(delta_lat2 * km_per_lat_deg);
141
142     % save delta_km to data_ways (raw data or new combined coordinates)
143     [data.x1_grouped] = x1{:};
144     [data.y1_grouped] = y1{:};
145     [data.x2_grouped] = x2{:};
146     [data.y2_grouped] = y2{:};
```

```
147     % If id_node1/2_grouped does not exist, this script will calculate
148     % x1/2_new and y1/2_new with wrong (=0) lat1/2_grouped and lat1/2_grouped.
149     % Correct for it by deleting those values
150     for i_ways = 1 : numel(data)
151
152         % set node1_new id/x/y empty
153         if isnan(data(i_ways).ID_node1_grouped)
154             data(i_ways).ID_node1_grouped = [];
155             data(i_ways).x1_grouped = [];
156             data(i_ways).y1_grouped = [];
157             data(i_ways).lon1_grouped = [];
158             data(i_ways).lat1_grouped = [];
159         end
160
161         % set node1_new id/x/y empty
162         if isnan(data(i_ways).ID_node2_grouped)
163             data(i_ways).ID_node2_grouped = [];
164             data(i_ways).x2_grouped = [];
165             data(i_ways).y2_grouped = [];
166             data(i_ways).lon2_grouped = [];
167             data(i_ways).lat2_grouped = [];
168         end
169     end
170
171     fprintf('   ... finished! (%5.3f seconds) \n \n' , toc)
172 end
```

Matlab Code 7.6: Funktion *my_group_neighbouring_endnodes*

In einer lokalen Liste werden sämtliche Koordinaten von allen Mitgliedern einer Gruppe gespeichert. Von all diesen Koordinaten wird der Mittelwert gebildet, sozusagen der Schwerpunkt der Fläche den diese Koordinaten auf einer *2D*-Ebene besitzen.

Diese Mittelwert-Koordinate wird die Koordinate des neuen, gemeinsamen Endknoten. Da diese Koordinate vorerst nur in Längen-/Breitengradinformation vorliegt, wird sie in *x/y*-Koordinate umgerechnet.

Beide Koordinaten werden nun einem jeden Way-Elemente dieser Gruppe als neue Koordinate zugewiesen.

7.7 my_add_final_coordinates()

Die siebte Funktion, *my_add_final_coordinates()*, wählt die finalen Koordinaten eines jeden Endknoten aus.

```
function data ...
    = my_add_final_coordinates(data)


    % DESCRIPTION
    % This function selects the final coordinates: If one or both endnodes
    % got grouped (because they were stacked and/or in a neighourhood),
    % those new grouped coordinates will be the final coordinates. If not,
    % then the original coordinates will be taken as the final coordinates.
    % The final coordinate will constists the ID, the lon/lat and the x/y
    % coordinates.
    %
    % INPUT
    % data ... original dataset
    %
    % OUTPUT
    % data .. updated dataset with new final coordinates fields


    tic
    disp('Start adding final coordinates...')


    % First, go through all ways and get the new endnode coordinates
    for i_ways = 1 : numel(data)

        % Check if there is a new node 1, if not, take old one
        if isempty(data(i_ways).ID_node1_grouped)
            data(i_ways).ID_node1_final = data(i_ways).ID_node1;
            data(i_ways).lon1_final = data(i_ways).lon1;
            data(i_ways).lat1_final = data(i_ways).lat1;
            data(i_ways).x1_final = data(i_ways).x1;
            data(i_ways).y1_final = data(i_ways).y1;
        else
            data(i_ways).ID_node1_final = data(i_ways).ID_node1_grouped;
            data(i_ways).lon1_final = data(i_ways).lon1_grouped;
            data(i_ways).lat1_final = data(i_ways).lat1_grouped;
            data(i_ways).x1_final = data(i_ways).x1_grouped;
            data(i_ways).y1_final = data(i_ways).y1_grouped;
        end


```

```
43          % Check if there is a new node 2, if not, take old one
44          if isempty(data(i_ways).ID_node2_grouped)
45              data(i_ways).ID_node2_final = data(i_ways).ID_node2;
46              data(i_ways).lon2_final = data(i_ways).lon2;
47              data(i_ways).lat2_final = data(i_ways).lat2;
48              data(i_ways).x2_final = data(i_ways).x2;
49              data(i_ways).y2_final = data(i_ways).y2;
50          else
51              data(i_ways).ID_node2_final = data(i_ways).ID_node2_grouped;
52              data(i_ways).lon2_final = data(i_ways).lon2_grouped;
53              data(i_ways).lat2_final = data(i_ways).lat2_grouped;
54              data(i_ways).x2_final = data(i_ways).x2_grouped;
55              data(i_ways).y2_final = data(i_ways).y2_grouped;
56          end
57      end
58
59      fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
60  end
```

Matlab Code 7.7: Funktion *my_add_final_coordinates*()

Obwohl nun in den Funktionen *my_group_stacked_endnodes()* sowie *my_group_neighbouring_endnodes()* manchen Elementen bereits neue Koordinaten zugewiesen worden sind, besitzen viele Endkonten noch ihre ursprünglichen Koordinaten.

Aus diesem Grund wird in der letzten Funktion dieses Moduls, *my_add_final_coordinates()*, jedem Way-Element die finale ID des Endknotens, die finalen Längen- und Breitengradinformationen sowie die finalen *x/y*-Koordinaten zugewiesen.

Diese finalen Information sind ident zu den Informationen von gruppierten Endknoten, und falls diese nicht vorhanden sind, ident zu den originalen Koordinaten.

Zukünftige Funktionen werden ab diesem Punkt nur noch auf die finalen Koordinaten zugreifen.

8 Funktionen - Modul Export

Das fünfte von sechs Programmmodulen ist das *Export-Modul*. In diesem wird der Datensatz, welcher in den vorherigen Modulen verkleinert und gruppiert wurde, für den Export aufbereitet und schlussendlich als *Excel*-Dateien exportiert.

Diese Aufbereitung besteht aus dem Löschen von kurzen Leitungen, welche in einen singulären Punkt gruppiert wurden, dem Extrahieren sämtlicher Informationen aus dem Datensatz, um spätere manuelle Untersuchungen zu ermöglichen, dem Vervielfachen von Leitungen, welche mehr-systemig sind, und dem Erfassen sämtlicher für den Export notwendiger Informationen sowie dem eigentlichen Export als *Excel*-Dateien.

Der Aufbau des *Export-Moduls* ist schematisch in Abb. 8.1 dargestellt.

Das *Export-Modul* besteht aus vier Funktionen:

1. *my_delete_singular_ways()*, beschrieben in Abschnitt 8.1,
2. *my_get_tags()*, beschrieben in Abschnitt 8.2,
3. *my_add_LtgsID_clone_ways()*, beschrieben in Abschnitt 8.3 und
4. *my_export_excel()*, beschrieben in Abschnitt 8.4

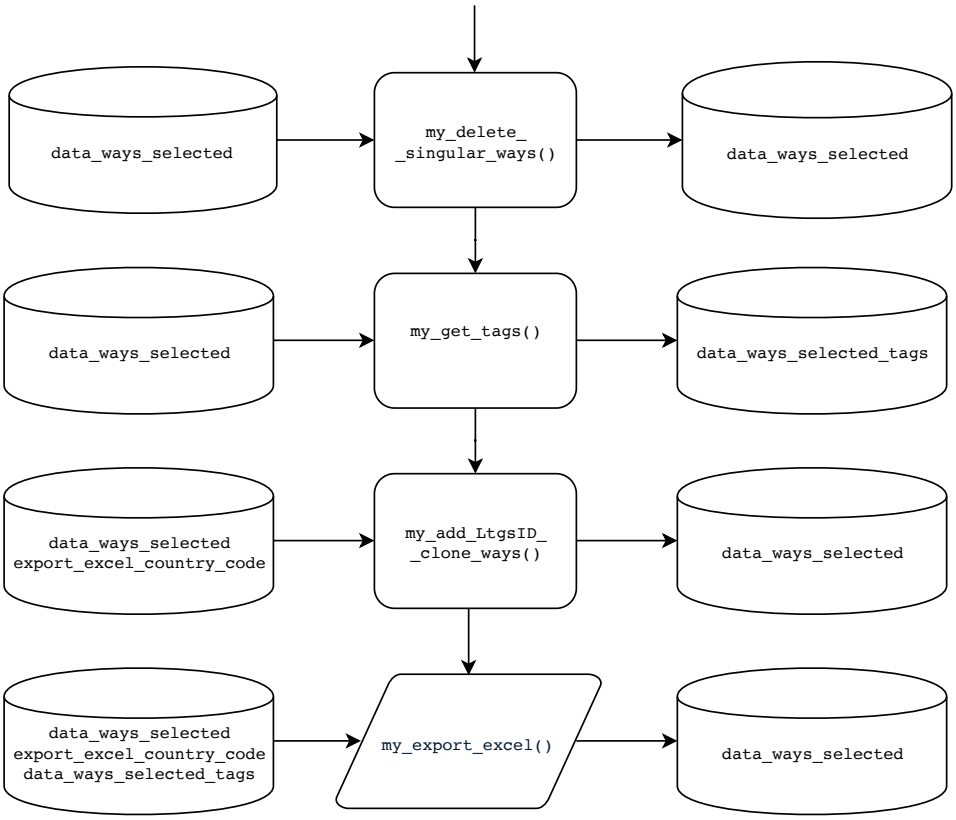


Abbildung 8.1: Schematischer Aufbau Modul 5 von 6

8.1 my_delete_singular_ways()

Die erste Funktion, *my_delete_singular_ways()*, entfernt aus dem Datensatz jene Leitungen, welche in sich selbst gruppiert wurden.

```
function data ...
    = my_delete_singular_ways(data)

    % DESCRIPTION
    % This function deletes all ways which have the same endpoints after
    % grouping, hence got "shrinked" into a singularity. This happens when
    % short lines, for example in a substation, are in the
    % grouping/neighbouring radius of multiply points. Then the endpoints
    % will be concentated in a single endpoint, hence the line
    % "disappears". Since those lines are for no further use, they will be
    % deleted prior exporting.
    %
    % INPUT
    % data ... orginal dataset
    %
    % OUTPUT
    % data ... new dataset with singularity-ways deleted

    tic
    disp('Start deleting ways which have the same endpoints after grouping...')

    % Initialize list of singular Way-Element IDs
    way_IDs_singular = [];

    % Go through all ways
    for i_ways = 1 : numel(data)

        % if both IDs are identical, mark them in a "to-delete" list
        if data(i_ways).ID_node1_final == data(i_ways).ID_node2_final
            way_IDs_singular = [way_IDs_singular, i_ways];
        end
    end

    % Delete all ways which have identical endpoints
    data(way_IDs_singular) = [];

    fprintf('   ... %d ways were deleted! \n', numel(way_IDs_singular))
    fprintf('   ... finished! (%5.3f seconds) \n \n' , toc)
end
```

Matlab Code 8.1: Funktion *my_delete_singular_ways*()

Diese zu entfernenden Leitungen können kurze Leitungen in Umspannwerken, welche in einem einzigen Knoten zusammengefasst wurden, sein, siehe Abb. 2.4 (a), oder kurze Stichleitungen, welche ebenfalls an einem Knoten angeschlossen waren, jedoch in diesen Knoten konzentriert wurden, siehe Abb. 2.4 (b).

8.2 my_get_tags()

In der zweiten Funktion, *my_get_tags()*, werden sämtliche Informationen, die sich im *field* »tags« eines jeden Way-Elementes befinden, separat in der Variable *data_ways_selected_tags* abgespeichert und zurückgegeben.

```
1  function data_tags ...
2      = my_get_tags(data)
3
4      % DESCRIPTION
5      % Since the database of Openstreetmap varies greatly, not all ways have
6      % the same set of "tags". In this matlab script only a few, very
7      % common, tags can be consider in an automatic approach. There may be
8      % still useful information in the tags, so all tags will be copied to a
9      % seperate variable, which then will be exported as "sheet2" in the
10     % final exported excel file. There all tags can manualley be reviewd
11     % for further investigation.
12     %
13     % INPUT
14     % data ... dataset prior to exporting
15     %
16     % OUTPUT
17     % data_tags ... all tags off all way elements
18
19     tic
20     disp('Start extracting all tags from all ways...')
21
22     % Preallocate new struct and start number of tag fields counter
23     data_tags.UID = [];
24     i_ways_tags = 1;
25
26     % go through every way
27     for i_ways = 1 : numel(data)
28
29
```

```
30          % if current way/UID was cloned somewhere in this skript, skip it
31          if i_ways > 1 && data_tags(i_ways_tags - 1).UID == data(i_ways).UID
32              continue;
33          end
34
35          % Save UID
36          data_tags(i_ways_tags).UID = data(i_ways).UID;
37
38          % get all tag-fieldnames of current way
39          fieldnames_current_way = fieldnames(data(i_ways).tags);
40
41          % Add the values of all tags to data_tag
42          for i_fieldname = 1 : numel(fieldnames_current_way)
43
44              % Get current field name
45              current_fieldname = fieldnames_current_way{i_fieldname};
46
47              % Get value of current field name
48              value = data(i_ways).tags.(current_fieldname);
49
50              % copy that value in the correspondending field
51              data_tags(i_ways_tags).(current_fieldname) = value;
52          end
53
54          % increase tags counter (i. E., got to next UID which is in list)
55          i_ways_tags = i_ways_tags + 1;
56      end
57
58      fprintf('   ... finished! (%5.3f seconds) \n \n' , toc)
59  end
```

Matlab Code 8.2: Funktion *my_get_tags*()

Da in jedem Datensatz selbst zwischen den Way-Elementen die Anzahl und Art der jeweiligen »tags« stark variiert, können nicht sämtliche Fälle automatisch in diesem Skript behandelt werden. Besonders das manchmal vorhandene *field* »tags / note« enthält sehr oft wertvolle Kommentare der Autoren, welche Hinweise auf die Art der Leitung und die genaue Implementierung in *ATLANTIS* damit geben.

In der Funktion *my_export_excel()*, beschrieben in Abschnitt 8.4, werden die »tags« aus der Variable *data_ways_selected_tags* zusammen mit dem restlichen Daten als *Excel*-Datei exportiert und sind dort auf dem zweiten Blatt zu finden. Dort können im *Excel* mithilfe der Sortier- und Filterfunktion diese »tags« auf nützliche Informationen untersucht werden.

8.3 my_add_LtgsID_clone_ways()

In der vorletzten Funktion dieses Moduls, *my_add_LtgsID_clone_ways()*, werden mehr-systemige Leitungen vervielfacht und jeder, nun finalen, Leitung eine *LtgsID*, eine eindeutige Leitungs-Identifikationsnummer, zugewiesen.

```
function data_new ...
    = my_add_LtgsID_clone_ways(data, export_excel_country_code)

    % DESCRIPTION
    % This function creates the "LtgsID", "Leitungs-ID"/"way ID" for every
    % way elment. The LtgsID is the main "name" a way has for ATLANTIS. The
    % LtgsID consist of the two character countrycode defined earlier and a
    % 4 digit counter. Ways, which need to be cloned (since they carry
    % more than one system) will be duplicated/tripled or quadrupled and
    % get an updated LtgsID with an "a, b, c" suffix.
    %
    % INPUT
    % data ... input dataset
    % export_excel_country_code ... the two-digit country code of current
    %                                dataset
    %
    % OUTPUT
    % data_new ... new dataset with cloned ways and field "LtgsID"

    tic
    disp('Start adding "LtgsID" and cloning ways...')

    % Initialize variables
    num_of_ways = numel(data);
    num_of_doubled_ways = 0;
    num_of_tripled_ways = 0;
    num_of_quadrupled_ways = 0;
    i_ways_new = 1;

    % Create 'LtgsID'
    LtgsID_Prefix = strcat('LTG', export_excel_country_code);
    LtgsID = strcat(repmat(LtgsID_Prefix, num_of_ways, 1), ...
                    num2str([1 : num_of_ways]', '%04.f'));

    % Add 'LtgsID' to data
    for i_ways = 1 : num_of_ways
        data(i_ways).LtgsID = LtgsID(i_ways);
    end


```

```
42      % Clone ways
43      for i_ways = 1 : num_of_ways + num_of_doubled_ways ...
44                                   + num_of_tripled_ways ...
45                                   + num_of_quadrupled_ways
46
47
48          % Run only if a way needs to be doubled
49          if data(i_ways).systems == 2
50
51              % Get a copy of that way
52              cloned_way_b = data(i_ways);
53
54              % Get LtgsID
55              LtgsID_current = data(i_ways).LtgsID;
56
57              % Update LtgsID on the original way and its clone
58              data(i_ways).LtgsID = strcat(LtgsID_current, 'a');
59              cloned_way_b.LtgsID = strcat(LtgsID_current, 'b');
60
61              % Insert the cloned data way
62              data_new(i_ways_new : i_ways_new+1) = [data(i_ways); cloned_way_b];
63
64              % Increase counter of ways
65              num_of_doubled_ways = num_of_doubled_ways + 1;
66
67              % Skip next way since it is the duplicated one
68              i_ways_new = i_ways_new + 2;
69
70
71          % Run only if a way needs to be tripled
72          elseif data(i_ways).systems == 3
73
74              % Get two copies of that way
75              [cloned_way_b, cloned_way_c] = deal(data(i_ways));
76
77              % Get LtgsID
78              LtgsID_current = data(i_ways).LtgsID;
79
80              % Update LtgsID on the original way and its clone
81              data(i_ways).LtgsID = strcat(LtgsID_current, 'a');
82              cloned_way_b.LtgsID = strcat(LtgsID_current, 'b');
83              cloned_way_c.LtgsID = strcat(LtgsID_current, 'c');
84
85              % Insert the cloned data way
86              data_new(i_ways_new : i_ways_new + 2) ...
87                  = [data(i_ways); cloned_way_b; cloned_way_c];
88
89              % Increase counter of ways
90              num_of_tripled_ways = num_of_tripled_ways + 2;
91
92              % Skip next two ways since they are the duplicated
93              i_ways_new = i_ways_new + 3;
```

```
94          % Run only if a way needs to be quadrupled
95          elseif data(i_ways).systems == 4
96
97
98              % Get two copies of that way
99              [cloned_way_b, cloned_way_c, cloned_way_d] = deal(data(i_ways));
100
101             % Get LtgsID
102             LtgsID_current = data(i_ways).LtgsID;
103
104             % Update LtgsID on the original way and its clone
105             data(i_ways).LtgsID = strcat(LtgsID_current, 'a');
106             cloned_way_b.LtgsID = strcat(LtgsID_current, 'b');
107             cloned_way_c.LtgsID = strcat(LtgsID_current, 'c');
108             cloned_way_d.LtgsID = strcat(LtgsID_current, 'd');
109
110             % Insert the cloned data way
111             data_new(i_ways_new : i_ways_new + 3) ...
112                 = [data(i_ways); cloned_way_b; cloned_way_c; cloned_way_d];
113
114             % Increase counter of ways
115             num_of_quadrupled_ways = num_of_quadrupled_ways + 3;
116
117             % Skip next two ways since they are the duplicated
118             i_ways_new = i_ways_new + 4;
119
120
121         % No way needs to be cloned
122         else
123             % copy current way to new struct
124             data_new(i_ways_new) = data(i_ways);
125
126             % Increase next-way counter
127             i_ways_new = i_ways_new + 1;
128         end
129     end
130
131     % Transpose the new data set to match others in workspace
132     data_new = data_new';
133
134
135     fprintf('   ... %d ways have been doubled, %d tripled, %d quadrupled.\n',...
136             num_of_doubled_ways, num_of_tripled_ways / 2, ...
137             num_of_quadrupled_ways / 3)
138
139     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
140 end
```

Matlab Code 8.3: Funktion *my_add_LtgsID_clone_ways*()

Die in dieser Funktion erstellte *LtgsID* wird in *ATLANTIS* als Hauptidentifikationselement einer Leitung genutzt und ermöglicht Rückschlüsse einer Leitung von *VISU*, der grafischen Oberfläche von *ATLANTIS*, auf die korrekte Zeile in der *Excel*-Datei und von dort auf die in MATLAB verwendete *UID*.

Von dieser *UID* kann über die Variable *data_ways_selected* auf die in *OpenStreetMap* genutzte *ID* eines jeden Way-Elements zurückgeführt werden.

Die *LtgsID* ist somit die dritte und finale Identifikationsnummer einer bis zum Ursprung zurückführbaren Benennungsschema.

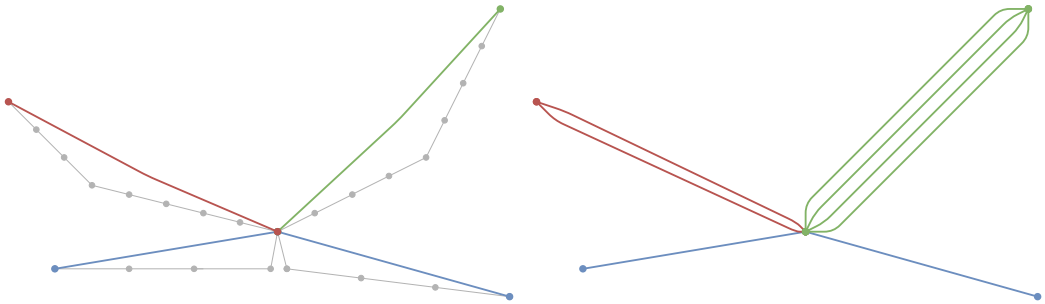
Die *LtgsID* setzt sich aus drei Teilen zusammen:

1. Den zwei Buchstaben des Länderkürzels, welche am Anfang des Programms in der Variable *export_excel_country_code* festgelegt wurde,
2. einer fortlaufenden vierstelligen Zahl mit führenden Nullen, sowie
3. dem optionalen Suffix »a«, »b«, »c« oder »d«.

Das Länderkürzel dient dazu, in *ATLANTIS* Leitungen konkret einem Land zuordnen zu können. Die Festlegung dieses Wertes wird in Kapitel 3.2 beschrieben. Das optionale Suffix kennzeichnet Leitungen, welche nach bestimmten Kriterien vervielfacht wurden, sprich an sich ident, jedoch mehrfach vorhanden sind:

Wurde in der Funktion *my_count_cables()*, beschrieben in Abschnitt 6.3, für ein Way-Element festgestellt, dass es sich um eine zwei-, drei- oder vier-systemige Leitung handelt, so wird diese Leitung in dieser Funktion verdoppelt, verdreifacht oder vervierfacht.

Dies passiert, da davon ausgegangen wird, dass eine zwei-systemige Leitung die doppelte Kapazität einer ein-systemigen Leitung aufweist. Das gleiche gilt sinngemäß für drei- und vier-systemige Leitungen.



(a) Datensatz nur mit Endpunkten (b) Datensatz mit vervielfachten Leitungen

Abbildung 8.2: Schematische Visualisierung der Leitungsvervielfachung

In Abb. 8.2 ist die Vervielfachung von Leitungen schematisch dargestellt. In diesem schematischen Beispiel wird angenommen, dass die rote Leitung sechs und die grüne Leitung zwölf Leiterseile aufweist.

8.4 my_export_excel()

In der letzten Funktion, *my_export_excel()*, werden sämtliche für den Export notwendigen Daten erstellt, die Knoten eindeutig benannt und schlussendlich in zwei *Excel*-Dateien in ein für *ATLANTIS* lesbares Format exportiert.

```
1  function data ...
2      = my_export_excel(data, export_excel_country_code, data_tags)
3
4      % DESCRIPTION
5      % This function exports the data to two excel files. Every unique endnode
6      % will recive a NUID (unique node ID), this too will be added to the
7      % added to the dataset. Columns will be created so that ALTANTIS can
8      % read the excel file. In the annotation ("Bemerkung") column additinal
9      % information will be written if necessary.
10     %
11     % INPUT
12     % data ... the dataset to export
13     % export_excel_country_code ... the countrycode to name LtgsID and NUID
14     % data_tags ... all values off all fields of all tags of all way elements
```

```
15      % OUTPUT
16      % data ... updated dataset (NUID have been added)
17      % (two Excel files in current directory: tbl_Stamm_Leitungen & _Knoten)
18
19
20      tic
21      disp('Start exporting data to Excel files... (may take a few seconds)')
22
23      % Initalize and preallocate variables used in this script
24      num_of_ways = numel(data);
25
26      %%% Assign NUID (=Node Unique ID)
27      % Preallocating variables used in for loop
28      [node1_data, node2_data] = deal(zeros(num_of_ways, 1));
29
30      % Get relevant data of nodes
31      node1_data(:, 1) = [data(:).ID_node1_final];
32      node1_data(:, 2) = [data(:).voltage];
33      node1_data(:, 3) = [data(:).lon1_final];
34      node1_data(:, 4) = [data(:).lat1_final];
35
36      node2_data(:, 1) = [data(:).ID_node2_final];
37      node2_data(:, 2) = [data(:).voltage];
38      node2_data(:, 3) = [data(:).lon2_final];
39      node2_data(:, 4) = [data(:).lat2_final];
40
41      % Get every unique node / voltage level combination
42      nodes_unique = unique([node1_data; node2_data], 'rows', 'first');
43
44      % Create unique IDs for the nodes, "NUID" = Node_Unique_ID
45      num_of_unique_nodes = size(nodes_unique, 1);
46      country_code = repmat(export_excel_country_code, num_of_unique_nodes, 1);
47      counter = string(num2str([1 : num_of_unique_nodes]', '%05.f'));
48      nuid = strcat(country_code, counter);
49
50      % Combine the ID and the list of unique nodes into a conversion file
51      nodes_conversion = [cellstr(nuid), num2cell(nodes_unique)];
52
53      % Go throuh every NUID and assign it to data_ways_selected
54      % where the node ID and the voltage level matches
55      for i_nuid = 1 : size(nodes_unique, 1)
56
57          % Get the orignal node ID of current NUID
58          node_org_ID = cell2mat(nodes_conversion(i_nuid, 2));
59
60          % Get the voltage level of current NUID
61          node_org_voltage = cell2mat(nodes_conversion(i_nuid, 3));
62
63          % Create a boolean index which node1 has exactly that org_ID
64          b_node1_ID_match = node1_data(:, 1) == node_org_ID;
65          b_node2_ID_match = node2_data(:, 1) == node_org_ID;
66
```

```
67          % Create a boolean index which voltage matches current NUID voltage
68          b_node1_voltage_match = node1_data(:, 2) == node_org_voltage;
69          b_node2_voltage_match = node2_data(:, 2) == node_org_voltage;
70
71
72          % Create a boolean index when both conditions are met
73          b_node1_id_and_voltage_ok = b_node1_ID_match & b_node1_voltage_match;
74          b_node2_id_and_voltage_ok = b_node2_ID_match & b_node2_voltage_match;
75
76          % assign every node which satifies both conditions current NUID
77          [data(b_node1_id_and_voltage_ok).node1_nuid] = deal(nuid(i_nuid));
78          [data(b_node2_id_and_voltage_ok).node2_nuid] = deal(nuid(i_nuid));
79      end
80
81      %%% Create strings for the Annotation "Anmerkung" column
82      % Prealloacte Annotations string
83      str_annotation = cell(num_of_ways, 1);
84
85      % go through all ways
86      for i_ways = 1 : num_of_ways
87
88          % Create string if current way has multiple voltage levels
89          if data(i_ways).vlevels ~= 1
90              str_annotation{i_ways, 1} = strcat(str_annotation{i_ways, 1}, ...
91                                                 ", multiple vlevels");
92          end
93
94          % Create string if current way is doubled/tripled/quadrupelt
95          if data(i_ways).systems == 2
96              str_annotation{i_ways, 1} = strcat(str_annotation{i_ways, 1}, ...
97                                                 ", 6 cables - 2 systems");
98
99          elseif data(i_ways).systems == 3
100             str_annotation{i_ways, 1} = strcat(str_annotation{i_ways, 1}, ...
101                                                ", 9 cables - 3 systems");
102
103         elseif data(i_ways).systems == 4
104             str_annotation{i_ways, 1} = strcat(str_annotation{i_ways, 1}, ...
105                                                ", 12 cables - 4 systems");
106         end
107
108         % Create string if current way is DC candidate
109         if data(i_ways).dc_candidate
110             str_annotation{i_ways, 1} = strcat(str_annotation{i_ways, 1}, ...
111                                                ", potentially DC");
112         end
113
114         % Add a blackspace if no annotation was made
115         if isempty(str_annotation{i_ways, 1})
116            str_annotation{i_ways, 1} = " ";
117         end
118     end
```

```
119     % Create column 'Anmerkung'
120     UID = [data(:).UID]';
121     Anmerkung = strcat(repmat("UID: ", num_of_ways, 1), ...
122                        num2str(UID, '%04.f'), string(str_annotation(:)));
123     %%% Get all the other variables needed to export "Stamm Leitungen"
124     % Get the "vonKnoten" and "nachKnoten" NUIDs
125     vonKnoten = [data(:).node1_nuid]';
126     nachKnoten = [data(:).node2_nuid]';
127
128     % Create column 'SpgsebeneWert'
129     SpgsebeneWert = [data(:).voltage]' / 1000;
130
131     % Create column 'LtgLaenge'
132     LtgLaenge = round([data(:).length]', 2);
133
134     % Create column 'LtgsID'
135     LtgsID = [data(:).LtgsID]';
136
137     % Create column 'Land'
138     Land = repmat(export_excel_country_code, num_of_ways, 1);
139
140     % Create column 'Inbetriebnahmejahr'
141     Inbetriebnahmejahr = 2000 * ones(num_of_ways, 1);
142
143     % Create all 0-entry columns for "Stamm_Leitungen"
144     [R, XL, XC, Itherm,  LetztesJahr, SpgsebeneTechnisch, Leistung, ...
145     PhiPsMax, TYNDP2010_Projektnummer, TYNDP2010_Projektstatus, ...
146     Itherm_NDJF, Itherm_MA, Itherm_MJJA, Itherm_SO] ...
147         = deal(zeros(num_of_ways, 1));
148
149
150     %%% Export "Stamm_Leitungen" to Excel
151     % Create Timestamp string
152     str_timestamp = datestr(now, 'yyyy-mm-dd_HH-MM-SS');
153
154     % Create Countrycode string
155     str_cc = [char(export_excel_country_code), '_'];
156
157     % Create table for "Stamm_Leitungen"
158     table_leitungen = table(LtgsID, Land, vonKnoten, nachKnoten,  ...
159                             SpgsebeneWert, R, XL, XC, Itherm, ...
160                             LtgLaenge, Inbetriebnahmejahr, LetztesJahr, ...
161                             SpgsebeneTechnisch, Leistung, Anmerkung, ...
162                             PhiPsMax, TYNDP2010_Projektnummer, ...
163                             TYNDP2010_Projektstatus, Itherm_NDJF, ...
164                             Itherm_MA, Itherm_MJJA, Itherm_SO);
165
166     % Write that table to a Excel file
167     writetable(table_leitungen, ...
168               ['tbl_Stamm_Leitungen_', str_cc, str_timestamp, '.xlsx'], ...
169                'Sheet', 1)
170
```

```
171     % Write all tags on sheet 2
172     writetable(struct2table(data_tags), ...
173               ['tbl_Stamm_Leitungen_', str_cc, str_timestamp, '.xlsx'], ...
174                'Sheet', 2)
175     fprintf(['   INFO: In "Stamm_Leitungen.xlsx" in  "Sheet 2" all tags ' ...
176              'from all UIDs are listed! \n' ...
177              '         Have a look for data inspection! \n'])
178
179
180     %%% Get all the other variables needed for export "Stamm Knoten"
181     % Create column 'KnotenID'
182     KnotenID = nuid;
183
184     % Create column 'Land'
185     Land = repmat(export_excel_country_code, num_of_unique_nodes, 1);
186
187     % Create column 'SpgsebeneWert'
188     SpgsebeneWert = cell2mat(nodes_conversion(:, 3)) / 1000;
189
190     % Create column 'lon' and 'lat'
191     lon = cell2mat(nodes_conversion(:, 4));
192     lat = cell2mat(nodes_conversion(:, 5));
193
194     % Create column 'Inbetriebnahmejahr'
195     Inbetriebnahmejahr = 2000 * ones(num_of_unique_nodes, 1);
196
197     % Create all 0-entry columns for "Stamm_Knoten"
198     [Knotenname, Bevoelkerung, Anmerkung, Gewicht2, Nuts] ...
199         = deal(zeros(num_of_unique_nodes, 1));
200
201
202     %%% Export "Stamm_Knoten" to Excel
203     % Create table for "Stamm_Knoten"
204     table_knoten = table(KnotenID, Land, Knotenname, SpgsebeneWert, lat, ...
205                          lon, Bevoelkerung, Anmerkung, Gewicht2, ...
206                          Inbetriebnahmejahr, Nuts);
207
208     % Write that table to a Excel file
209     writetable(table_knoten, ['tbl_Stamm_Knoten_', str_cc, ...
210                               str_timestamp, '.xlsx'])
211
212     fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
213 end
```

Matlab Code 8.4: Funktion *my_export_excel()*

Da diese Funktion die längste des gesamten Programms ist, ist sie zur besseren Übersicht in 6 Teilbereiche unterteilt:

1. Erstellen und Zuweisen der *NUID*
2. Erstellen der Spalte »Anmerkung«
3. Erstellen der restlichen Spalten für »Stamm_Leitungen«
4. Exportieren der *Excel*-Datei »Stamm_Leitungen«
5. Erstellen der restlichen Spalten für »Stamm_Knoten«
6. Exportieren der *Excel*-Datei »Stamm_Knoten«

8.4.1 Erstellen und Zuweisen der NUID

Im ersten Teilbereich wird die *NUID* erstellt und zugewiesen. Die *NUID* ist in Anlehnung an die *UID* die eindeutige Identifikationsnummer für Endknoten, also Node-Elemente.

Zuerst werden dazu sämtliche notwendigen Informationen (*OpenStreetMap* Node-ID, Spannungsebene, Längen- und Breitengrad) vom eigentlichem Datensatz extrahiert und mittels der MATLAB Funktion *unique()* auf Einzigartigkeit geprüft.

Dies bedeutet auch, dass wenn z.B. in einem gruppierten Endpunkt eines Umspannwerkes Leitungen unterschiedlicher Spannungsebenen enden, für jede Spannungsebene ein einzigartiger Endknoten erstellt wird.

Dies ist für *ATLANTIS* notwendig, um so in sich vollständige Spannungsebenennetze zu haben.

Die *NUID* besteht, ähnlich wie die *LtgsID*, aus mehreren Bestandteilen:

1. Den zwei Buchstaben des Länderkürzels und
2. einer fortlaufenden fünfstelligen Zahl mit führenden Nullen.

Das Länderkürzel wird am Anfang des Programms in der Variable *export_excel_country_code* (Kapitel 3.2) festgelegt.

Die nach diesem Schema erstellen *NUIDs* werden in der lokalen Variable *nodes_converison* den einzigartigen Knoten zugewiesen.

Jedem Way-Element werden nun die jeweiligen *NUIDs* zugewiesen, sofern die originale Node-ID aus *OpenStreetMap* als auch die Spannungsebene übereinstimmen.

8.4.2 Erstellen der Spalte »Anmerkung«

Im zweiten Teilbereich werden das *string array* für die Spalte »Anmerkung« erstellt. Diese Anmerkungen enthalten wichtige Informationen:

- Die UID des Way-Elements,
- eine Information, ob diese Leitung mehrere Spannungsebenen aufweist (optional),
- eine Information, ob diese Leitung auf Grund der Leiterseilanzahl vervielfacht wurde (optional) und
- eine Information, ob es sich bei dieser Leitung um eine Gleichspannungsleitung handeln könnte (optional).

Hat die Leitung mehrere Spannungsebenen, so wurde sie bereits automatisch verdoppelt oder verdreifacht. Dies wurde in Abschnitt 5.1 beschrieben.

In diesem Fall ist manuell die Anzahl der Leiterseile zu aktualisieren. Dies wird nicht automatisch gemacht, da es zu viele verschiedene Möglichkeiten der Aufteilung der Leiterseile auf die jeweilige Spannungsebene gibt, als dass ein verlässliches, fehlerfreies Verhalten erwarten werden kann.

Hat die Leitung 6, 9 oder 12 Leiterseile, so wurde diese bereits verdoppelt, verdreifacht oder vervierfacht. Der Prozess der Vervielfachung wird in Abschnitt 6.3 und 8.3 beschrieben.

Sollten beide Fälle, mehrere Spannungsebenen und bereits erfolgte Vervielfachung der Leitung auf Grund der Anzahl der Leiterseile, zugetroffen sein, muss sowieso diese Leitung manuell kontrolliert werden und ggf. angepasst und einzelne Duplikate entfernt werden.

Im vierten Fall wird eine Anmerkung hinzugefügt, sollte die Leitung eines der Kriterien beschrieben in Abschnitt 6.2 erfüllen und möglicherweise eine Gleichspannungsleitung sein. In diesem Fall empfiehlt es sich, auf der zweiten Seite der *Excel*-Datei die restlichen Informationen des *fields* »tags« zu kontrollieren.

8.4.3 Erstellen der Spalten für »Stamm_Leitungen«

Im dritten Teilbereich werden die restlichen Spalten erstellt, welche in der *Excel*-Datei »Stamm_Leitungen« benötigt werden.

Die beiden Spalten *vonKnoten* und *nachKnoten* enthalten jeweils die Werte des *endnode1* und *endnode2*. Es wird keinen Wert auf die Richtung des tatsächlichen Leistungsflusses gelegt, die Zuordnung »von« und »nach« ist somit willkürlich.

Die Spalte *SpgsebeneWert* enthält Information über die aktuelle Spannungsebene in kV.

In der Spalte *LtgLaenge* befindet sich die auf zwei Dezimalstellen gerundete Leitungslänge in km.

Die Erstellung der *LtgsID* wurde bereits in Abschnitt 8.3 beschrieben.

In der Spalte *Land* wird das Länderkürzel gespeichert, der Wert wird aus der Variable *export_excel_country_code* entnommen.

Da es sich bei den Leitungen im *OpenStreetMap* um existierende Leitungen handelt und keine geplanten *Projektleitungen* enthalten sind, wird für das *Inbetriebnahmejahr* stets 2000 gesetzt.

Sollte das Startjahr der Simulation nach 2000 sein, so muss manuell kontrolliert werden, ob Leitungen zwischen dem Jahr 2000 und dem Startjahr in Betrieb genommen wurden und das entsprechende Inbetriebnahmejahr gesetzt werden.

Folgende Elemente werden lediglich mit dem Wert »0« aufgefüllt:

- *R*,
- *XL*,
- *XC*,
- *Itherm*,
- *LetztesJahr*,
- *SpgsebeneTechnisch*,
- *Leistung*,
- *PhiPsMax*,
- *TYNDP2010_Projektnummer*,
- *TYNDP2010_Projektstatus*,
- *Itherm_NDJF*,
- *Itherm_MA*,
- *Itherm_MJJA* und
- *Itherm_SO*

8.4.4 Exportieren der Excel-Datei »Stamm_Leitungen«

Die Daten aller Leitungen werden im vierten Teilbereich als *Excel*-Datei exportiert.

Auf dem ersten Blatt der *Excel*-Datei befindet sich der Datensatz der Variable *data_ways_selected* und die in Abschnitt 8.4.3 beschriebenen Spalten.

Auf dem zweiten Blatt der *Excel*-Datei befinden sich die von *my_get_tags()* in Abschnitt 8.2 beschriebenen »tags« des Datensatzes aus *data_ways_selected*.

Der Dateiname wird aus folgenden Teilen zusammengesetzt:

- Dem String *tbl_Stamm_Leitungen*,
- dem aktuellen Zeitstempel im Format *yyyy-mm-dd_HH-MM-SS*,
- dem Länderkürzel aus der Variable *export_excel_country_code* und
- der Dateiendung *.xlsx*.

8.4.5 Erstellen der Spalten für »Stamm_Knoten«

Im fünften Teilbereich werden die restlichen Spalten erstellt, welche in der *Excel*-Datei »Stamm_Knoten« benötigt werden.

Die Spalte *KnotenID* enthält die *NUID*, wie sie in Abschnitt 8.4.1 beschrieben wurde.

In der Spalte *Land* wird das Länderkürzel gespeichert, der Wert wird aus der Variable *export_excel_country_code* entnommen.

Die Spalte *SpgsebeneWert* enthält Information über die aktuelle Spannungsebene in kV.

Die beiden Spalten *lon* und *lat* enthalten die Breiten- und Längengradkoordinaten der jeweiligen *NUID* aus Abschnitt 8.4.1.

Das *Inbetriebnahmejahr* ist stets das Jahr 2000, diese Spalte wird somit vorausgefüllt.

Folgende Elemente werden lediglich mit dem Wert »0« aufgefüllt:

- *Knotenname*,
- *Bevoelkerung*,
- *Anmerkung*,
- *Gewicht2* und
- *Nuts*.

8.4.6 Exportieren der Excel-Datei »Stamm_Knoten«

Die Daten aller Knoten werden im fünften und letzten Teilbereich als *Excel*-Datei exportiert.

Auf dem ersten Blatt der *Excel*-Datei befindet sich die Daten sämtlicher in Abschnitt 8.4.1 beschriebener Endknoten und die in Abschnitt 8.4.5 beschriebenen Spalten.

Der Dateiname wird aus folgenden Teilen zusammengesetzt:

- Dem String *tbl_Stamm_Knoten*,
- dem aktuellen Zeitstempel im Format *yyyy-mm-dd_HH-MM-SS*,
- dem Länderkürzel aus der Variable *export_excel_country_code* und
- der Dateiendung *.xlsx*.

9 Funktionen - Modul Visualisierung

Das letzte Programmmodul ist das Modul *Visualisierung*. In diesem wird der Datensatz in *data_ways_selected* optional in drei verschiedenen Stadien visuell dargestellt:

1. Im Originalzustand, so wie die Daten importiert wurden,
2. während der Gruppierung der Endpunkte und
3. im Finalzustand, so wie die Daten exportiert werden.

Der Aufbau des *Visualisierung Moduls* ist schematisch in Abb. 9.1 dargestellt.

Das Modul *Visualisierung* besteht aus drei Funktionen: .

1. *my_plot_ways_original()*, beschrieben in Abschnitt 9.1,
2. *my_plot_ways_grouping()*, beschrieben in Abschnitt 9.2 und
3. *my_plot_ways_final()*, beschrieben in Abschnitt 9.3.

Abbildung 9.1: Schematischer Aufbau Modul 6 von 6

9.1 my_plot_ways_original()

Die erste Funktion, *my_plot_ways_original()*, stellt die Daten so dar, wie sie von *OpenStreetMap* exportiert und nach *Modul 2 - Spannungsebenenauswahl*, beschrieben in Kapitel 5, nach Spannungsebenen selektiert wurden.

```
1  function my_plot_ways_original(data, data_busbars, ...
2                                 voltage_levels_selected, bool)
3
4      % DESCRIPTION
5      % This function plots the original dataset as it was. Two plots will
6      % be genereted if the flag in "bool" was set: A plot with a lon/lat
7      % coordinate system and a plot with a inaccuarte, but more intuitive
8      % x/y plot in km. Since Matlab is a bit tricky with legends and color
9      % coding of same plots, a workaround with pseudo-points is necessary.
10     % There are a total of 12 different colors which are easy
11     % distinguishable. If more than 12 voltage levels will be selected,
12     % colors will repeat.
13     %
14     % INPUT
15     % data ... dataset with data to plot
16     % data_busbars ... the busbars which have been deleted from data
17     % voltage_levels_selected ... list of selected voltage levels to
18     %                             determine color
19     % bool ... boolean operator to toggle visualisations on/off
20     %
21     % OUTPUT
22     % (none)
23
24     if bool.plot_ways_original
25         tic
26         disp('Start ploting original ways... (takes a few seconds)')
27
28         % Create custom 12 color qualitative Colormap for better distinctness
29         % Credits: Colormap based on "paired", by www.ColorBrewer.org
30         colormap = [ 51,160, 44;  31,120,180; 177, 89, 40; 106, 61,154;
31                     255,127,  0; 178,223,138; 227, 26, 28; 255,255,153;
32                     166,206,227; 202,178,214; 251,154,153; 253,191,111;] / 255;
33
34         % Create a warning if colors of voltage levels do repeat
35         if numel(voltage_levels_selected) > 12
36             fprintf(['   ATTENTION!  More than 12 voltage levels ' ...
37                                      'are selected.\n' ...
38                      '               Colors of voltage lines do repeat now!'...
39                      '\n               It is recommended ' ...
40                      'to select max. 12 voltage levels.\n'])
41         end
```

```
42          % Create figure for deg Plot
43          figure
44          hold on
45          grid on
46          title('Original ways, only selected voltages, lon/lat coordinates')
47          xlabel('Longitude [°]'), ylabel('Latitude [°]')
48
49
50          % Working around a Matlab Bug: To create concurrent coloring
51          % and labeling, pseudo-points at the origin have to be plotted first
52          % in the correct color order, then the legend can be created,
53          % then the pseudo points will be overwritten with white color and
54          % finally the real data can be plotted.
55
56          % Calculate midpoint to place the pseudo-points
57          lat_mean = mean([[data.lat1], [data.lat2]]);
58          lon_mean = mean([[data.lon1], [data.lon2]]);
59
60          for i_vlevel = numel(voltage_levels_selected) : -1 : 1
61
62              % Cycle through the indices of 1:12 even if it exceeds 12
63              % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
64              i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
65
66              % Pick for each voltage level corresponding color
67              current_color = colormap(i_colormap, :);
68
69              % Plot pseudo-points at the origin in correct color order
70              plot(lon_mean, lat_mean, 'o-' , 'Color', current_color)
71          end
72
73          % create legend labels
74          labels = [num2str(flipud(voltage_levels_selected) / 1000), ...
75                    repmat(' kV', numel(voltage_levels_selected), 1)];
76
77          % Create legend in correct color order
78          legend(labels, 'Location', 'northeastoutside', 'AutoUpdate', 'off')
79
80          % Set the pseudo-points invisible by overriding with a white point
81          plot(lon_mean, lat_mean, 'o-' , 'Color', [1 1 1])
82
83          % get all coordinates of all busbars
84          busbars_lon = [data_busbars.lon1; data_busbars.lon2];
85          busbars_lat = [data_busbars.lat1; data_busbars.lat2];
86
87          % Plot all busbars of current_voltage with black "x" and crossed lines
88          plot(busbars_lon, busbars_lat, 'kx-', 'LineWidth', 1)
89
90
91          % Now plot the real data in correct color order, with highest vlevel
92          % on top of the other voltage levels (therefore reverse for-loop)
93
```

```
94          for i_vlevel = numel(voltage_levels_selected) : -1 : 1
95              % Cycle through the indices of 1:12 even if it exceeds 12
96              % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
97              i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
98
99              % Pick for each voltage level a color
100             current_color = colormap(i_colormap, :);
101
102             % current voltage level in this loop:
103             current_voltage = voltage_levels_selected(i_vlevel);
104
105             % create boolean index with all wayelement in current voltage level
106             b_current_voltage = [data.voltage] == current_voltage;
107
108             % get all ways with the current voltage level
109             current_ways = data(b_current_voltage);
110
111             % get all coordinates of current ways
112             busbars_lon = [current_ways.lon1; current_ways.lon2];
113             busbars_lat = [current_ways.lat1; current_ways.lat2];
114
115             % Plot all ways of current_voltage in corresponding color
116             plot(busbars_lon, busbars_lat, '-o', 'Color', current_color)
117         end
118
119
120         % Create figure for X/Y km Plot
121         figure
122         hold on
123         grid on
124         title('Original ways, only selected voltages, x/y coordinates')
125         xlabel('x - distance from midpoint [km]')
126         ylabel('y - distance from midpoint [km]')
127
128
129         % Working around a Matlab Bug: To create concurrent coloring
130         % and labeling, pseudo-points at the origin have to be plotted first
131         % in the correct color order, then the legend can be created,
132         % then the pseudo points will be overwritten with white color and
133         % finally the real data can be plotted.
134         for i_vlevel = numel(voltage_levels_selected) : -1 : 1
135
136             % Cycle through the indices of 1:12 even if it exceeds 12
137             % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
138             i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
139
140             % Pick for each voltage level corresponding color
141             current_color = colormap(i_colormap, :);
142
143             % Plot pseudo-points at the origin in correct color order
144             plot([0, 0], [0, 0], 'o-' , 'Color', current_color)
145         end
```

```
146         % create legend labels
147         labels = [num2str(flipud(voltage_levels_selected) / 1000), ...
148                   repmat(' kV', numel(voltage_levels_selected), 1)];
149 
150         % Create legend in correct color order
151         legend(labels, 'Location', 'northeastoutside', 'AutoUpdate', 'off')
152 
153         % Set the pseudo-points invisible by overriding with a white point
154         plot([0, 0], [0, 0], 'o-' , 'Color', [1 1 1])
155 
156         % get all coordinates of all busbars
157         busbars_x = [data_busbars.x1; data_busbars.x2];
158         busbars_y = [data_busbars.y1; data_busbars.y2];
159 
160         % Plot all ways of current_voltage in corresponding color
161         plot(busbars_x, busbars_y, 'kx-', 'LineWidth', 1)
162 
163         % Now plot the real data in correct color order, with highest vlevel
164         % on top of the other voltage levels (therefore reverse for-loop)
165         for i_vlevel = numel(voltage_levels_selected) : -1 : 1
166 
167             % Cycle through the indices of 1:12 even if it exceeds 12
168             % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
169             i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
170 
171             % Pick for each voltage level a color
172             current_color = colormap(i_colormap, :);
173 
174             % current voltage   level in this loop:
175             current_voltage = voltage_levels_selected(i_vlevel);
176 
177             % create boolean index with all wayelement in current voltage level
178             b_current_voltage = [data.voltage] == current_voltage;
179 
180             % get all ways with the current voltage level
181             current_ways = data(b_current_voltage);
182 
183             % get all coordinates of current ways
184             x = [current_ways.x1; current_ways.x2];
185             y = [current_ways.y1; current_ways.y2];
186 
187             % Plot all ways of current_voltage in corresponding color
188             plot(x, y, '-o', 'Color', current_color)
189         end
190 
191         fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
192     end
193 end
```

Matlab Code 9.1: Funktion *my_plot_ways_original()*

Die Funktion generiert zwei Plots, einen mit Längen-/Breitengradkoordinaten und einen mit *x/y*-Koordinaten. In Abb. 9.2 ist ein Umspannwerk dargestellt.

Dem Datensatz wurden aus Performancegründen bereits singuläre Way-Elemente (siehe Abschnitt 8.1) entfernt, die Busbars und Bays wurden jedoch zur Darstellung in der Variable *data_busbars* gespeichert.

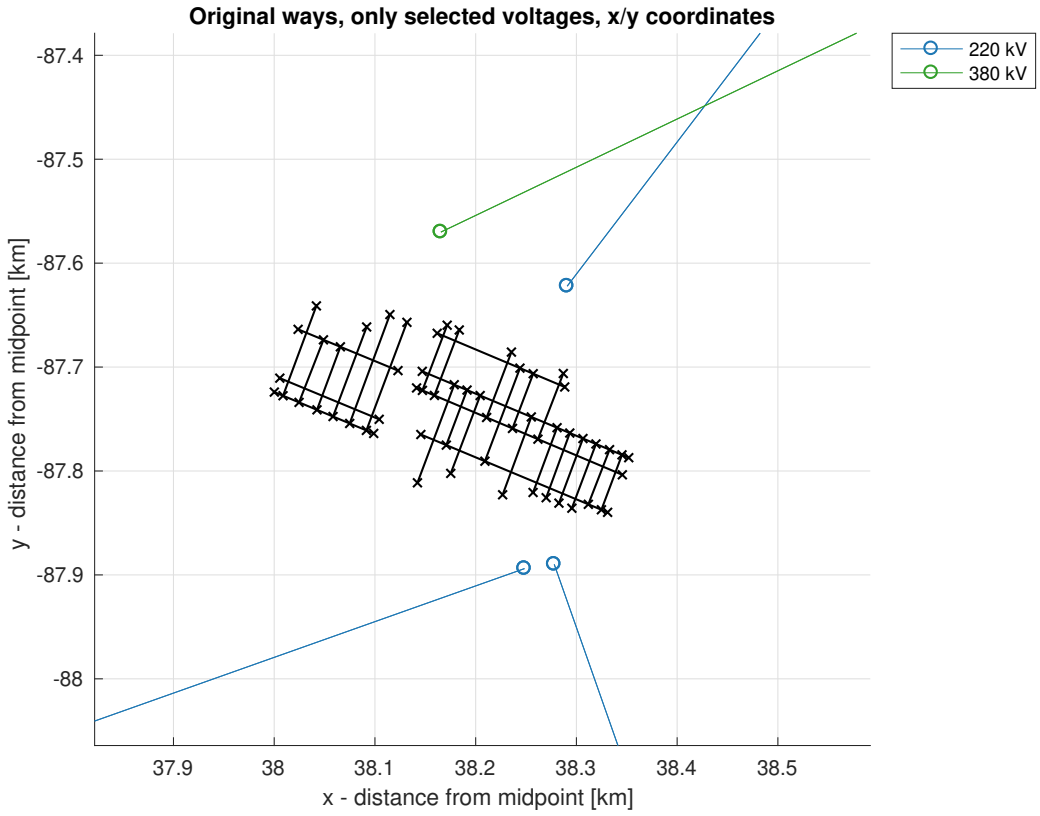


Abbildung 9.2: Visualisierung von Funktion *my_plot_ways_original()*

Durch Erstellen beider Visualisierungen können interessante Stellen gefunden und so manuell zwischen Längen-/Breitengradkoordinaten und *x/y*-Koordinaten verglichen werden.

Die Way-Elemente werden farblich nach Spannungsebenen sortiert dargestellt, beginnend mit der niedrigsten Spannungsebene zuerst und die höchste Spannungsebene über allen anderen. Es steht eine Farbpalette von 12 gut zu unterscheidenden Farben bereit [9].

Falls mehr als 12 Spannungsebenen dargestellt werden, wiederholt sich die Farbpalette und ein entsprechender Warnhinweis wird in der Konsole ausgegeben.

Die darzustellenden Spannungsebenen werden in der Variable *voltage_levels_selected* übergeben.

Way-Elemente, welche als *Busbars* oder *Bays* erkannt wurden, werden in schwarzer Farbe mit »x« statt »o« als Endpunkten dargestellt. Diese werden in der Variable *data_busbars* übergeben. Dadurch ist schnell zu erkennen, welche Elemente in der Funktion *my_delete_busbars()*, beschrieben in Abschnitt 6.1, automatisch entfernt werden.

Da es in MATLAB nicht einfach möglich ist, viele Elemente in einer bestimmten Farbe so zu plotten und dennoch nur einen Legendeneintrag je Spannungsebene darzustellen, werden die Way-Elemente in einem etwas kompliziert anmutenden Verfahren dargestellt:

Zuerst wird für jede Spannungsebene im Ursprung ein Punkt in der jeweiligen Farbe erstellt. Danach wird die Legende erstellt und die Punkte wieder gelöscht, sprich von einem weißen Punkt überdeckt. Erst danach werden die eigentlichen Way-Elemente dargestellt, ohne die bereits erstellte Legende zu beeinflussen.

Ob dieser Plot überhaupt erstellt wird, wird in der Variable *bool*, beschrieben in Kapitel 3.2, festgelegt.

9.2 my_plot_ways_grouping()

In der zweiten Visualisierungsfunktion, *my_plot_ways_grouping()*, wird der Übergang zwischen den Visualisierungen in *my_plot_ways_original()* und *my_plot_ways_final()* dargestellt.

```
function my_plot_ways_grouping(data, data_busbars, grouped_xy_coordinates, ...
                               neighbourhood_threshold, bool)
    % DESCRIPTION
    % This function will plot the transition while grouping endnodes. In
    % grey with dotted lines the orignal dataset will be plotted, all
    % endnodes which will be grouped together, so which are stacked or in a
    % neighourhood, will be plotted in a different color (be aware that by
    % accident neighouring neighbourhood-groups can occasionally have the
    % same colors!). Over all grouped endnodes a circle with the threshhold
    % radius will be plotted, this is very helpful to determine the correct
    % value for the treshold. If the plot reveals that obviously
    % neighbouring groups wont be grouped correctly, it is useful to
    % increase the threshold radius, the opposite is true if endnodes,
    % which should not be grouped together, will be grouped.
    %
    % INPUT
    % data ... dataset with data to plot
    % data_busbars ... the busbars which have been deleted from data
    % grouped_xy_coordinates ... all x/y coordinates of a group
    % neighbourhood_threshold ... the radius of grouping
    % bool ... boolean operator to toggle visualisations on/off
    %
    % OUTPUT
    % (none)

    if bool.plot_ways_grouping

        disp('Start plotting all grouped endnodes... (takes a few seconds)')

        % Start figure
        figure
        hold on
        grid on
        title('Original and final ways with grouping-circles')
        xlabel('x - distance from midpoint [km]')
        ylabel('y - distance from midpoint [km]')

        % Plot all ways dashed with all original endnodes in light grey
        x = [[data.x1]; [data.x2]];
        y = [[data.y1]; [data.y2]];
        plot(x, y, 'o--k', 'Color', [0.6 0.6 0.6])
```

```
42          % get all coordinates of all busbars
43          busbars_lon = [data_busbars.lon1; data_busbars.lon2];
44          busbars_lat = [data_busbars.lat1; data_busbars.lat2];
45
46          % Plot all busbars of current_voltage with black "x" and crossed lines
47          plot(busbars_lon, busbars_lat, 'o--', 'Color', [0.6 0.6 0.6])
48
49          % Plot circles around each grouped endpoint
50          origin_circles = reshape(nonzeros(grouped_xy_coordinates'), 2, [])';
51          radii = neighbourhood_threshold * ones(size(origin_circles, 1), 1);
52          viscircles(origin_circles, radii, 'LineWidth', 1, 'LineStyle', ':');
53
54          % Plot the new ways
55          plot([[data.x1_final]; [data.x2_final]], ...
56               [[data.y1_final]; [data.y2_final]], 'k-o')
57
58          % Plot all new grouped endpoints in pink
59          x_grouped = [[data.x1_grouped],[data.x2_grouped]];
60          y_grouped = [[data.y1_grouped],[data.y2_grouped]];
61          plot(x_grouped, y_grouped, '.m', 'Markersize', 15)
62
63          % Plot all groups of combined endpoints in a different color
64          for i_group = 1 : size(grouped_xy_coordinates, 1)
65              group_xy = nonzeros(grouped_xy_coordinates(i_group, :));
66              plot(group_xy(1:2:end), group_xy(2:2:end), '*')
67          end
68
69          fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
70      end
71  end
```

Matlab Code 9.2: Funktion *my_plot_ways_grouping()*

Die Way-Elemente, ausgenommen Busbars und Bays, aus *my_plot_ways_original()* werden in grauer Farbe und strichliert als erstes visualisiert.

Endpunkte, welche nach dem im Kapitel 7 beschriebenen Verfahren gruppiert wurden, werden gruppenweise in unterschiedlichen Farben geplottet.

Da diese Farben zufällig den jeweiligen Gruppen zugeteilt werden, kann es vorkommen, dass nebeneinander liegende Gruppen die selbe Farbe erhalten, obwohl sie gar nicht der gleichen Gruppe angehören.

Der neue, finale Endpunkt wird größer in der Farbe violett dargestellt und befindet sich immer in der Mitte, genau genommen dem Schwerpunkt der Punktfläche, der Gruppe.

Von diesem neuem Mittelpunkt werden mit schwarzen, durchgezogenen Linien die neuen Way-Elemente aus *my_plot_ways_final()* dargestellt.

Die wohl wichtigste Information dieser Visualisierung sind die rot strichlierten Kreise, welche um jeden originalen Endnode einer Gruppe dargestellt werden.

Diese Kreise haben den in der Variable *neighbourhood_treshold* festgelegten Radius und visualisieren den Einzugsradius des *Moduls Gruppierung* aus Kapitel 7.

Nicht jeder Endpunkt in einem Umspannwerk muss von jedem Endpunkt im Radius liegen, es reicht, wenn sich die Kreise überlagern und so die Endpunkte aneinander ketten. Sämtliche aneinander geketteten Endpunkte werden schlussendlich in eine Gruppe zusammengefasst.

Sollte sich bei manueller Kontrolle herausstellen, dass die Kreise zu groß sind und ungewollt andere Endpunkte, z.B. bei parallel verlaufenden Leitungen, miterfassen, so ist der Wert der Variable *neighbourhood_treshold* zu verringern.

Im Gegenzug muss der Wert Variable *neighbourhood_treshold* erhöht werden, wenn nicht alle Endpunkte eines z.B. Umspannwerkes erfasst werden und sich innerhalb eines Umspannwerkes mehrere Untergruppen bilden, die nicht zusammengefasst werden und ggf. mit einem eigenem Way-Element verbunden werden.

Es ist zu beachten, dass durch die Konvertierung in *x/y*-Koordinaten eine naturgemäße Verzerrung stattfindet, da es nicht ohne Kompromisse oder großem Rechenaufwand möglich ist, eine runde dreidimensionale Kugeloberfläche auf eine zweidimensionale ebene Fläche zu projizieren.

Im Ursprung des *x/y*-Koordinatensystem ist dieser Effekt vernachlässigbar, je weiter man sich vom Ursprung entfernt, desto stärker ist die Verzerrung ausgeprägt.

Dies gilt jedoch in erster Linie bei einer maßgeblichen Distanz in Nord-/Süd-Richtung, sprich bei einer großen Änderungen des Breitengrades. Bei einer großen Änderung des Längengrades jedoch annähernd gleichbleibenden Breitengrades tritt dieser Effekt nur in geringen Ausmaßen auf.

Konkret wirkt sich das insofern aus, dass je nach Position auf nördlicher bzw. südlicher Hemisphäre die in MATLAB dargestellte Distanz im *x/y*-Koordinatensystem geringer bzw. größer als die tatsächliche Distanz ist. In Abschnitt 4.3 wird dieses Thema ebenso behandelt.

Aus Komplexitätsgründen wird in dieser Visualisierung auf eine optische Unterscheidung zwischen den Spannungsebenen verzichtet. Diese Information ist in den Visualisierungen von *my_plot_ways_original()* und *my_plot_ways_final()* zu finden.

Diese Visualisierung wird ausschließlich in *x/y*-Koordinaten erstellt, da dies maßgeblich für die Einstellung der Variable *neighbourhood_treshold* ist. Die Einstellung dieser Variable wird in Kapitel 3.2 genauer beschrieben.

Ob dieser Plot überhaupt erstellt wird, wird in der Variable *bool*, beschrieben in Kapitel 3.2, festgelegt. In Abb. 9.3 ist das Umspannwerk aus Abb. 9.2 in der Gruppierungsphase zu sehen.

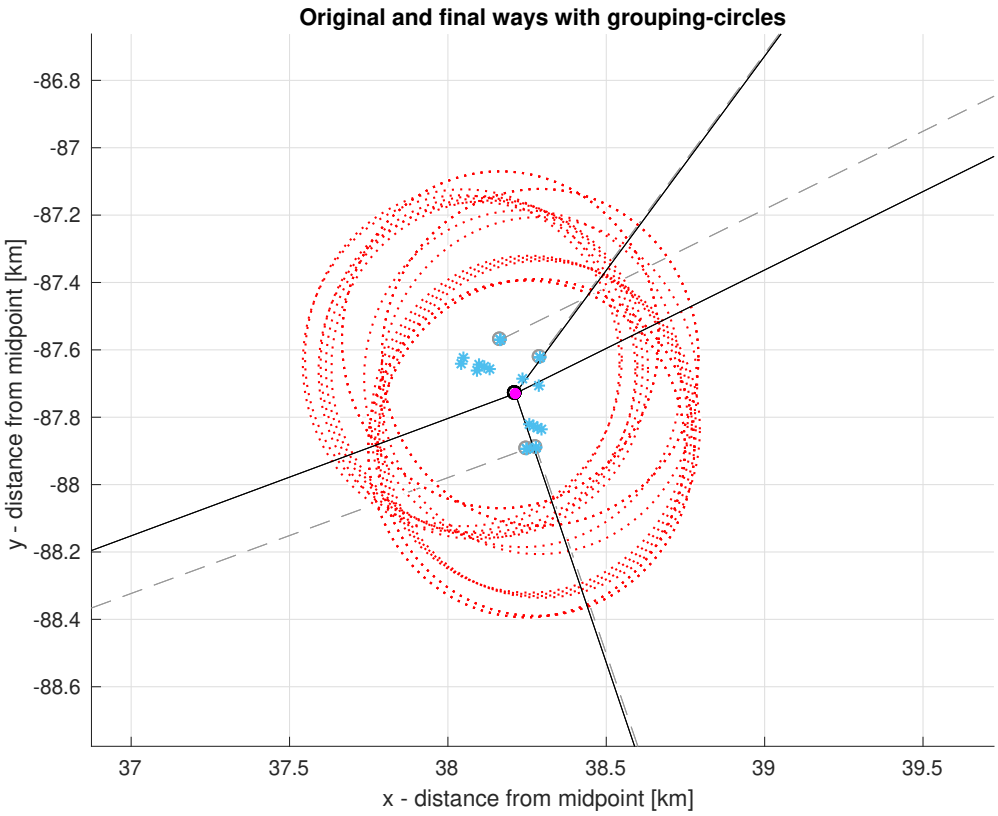


Abbildung 9.3: Visualisierung von Funktion *my_plot_ways_grouping()*

9.3 my_plot_ways_final()

Die dritte und allerletzte Funktion des MATLAB Programms, *my_plot_ways_final()*, stellt die Daten so dar, wie sie von *my_export_excel()*, beschrieben in Abschnitt 8.4 schlussendlich exportiert und in *ATLANTIS* dargestellt werden.

```
function my_plot_ways_final(data, voltage_levels_selected, bool)

    % DESCRIPTION
    % This function plots the final dataset as it will be exported. Two plots
    % will be genereted if the flag in "bool" was set: A plot with a lon/lat
    % coordinate system and a plot with a inaccuarte, but more intuitive
    % x/y plot in km. Since Matlab is a bit tricky with legends and color
    % coding of same plots, a workaround with pseudo-points is necessary.
    % There are a total of 12 different colors which are easy
    % distinguishable. If more than 12 voltage levels will be selected,
    % colors will repeat.
    %
    % INPUT
    % data ... dataset with data to plot
    % voltage_levels_selected ... list of selected voltage levels to
    %                             determine color
    % bool ... boolean operator to toggle visualisations on/off
    %
    % OUTPUT
    % (none)

    if bool.plot_ways_final
        tic
        disp('Start ploting final ways... (takes a few seconds)')

        % Create custom 12 color qualitative Colormap for better distinctness
        % Credits: Colormap based on "paired", by www.ColorBrewer.org
        colormap = [ 51,160, 44;  31,120,180; 177, 89, 40; 106, 61,154;
                    255,127,  0; 178,223,138; 227, 26, 28; 255,255,153;
                    166,206,227; 202,178,214; 251,154,153; 253,191,111;] / 255;

        % Create a warning if colors of voltage levels do repeat
        if numel(voltage_levels_selected) > 12
            fprintf(['   ATTENTION!  More than 12 voltage levels ' ...
                                    'are selected.\n' ...
                     '               Colors of voltage lines do repeat now!'...
                     '\n               It is recommended ' ...
                     'to select max. 12 voltage levels.\n'])
        end
```

```
40          %%% Create figure for degree
41          figure
42          hold on
43          grid on
44          title('Final ways as exported, lon/lat coordinates')
45          xlabel('Longitude [°]'), ylabel('Latitude [°]')
46
47          % Working around a Matlab Bug: To create concurrent coloring
48          % and labeling, pseudo-points at the origin have to be plotted first
49          % in the correct color order, then the legend can be created,
50          % then the pseudo points will be overwritten with white color and
51          % finally the real data can be plotted.
52
53          % Calculate midpoint to place the pseudo-points
54          lat_mean = mean([[data.lat1_final], [data.lat2_final]]);
55          lon_mean = mean([[data.lon1_final], [data.lon2_final]]);
56
57          for i_vlevel = numel(voltage_levels_selected) : -1 : 1
58
59              % Cycle through the indices of 1:12 even if it exceeds 12
60              % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
61              i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
62
63              % Pick for each voltage level corresponding color
64              current_color = colormap(i_colormap, :);
65
66              % Plot pseudo-points at the origin in correct color order
67              plot(lon_mean, lat_mean, 'o-' , 'Color', current_color)
68          end
69
70          % create legend labels
71          labels = [num2str(flipud(voltage_levels_selected) / 1000), ...
72                    repmat(' kV', numel(voltage_levels_selected), 1)];
73
74          % Create legend in correct color order
75          legend(labels, 'Location', 'northeastoutside', 'AutoUpdate', 'off')
76
77          % Set the pseudo-points invisible by overriding with a white point
78          plot(lon_mean, lat_mean, 'o-' , 'Color', [1 1 1])
79
80          % Now plot the real data in correct color order, with highest vlevel
81          % on top of the other voltage levels (therefore reverse for-loop)
82          for i_vlevel = numel(voltage_levels_selected) : -1 : 1
83
84              % Cycle through the indices of 1:12 even if it exceeds 12
85              % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
86              i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
87
88              % Pick for each voltage level a color
89              current_color = colormap(i_colormap, :);
90
91
```

```
92              % current voltage level in this loop:
93              current_voltage = voltage_levels_selected(i_vlevel);
94
95              % create boolean index with all wayelement in current voltage level
96              b_current_voltage = [data.voltage] == current_voltage;
97
98              % get all ways with the current voltage level
99              current_ways = data(b_current_voltage);
100
101             % get all coordinates of current ways
102             lon = [current_ways.lon1_final; current_ways.lon2_final];
103             lat = [current_ways.lat1_final; current_ways.lat2_final];
104
105             % Plot all ways of current_voltage in corresponding color
106             plot(lon, lat, '-o', 'Color', current_color)
107         end
108
109
110         %%% Create figure for X/Y km
111         figure
112         hold on
113         grid on
114         title('Final ways as exported, x/y coordinates')
115         xlabel('x - distance from midpoint [km]')
116         ylabel('y - distance from midpoint [km]')
117
118
119         % Working around a Matlab Bug: To create concurrent coloring
120         % and labeling, pseudo-points at the origin have to be plotted first
121         % in the correct color order, then the legend can be created,
122         % then the pseudo points will be overwritten with white color and
123         % finally the real data can be plotted.
124         for i_vlevel = numel(voltage_levels_selected) : -1 : 1
125
126             % Cycle through the indices of 1:12 even if it exceeds 12
127             % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
128             i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
129
130             % Pick for each voltage level corresponding color
131             current_color = colormap(i_colormap, :);
132
133             % Plot pseudo-points at the origin in correct color order
134             plot([0, 0], [0, 0], 'o-' , 'Color', current_color)
135         end
136
137         % create legend labels
138         labels = [num2str(flipud(voltage_levels_selected) / 1000), ...
139                   repmat(' kV', numel(voltage_levels_selected), 1)];
140
141         % Create legend in correct color order
142         legend(labels, 'Location', 'northeastoutside', 'AutoUpdate', 'off')
143
```

```
144         % Set the pseudo-points invisible by overriding with a white point
145         plot([0, 0], [0, 0], 'o-' , 'Color', [1 1 1])
146
147
148         % Now plot the real data in correct color order, with highest vlevel
149         % on top of the other voltage levels (therefore reverse for-loop)
150         for i_vlevel = numel(voltage_levels_selected) : -1 : 1
151
152             % Cycle through the indices of 1:12 even if it exceeds 12
153             % e.g.: 1:12 maps to 1:12, 13:24 maps to 1:12 too, etc.
154             i_colormap = i_vlevel - floor((i_vlevel-1)/12)*12;
155
156             % Pick for each voltage level a color
157             current_color = colormap(i_colormap, :);
158
159             % current voltage level in this loop:
160             current_voltage = voltage_levels_selected(i_vlevel);
161
162             % create boolean index with all wayelement in current voltage level
163             b_current_voltage = [data.voltage] == current_voltage;
164
165             % get all ways with the current voltage level
166             current_ways = data(b_current_voltage);
167
168             % get all coordinates of current ways
169             x = [current_ways.x1_final; current_ways.x2_final];
170             y = [current_ways.y1_final; current_ways.y2_final];
171
172             % Plot all ways of current_voltage in corresponding color
173             plot(x, y, '-o', 'Color', current_color)
174         end
175
176         fprintf('   ... finished! (%5.3f seconds) \n \n', toc)
177     end
178 end
```

Matlab Code 9.3: Funktion *my_plot_ways_final()*

Die Funktion generiert zwei Plots, einen mit Längen-/Breitengradkoordinaten und einen mit *x/y*-Koordinaten.

Durch Erstellen beider Visualisierungen können interessante Stellen gefunden und so manuell zwischen Längen-/Breitengradkoordinaten und *x/y*-Koordinaten verglichen werden.

Die Way-Elemente werden farblich nach Spannungsebenen sortiert dargestellt, beginnend mit der niedrigsten Spannungsebene zuerst und die höchste Spannungsebene über anderen. Es steht eine Farbpalette von 12 gut zu unterscheidenden Farben bereit.

Falls mehr als 12 Spannungsebenen dargestellt werden, wiederholt sich die Farbpalette und ein entsprechender Warnhinweis wird in der Konsole ausgegeben.

Da es in MATLAB nicht einfach möglich ist, viele Elemente in einer bestimmten Farbe so zu plotten und dennoch nur einen Legendeneintrag je Spannungsebene darzustellen, werden die Way-Elemente in einem etwas kompliziert anmutenden Verfahren dargestellt:

Zuerst wird für jede Spannungsebene im Ursprung ein Punkt in der jeweiligen Farbe erstellt. Danach wird die Legende erstellt und die Punkte wieder gelöscht, sprich von einem weißen Punkt überdeckt. Erst danach werden die eigentlichen Way-Elemente dargestellt, ohne die bereits erstellte Legende zu beeinflussen.

Ob dieser Plot überhaupt erstellt wird, wird in der Variable *bool*, beschrieben in Kapitel 3.2, festgelegt. In Abb. 9.4 ist das Umspannwerk aus Abb.9.2 und 9.3 in der finalen Version zu sehen.

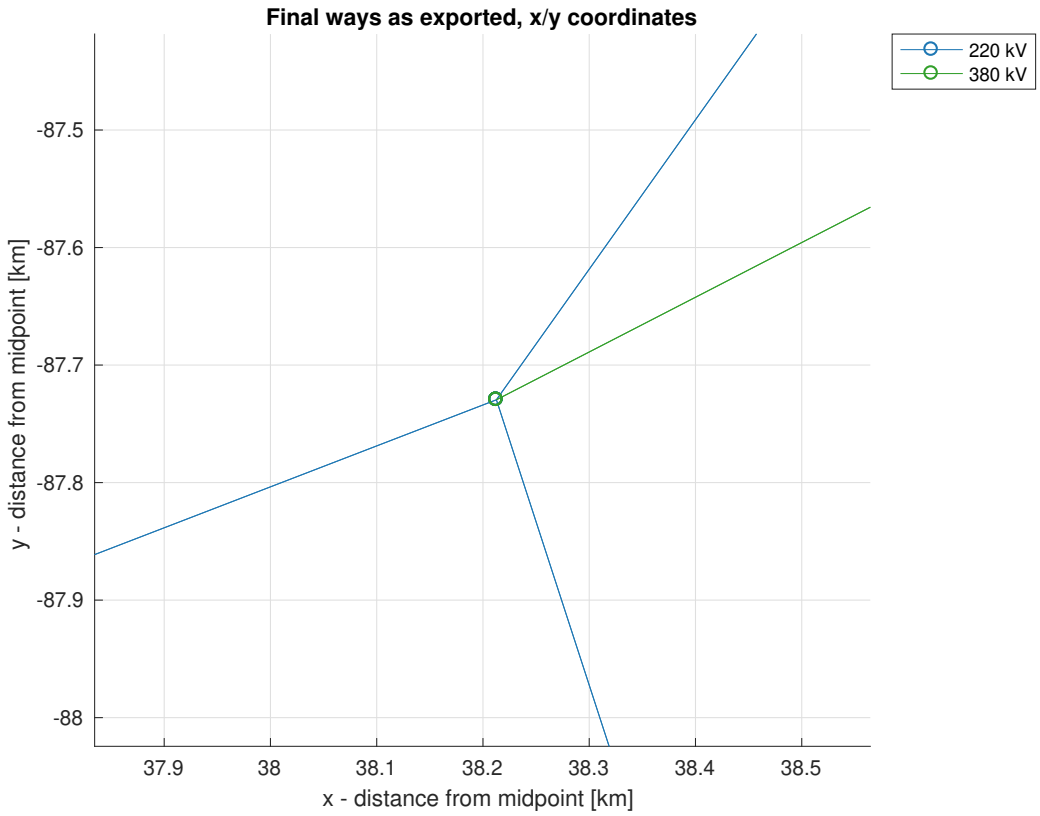


Abbildung 9.4: Visualisierung von Funktion *my_plot_ways_final()*

10 Ergebnis

In diesem letzten Kapitel werden die Ergebnisse des MATLAB Programms diskutiert. Als Beispiel dazu dient das Höchstspannungsnetz in Österreich, genaugenommen der 220 kV und 380 kV Spannungsebene.

In Abb. 10.1 ist ein Auszug der Website [4] zu sehen. In der Abfrage werden sämtliche Way-Elemente dargestellt, welche im *field* »power« den Wert »line« und im *field* »voltage« den Wert »220 000« oder »380 000« aufweisen.

Es ist hierbei anzumerken, dass in der visuellen Darstellung im Browser ein Renderfehler auftritt und nicht sämtliche Way-Elemente des Datensatzes dargestellt werden. Dies ist deutlich daran zu erkennen, dass in Tirol und im Burgenland Leitungsstücke nicht dargestellt werden, obwohl nach manueller Kontrolle diese Way-Elemente sich sehr wohl im Export-Datensatz befinden.

In Abb. 10.2 ist eben jener Datensatz aus Abb. 10.1 nach Durchlauf des MATLAB Programms dargestellt.

Die Visualisierung erfolgte mit der Funktion *my_plot_ways_final()*, wie sie in Abschnitt 9.3 beschrieben wurde. Hier ist sind auch deutlich die in Abb. 10.1 fehlenden Leitungsstücke in Tirol sowie im Burgenland zu erkennen.

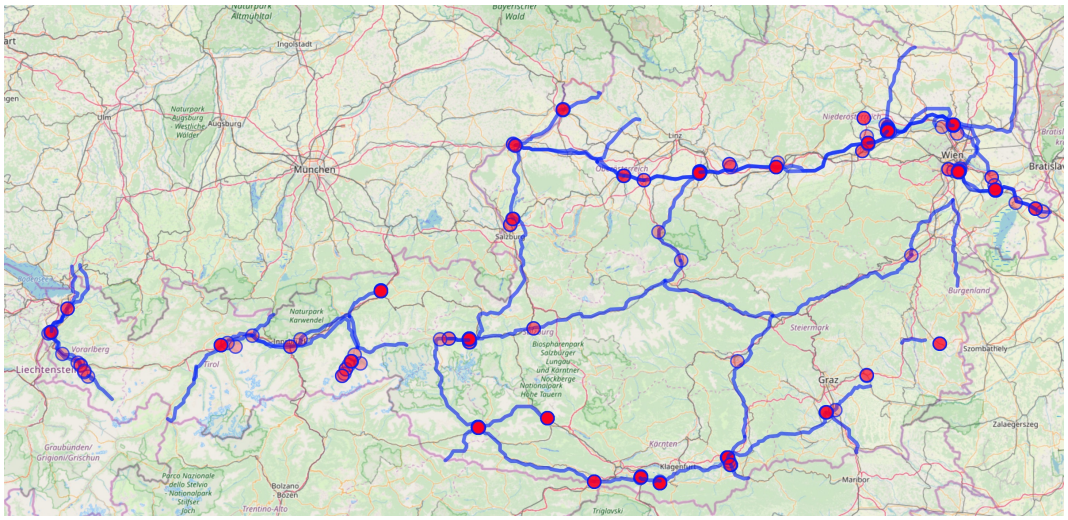


Abbildung 10.1: *OpenStreetMap* - Datenbasis vor MATLAB Skript [4]

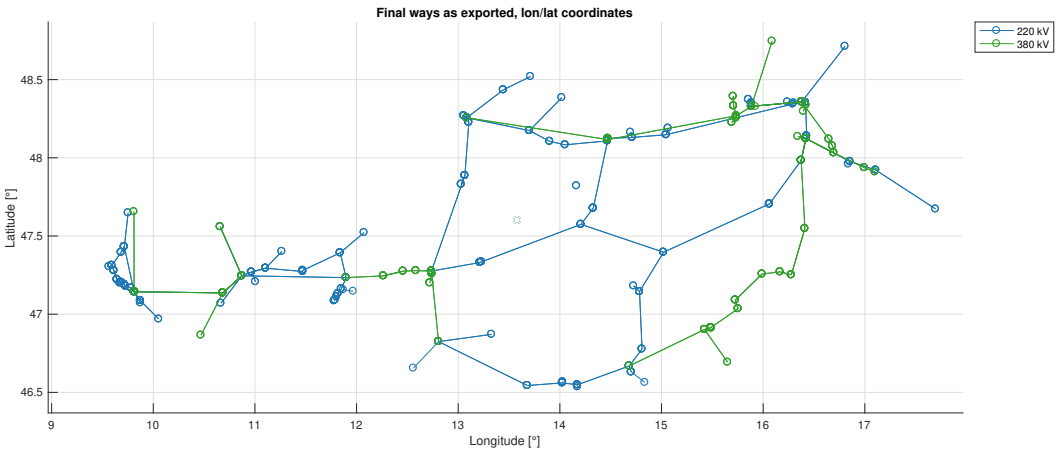


Abbildung 10.2: *OpenStreetMap* - Datenbasis nach MATLAB Skript

In Abb. 10.3 ist das österreichische Höchstspannungsnetz der 220 kV und 380 kV Ebene in *VISU*, dem Visualisierungsprogramm von *ATLANTIS* abgebildet.

Die Daten dieser Leitungen wurden händisch aus der Netzkarte der *ENTSO-E* und anderen Datenquellen übernommen, wie in Abschnitt 2 beschrieben wurde.

Im Vergleich dazu ist der Datensatz von *OpenStreetMap*, wie in Abb. 10.2 dargestellt, in Abb. 10.4 ebenso von *VISU* visualisiert.

Ein näherer Vergleich der Abbildungen 10.3 und 10.4 zeigt folgende Erkenntnisse:

- Die Höchstspannungsnetze stimmen sehr gut überein,
- das Netz von *OpenStreetMap* ist detailreicher als das der *ENTSO-E*,
- dennoch sind einige Leitungen aus dem Netz der *ENTSO-E* nicht in der Datenbasis von *OpenStreetMap* enthalten,
- grenzüberschreitende Leitungen wurden beim händischen Hinzufügen bis ins Nachbarland geführt, im Datensatz von *OpenStreetMap* enden diese im Normalfall am Grenzübergang.

Zusammenfassend kann gesagt werden, dass kleinere Unterschiede sehr wohl zu erkennen sind. Es lässt sich jedoch nicht pauschalisieren, welcher der beiden Datensätze die qualitativ hochwertigeren Informationen aufweist, da dies von Detail zu Detail unterschiedlich zu beurteilen ist.

Die Daten aus Abb. 10.4 wurden in ca. 5 - 10 Arbeitsminuten gewonnen und bedürfen mit manueller Nachbearbeitung bei diesem Datensatzumfang insgesamt circa einen halben bis einen ganzen Arbeitstag, um das Höchstspannungsnetz von Österreich für *ATLANTIS* zugänglich zu machen.

In diesem Arbeitszeitraum ist es auch möglich, die Komplexität zu reduzieren und so den selben Detailgrad der *ENTSO-E* Netzkarte zu erreichen, bzw. den Datensatz aus *OpenStreetMap* dem der *ENTSO-E* Netzkarte anzugleichen.

Die händische Übertragung der Daten aus Abb. 10.3 nahmen im Gegensatz dazu jedoch mehrere Tage bis mehrere Wochen Personeneinsatz in Anspruch.

Im Angesicht dieser überwältigenden Zeit- und Kostenersparnis sowie der häufigen Nichtverfügbarkeit einer offiziellen Netzkarte, sind die kleineren detaillierten Unterschiede vernachlässigbar.

Dem in dieser Arbeit vorgestellten MATLAB Programm kann somit ohne weiteres attestiert werden, die in Abschnitt 2 aufgezählten Punkte erfüllt zu haben und für *ATLANTIS* von sehr großem Nutzen zu sein.

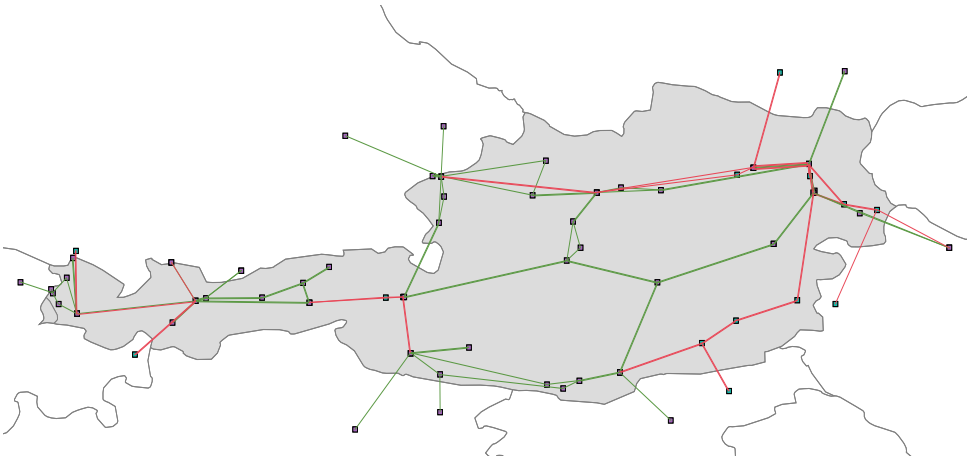


Abbildung 10.3: *ATLANTIS* - Datenbasis *ENTSO-E* [2]

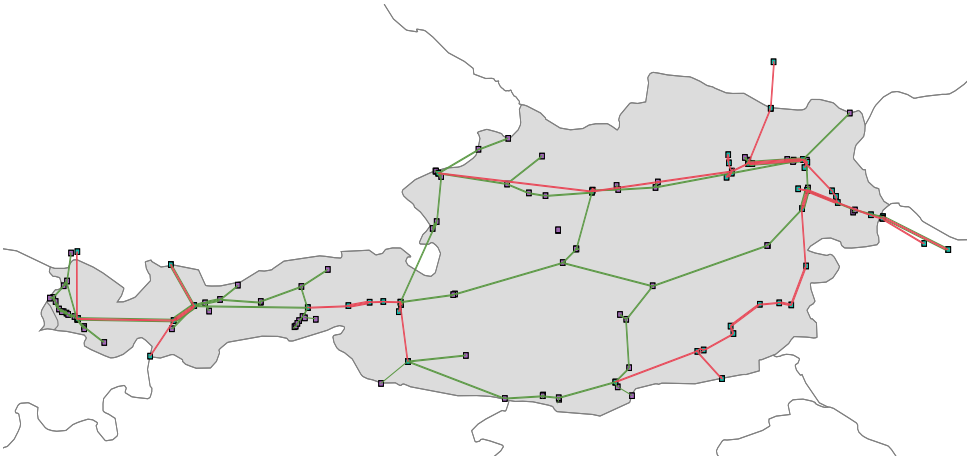


Abbildung 10.4: *ATLANTIS* - Datenbasis *OpenStreetMap* [2]

11 Ausblick

Durch den modularen Aufbau lässt sich das Programm in Zukunft leicht anpassen. Dies kann der Fall sein, wenn die Daten bezüglich eines anderen Aspektes untersucht werden sollen.

Beispielsweise kann das die Erkennungen und Entfernung von Leitungen für den Bahnstrom sein, dies könnte durch die Anzahl der Kabel oder durch Schlagwörter wie »ÖBB«, welche im Namen der Leitung vorkommen könnten, bewerkstelligt werden.

Sollten neue Datenquellen gefunden werden, könnte eine weitere Importfunktion geschrieben werden, welche Daten aus einer anderen Quelle importiert. Sämtliche Funktionen des Programms könnten dennoch weiterhin genutzt werden.

Aus Kompatibilitätsgründen, konkret die Notwendigkeit der optionalen und kostenpflichtigen MATLAB *Mapping-Toolbox*, wurde in diesem Programm bewusst auf eine Visualisierung mit geographischen Ländergrenzen verzichtet. Diese Art von Visualisierung könnte aber nützlich sein.

Eine weitere Erweiterungsmöglichkeit wäre die Berechnung der tatsächlichen Länge einer Leitung. Die Daten dazu sind grundsätzlich vorhanden, werden jedoch derzeit in der Importfunktion verworfen. In dieser Version wird als Leitungslänge die Luftlinie zwischen den Endpunkten genutzt, diese Berechnung ist aber nicht akkurat für die in *ATLANTIS* durchgeführten Berechnungen der Leitungsverluste.

12 Zusammenfassung

Zusammengefasst kann gesagt werden, dass der Import der Daten von *OpenStreetMap* in ein für *ATLANTIS* lesbares Format sehr gut funktioniert.

Der Datensatz wird erfolgreich auf das wesentliche reduziert und dennoch werden dem Benutzer Möglichkeiten geboten, die wichtigsten Informationen schnell zu erkennen und alle anderen im Datensatz vorhanden Zusatzinformationen gut und übersichtlich analysieren zu können.

Verschiedenste Visualisierung helfen die Daten besser interpretieren und Einstellungen konkreter setzen zu können. Durch die optionale Aktivierbarkeit dieser Visualisierungen können große Datenmengen dennoch sehr effizient und schnell bearbeitet werden.

Durch die Verwendungen von einfachen MATLAB Standardfunktionen und ansonsten ausschließlich selbstgeschriebenen, gut kommentierten Funktionen, ist das Programm leicht verständlich und weist eine hohe Kompatibilität zu älteren Versionen von MATLAB auf.

Das Programm ist in der Lage, mit verschiedenste Fehlerfällen selbstständig umzugehen, ohne eine *Error-Message* anzuzeigen und somit die Ausführung vorzeitig und unvollständig zu beenden.

Abbildungsverzeichnis

Matlab Code Verzeichnis

Literaturverzeichnis

[1] *https://www.tugraz.at/institute/iee/atlantis*, abgerufen am 15.10.2019

[2] Visualisierungsprogramm *VISU* für *ATLANTIS*, Institut für Elektrizitätswirtschaft und Energieinnovation, Technische Universität Graz

[3] *https://www.entsoe.eu/data/map*, abgerufen am 15.10.2019

[4] *https://overpass-turbo.eu*, abgerufen am 15.10.2019

[5] *https://de.mathworks.com/help/matlab/ref/format.html*, abgerufen am 15.10.2019

[6] *https://gis.stackexchange.com/questions/75528/understanding-terms-in-length-of-degree-formula/75535*, abgerufen am 15.10.2019

[7] *https://de.mathworks.com/help/matlab/data_analysis/missing-data-in-matlab.html*, abgerufen am 15.10.2019

[8] *https://de.mathworks.com/help/matlab/matlab_prog/vectorization.html*, abgerufen am 15.10.2019

[9] Colormap »paired«, *www.ColorBrewer.org*, abgerufen am 15.10.2019