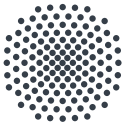

Universität Stuttgart
Institut für Straßen- und Verkehrswesen
Lehrstuhl für Verkehrsplanung und Verkehrsleittechnik

Lutftlinientool zum Erstellen von Luftlinienverbindungen für gegebene Bezirke anhand der RIN.

Generated by Doxygen 1.8.17

Inhaltsverzeichnis

1 Anwendung

1.1 Anwendung via Dialog

Zwei Möglichkeiten

- Visumversion ist noch geschlossen: GUI extern starten (*llt_GUI.py* ausführen)
- Visumversion ist bereits geöffnet:
  - Das Ausführen der GUI (*llt_GUI.py*) in das Skriptmenü integrieren
  - Skript via Skriptmenü starten Hinweis: Wenn die GUI mehrmals gestarten & beendet wird, erscheint eine Fehlermeldung. Diese kann ignoriert werden, die Funktionalität ist trotzdem gegeben.

1.2 Aufruf via Code

Das Luftlinientool kann auch ohne GUI angewendet werden. Dazu muss als Codeausführung eine Instanz der Klasse LuftlinienCalculator erstellt werden. Danach kann auf die Methoden der Instanz (Import, Berechnung, Export) zugegriffen werden Ein Beispiel ist unter *Bsp_Aufruf_ohne_GUI.py* zu sehen.

auszuführende Schritte

1. Parameter setzen (welche VFS, Attributswerte etc.)
2. Luftlinienkalkulatorobjekt initialisieren
   Code: Aufruf Konstruktor mit Parameterübergabe
   GUI: "Daten einlesen" in Toolbar ausführen
3. Luftlinienverbindungen berechnen/erzeugen
   Code: Aufruf calculate_main Methode
   GUI: "Berechnung Luftlinien-Netz" in Toolbar ausführen
4. Ergebnisse in gewünschter Form nach Visum exportieren
   Code: Aufruf der export_matrix/export_net Methode
   GUI: Die entsprechenden Buttons (Mtx/Net) in der Spalte "anlegen in Visum als" verwenden. Alternativ überträgt der Button "Import nach Visum alle VFS Strecken + Mtx" die kombinierten Ergebnisse aller VFS.

Anmerkungen:

- Bei der GUI werden aktuelle Berechnungen zurückgesetzt, wenn die Auswahl eines der Bezirksattribtue geändert wird. Dabei wird eine neue Instanz des LLT Kalkulators erstellt.
- Werden Bezirkswerte in Visum geändert, werden diese nicht automatisch im LLT Kalkulator geändert. Deshalb muss der Verfahrensablauf ab Schritt 2 wieder ausgeführt werden.
- Die initialen Parameterwerte können in der GUI über das Tool "Defaultwerte" wieder aufgerufen werden

- "Ergebnisse initialisieren" ermöglicht das Löschen bereits vorhandener Ergebnisse
- Schritt 3 verwendet die aktuell in der GUI eingegebenen Parameter. Vor der Rechnung mit neuen Parametern empfiehlt sich das Löschen der vorhandenen Ergebnisse ("Ergebnisse initialisieren").

1.3 Vorraussetzung

Netz mit kategorisierten Bezirken:

- Attribut für die Zentralität (Bezirke): je kleiner die Zahl, desto größer ist die Zentralität des Bezirks
  Bsp.: Angabe der Zentralität über die Typnummer
  - 0 ... Metropolregion
  - 1 ... Oberzentrum
  - 2 ... Mittelzentrum
  - 3 ... Grundzentrum
  - 4 ... Ort ohne zentrale Funktion
  - 5 ... Teilort
- (optional) aktiver Bezirksfilter
- (optional) Angabe eines Attributs, welcher Bezirk als Quelle verwendet werden soll) {0=Nein, 1=Ja}
- (optional) Angabe eines Attributs, welcher Bezirk als Ziel verwendet werden soll) {0=Nein, 1=Ja}

2 Allgemeines Vorgehen und Methodik

2.1 Ermittlung der Verbindungen für eine Verbindungsfunktionsstufe VFS

1. Initialisierung und Filterung von Bezirken
2. Bestimmung der Dreiecksverbindungen zwischen den Bezirken durch Delaunay-Triangulation.
3. Berücksichtigung der n-nächsten Nachbarn.
4. Sicherstellung der Verbindung zu Versorgungszentren.
5. Filtern der Bezirke und Verbindungen.
6. Erstellung einer symmetrischen Adjazenzmatrix, die alle Verbindungen korrekt abbildet.

Umsetzung in der Funktion calculate_vfs (self, vfs) der Klasse LuftlinienCalculator.

Initialisierung und Filterung von Bezirken

- Zunächst werden die Bezirke mit Stufe $\geq$ VFS gefiltert, um nur die Bezirke zu berücksichtigen, die für die aktuelle Verbindungsfunktionsstufe VFS relevant sind. Bezirke, die inaktiv sind oder keine Verbindungsanforderungen erfüllen, werden ausgeschlossen.
- Eine Überprüfung erfolgt, um sicherzustellen, dass keine Bezirke identische Koordinaten haben, da dies bei der Delaunay-Triangulation zu Problemen führen könnte.

Delaunay-Triangulation

- Bei genügend aktiven Bezirken wird eine Delaunay-Triangulation durchgeführt, um die Bezirke zu Dreiecken zu verbinden. Diese Dreiecke repräsentieren die Nachbarschaften zwischen den Bezirken.
- Die resultierenden Nachbarschaftsverbindungen werden in einer Adjazenzmatrix (symmetrisch) gespeichert.
- Kann auch Verbindungen besserer Funktionsstufen ermitteln (Bezirke der Verbindung mit Stufe $>$ VFS).

Berücksichtigung n-nächste Nachbarn

- Berechnung Erreichbarkeitsmatrix der Adjazenzmatrix A in n Schritten

$$E_n = A^n$$

- Update der Adjazenzmatrix = Erreichbarkeitsmatrix.

Verbindungen mit Versorgungszentren

- Es wird überprüft, ob jeder Bezirk mit einer ausreichenden Anzahl an Versorgungszentren verbunden ist.
- Wenn dies nicht der Fall ist, werden die nächstgelegenen Versorgungszentren ermittelt und Verbindungen hergestellt.

Aktive und inaktive Verbindungen

- Nur aktive Bezirke werden als Quelle und Ziel von Verbindungen berücksichtigt.
- Der Filter „istQuelle“ definiert, welcher Bezirk als Quelle verwendet wird; „istZiel“ definiert die Ziel-Bezirke. Aus Symmetriegründen gilt für die Relationen $od = do$. Dennoch sind „istZiel“-Bezirke kein Teil der Dreiecke, sondern werden von den „istQuelle“-Bezirken erreicht.
- Umsetzung: Auf die Adjazenzmatrizen wird eine Filtermaske gelegt. Diese ist das Ergebnis des dyadischen Produkts von „istQuelle“ und „istZiel“ und anschließender Symmetriesierung.

Beispiel: Gegeben sei „istQuelle“ $o = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \end{bmatrix}^T$ und „istZiel“ $d = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \end{bmatrix}^T$. Damit folgt:

$$M_{o \vee d} = o \cdot d^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$M_{symmetrisch} = M_{o \vee d} + M_{o \vee d}^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

2.2 Export der Luftlinieninfrastruktur der VFS

Der Export der ermittelten Luftlinien der spezifizierten Verbindungsfunktionsstufen als eigenständige Infrastruktur (Knoten, Streckentypen, Strecken, Anbindungen) ermöglicht die Veranschaulichung der Ergebnisse oder die Verwendung des resultierenden Dreiecksnetzes.

Bei der Umwandlung der Adjazenzmatrizen sind zwei Herausforderungen zu beachten:

- Kompatibilität mit der bereits bestehenden Infrastruktur eines Visummodells.
- Vermeidung von Überschneidungen bei der Kombination mehrerer Verbindungsfunktionsstufen und

korrekte Zuordnung der VFS bei Strecken, die von mehreren Stufen berücksichtigt werden.

Deshalb erfolgt der Export der Luftlinieninfrastruktur in 2 Schritten:

1. Bei Bedarf werden die benötigten Infrastrukturobjekte (Knoten, Streckentypen, Strecken, Anbindungen) unter Berücksichtigung aller berechneten Adjazenzmatrizen aktualisiert und als Attribute der Instanz des Luftlinienkalkulators gespeichert. Die Umsetzung erfolgt in der Funktion extract_net der Klasse LuftlinienCalculator. Die für Visum relevante Nummerierung der Objekte berücksichtigt dabei die vorhandenen Objekte in der aktiven Visuminstanz.
2. Die Funktion export_net erstellt eine .net Datei mit der Infrastruktur der gewählten Verbindungsfunktionsstufen und öffnet diese in der aktuellen Visuminstanz.

3 Dateien

3.1 Überblick Dateien

Hier folgen die Dateien mit einer Kurzbeschreibung (wenn verfügbar):

3.2 Pakete

Hier folgen die Pakete mit einer Kurzbeschreibung (wenn verfügbar):

3.3 Dokumentation luftlinientool.py

Enthält allgemeine Methoden und die Klasse LuftlinienCalculator zur Berechnung von Luftlinienverbindungen unter Berücksichtigung der Zentralitäten von Bezirken.

Klassen

- class LuftlinienCalculator
  Klasse LuftlinienCalculator eine Instanz der Klasse enthält Attribute und Berechnungsmöglichkeiten um die VFS zwischen Bezirken zu ermitteln.

Funktionen

- def open_visum (path, version=240)
  Öffnet eine Visuminstanz falls nicht bereits offen ermöglicht simultanes Aufrufen der Datei Visumintern und -extern.
- def write_object_to_net (object, df_object_attributes_to_write, file)
  Exportiert die Daten eines Visumobjekttyps in Netzdateiformat.

- def is_symmetric (matrix, tol=1e-8)
  Überprüft eine Matrif auf Symmetrie.
- def calculate_distance_coordinates_haversine (x1, y1, vec_x2, vec_y2)
  Berechnung der Distanz zwischen Koordinaten (Lat, Lon) Implementation der Haversine Formel.
- def calculate_eucl_distance_coordinates (x1, y1, vec_x2, vec_y2)
  Berechnung der Distanz zwischen Koordinaten (x, y) Euklidische Distanzberechnung!
- def get_nearest_points_from_set (x_point, y_point, array_points, formula, n=None)
  Identifiziert die nächsten n Punkte aus einer gegebenen Punktemenge zu einem einzelnen Punkt. Zuerst werden die Distanzen aller Punkte zu dem einzelnen Punkt berechnet. Anschließend werden die n am kürzesten entfernten Punkte gefiltert und deren Indizes zurückgegeben.
- def show_info (Path path_scripts=Path.cwd())
  Öffnet die Readme Datei.

3.3.1 Ausführliche Beschreibung

Enthält allgemeine Methoden und die Klasse LuftlinienCalculator zur Berechnung von Luftlinienverbindungen unter Berücksichtigung der Zentralitäten von Bezirken.

3.3.2 Dokumentation der Funktionen

calculate_distance_coordinates_haversine()

```
def luftlinientool.calculate_distance_coordinates_haversine (
                x1,
                y1,
                vec_x2,
                vec_y2 )
```

Berechnung der Distanz zwischen Koordinaten (Lat, Lon) Implementation der Haversine Formel.

Parameter

in	*x1*	x-Koordinate Punkt 1
in	*y1*	y-Koordinate Punkt 1
in	*vec_x2*	x-Koordinate Punktevektor
in	*vec_y2*	y-Koordinate Punktevektor

Rückgabe

Vektor mit den Distanzen aller Punkte des Punktevektors zu Punkt 1

Definiert in Zeile 74 der Datei luftlinientool.py.

calculate_eucl_distance_coordinates()

```
def luftlinientool.calculate_eucl_distance_coordinates (
              x1,
              y1,
              vec_x2,
              vec_y2 )
```

Berechnung der Distanz zwischen Koordinaten (x, y) Euklidische Distanzberechnung!

Parameter

in	*x1*	x-Koordinate Punkt 1
in	*y1*	y-Koordinate Punkt 1
in	*vec_x2*	x-Koordinate Punktevektor
in	*vec_y2*	y-Koordinate Punktevektor

Rückgabe

Vektor mit den Distanzen aller Punkte des Punktevektors zu Punkt 1

Definiert in Zeile 101 der Datei luftlinientool.py.

get_nearest_points_from_set()

```
def luftlinientool.get_nearest_points_from_set (
              x_point,
              y_point,
              array_points,
              formula,
              n = None )
```

Identifiziert die nächsten n Punkte aus einer gegebenen Punktemenge zu einem einzelnen Punkt.

Zuerst werden die Distanzen aller Punkte zu dem einzelnen Punkt berechnet. Anschließend werden die n am kürzesten entfernten Punkte gefiltert und deren Indizes zurückgegeben.

Parameter

in	*x_point*	x-Koordinate des Referenzpunktes
in	*y_point*	y-Koordinate des Referenzpunktes
in	*array_points*	Array mit den x- & y-Koordinaten der Punkte
in	*n*	gewünschte Punkteanzahl

Rückgabe

list_indizes: Liste der Indizes der nächstgelegenen n Punkte

Definiert in Zeile 120 der Datei luftlinientool.py.

is_symmetric()

```
def luftlinientool.is_symmetric (
              matrix,
              tol = 1e-8 )
```

Überprüft eine Matrif auf Symmetrie.

Parameter

in	*matrix*	Matrix, die auf Symmetrie getestet werden soll
in	*tol*	Toleranz für erlaubte Abweichung, default 1e-8

Rückgabe

: True oder False

Definiert in Zeile 61 der Datei luftlinientool.py.

open_visum()

```
def luftlinientool.open_visum (
              path,
```

version = 240)

Öffnet eine Visuminstanz falls nicht bereits offen ermöglicht simultanes Aufrufen der Datei Visumintern und -extern.

Parameter

path	Dateipfad (Path/str) einer Visumversionsdatei
version	Visumversion, default 22

Rückgabe

: Visuminstanz

Definiert in Zeile 24 der Datei luftlinientool.py.

write_object_to_net()

```
def luftlinientool.write_object_to_net (
              object,
              df_object_attributes_to_write,
              file )
```

Exportiert die Daten eines Visumobjekttyps in Netzdateiformat.

Parameter

in	*object*	Visumobjekttyp (Singular), z.B. 'link'
in	*df_object_attributes_to_write*	Datentabelle des Objekts. Tabelle enthält nur Attribute, die in Visum importiert werden können (insbesondere die notwendigen Attribute)
in	*file*	Zieldatei, im Schreib- oder Erweiterungsmodus (w oder a)

Definiert in Zeile 45 der Datei luftlinientool.py.

3.4 Dokumentation testing_scenarios.py

Enthält Methoden und einen Ablauf um definierte Testszenarien auszuführen und das Ergebnis in einer PPP darzustellen.

3.4.1 Ausführliche Beschreibung

Enthält Methoden und einen Ablauf um definierte Testszenarien auszuführen und das Ergebnis in einer PPP darzustellen.

4 Klassen

4.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

4.2 Klassenhierarchie

Die Liste der Klassen ist – mit Einschränkungen– alphabetisch sortiert:

4.3 Dokumentation luftlinientool.LuftlinienCalculator

Öffentliche Methoden

- def __init__ (self, source, str attr_vfs="TypeNo", dict dict_vfs={"VFS 0":0, "VFS 1":1, "VFS 2":2, "VFS 3":3, "VFS 4":4, "VFS 5":5}, max_entfernung=1, anz_versorger=0, attr_quelle=None, attr_ziel=None, bool use_filter=False, str formula_distance="euclidean", path_output=None)
  Konstruktor.

- def adj_matrix_to_links (self, list_vfs=None)
  Übersetzt die Adjazenzmatrizen der gewünschten VFS in eine Streckenliste.
- def adj_matrix_to_set_of_connected_zones (self, vfs, use_zone_names=True)
  Wandelt die Adjazenzmatrix in eine Liste der verbundenen Bezirke je Bezirk um.
- def calculate_reachability_max_steps (self, max_steps, vfs)
  Berechnet, welche Nachbarn innerhalb von n Schritten erreicht werden können.
- def calculate_main (self)
  Berechnet für jede hinterlegte VFS der Instanz die Adjazenzmatrix.
- def calculate_vfs (self, vfs)
  Berechnet die Verbindungen einer VFS.
- def delete_unused_nodes (self)
  Löscht Knoten in Visum, die keine Strecken anbinden.
- def export_matrix (self, list_vfs=None)
  Exportiert die gewünschten Adjazenzmatrizen entweder direkt nach Visum (falls Visuminstanz verknüpft) oder als .mtx datei.
- def extract_net (self)
  Erstellt die Infrastrukturobjekte als Vorbereitung für den Export der Infrastruktur in Form von dicts für Knoten, Strecken, Streckentypen.
- def export_net (self, links_additive=True, list_vfs=None, create_connectors=True)
  Exportiert eine Netzdatei falls eine Visuminstanz übergeben wird, wird die Netdatei in Visum geladen.
- def export_zones_uda_connections (self, vfs)
  Exportiert die Verbindungen sowie die Anzahl der Verbindungen als Bezirk UDAs nach Visum.
- def init_results (self)
  Initialisiert die Adjazenzmatrizen.
- def filter_links_vfs (self)
  Filtert die Strecken der eingefügten Streckentypen in Visum.
- def filter_zones_source_targets (self, bool filterFromZones=True)
  Filtert die Bezirke, für die das gegebene Attribut größer 0 ist.
- def delete_added_links (self)
  Löscht die Strecken der VFS.

Öffentliche Attribute

- debug_mode
  Flag Debugmodus.
- attr_zones
  relevante Bezirksattribute
- attr_central_level
  Attribut Zentralität.
- path_output
  Rückgabeverzeichnis.
- vfs
  Liste der VFS, die bearbeitet werden sollen.
- formula_dist
  Abstandsberechnung.
- nachbarschaftsgrad_vfs
  Vorgabe, bis zu welchem Nachbarschaftsgrad gleichrangige Verbindungen verfolgt werden sollen (ehemals Austauschfkt)
- anz_versorger_vfs

Vorgabe, wie viele (höherrangige) Versorger verbunden werden sollen.

- visum
  Visuminstanz.
- zones
  Tabelle mit den Bezirksdaten.
- attr_is_from_zone
  Attribut Quellfilter.
- attr_is_to_zone
  Attribut Zielfilter.
- language
  Eingestellte Sprache Visuminstanz.
- dict_export_zone2node
  LookupTable Infrastruktur: Dem Bezirk zugeordnete Knotennummer.
- dict_export_links_vfs
  LookUpTable Infrastruktur: Zuordnung interne Streckennummer zu Streckennummer Visum.
- dict_export_linktypes
  LookUpTable Infrastruktur: Zuordnung Verbingungsfunktionsstufe - Streckentyp Visum.
- edges
  DataFrame mit den Streckendaten der Luftlinienverbindungen.
- matrizen_VFS
  Dict mit den resultierenden Adjazenzmatrizen der Verbindungsfunktionsstufen.

4.3.1 Ausführliche Beschreibung

Die Klasse enthält Attribute und Berechnungsmöglichkeiten um die VFS zwischen Bezirken zu ermitteln

Definiert in Zeile 150 der Datei luftlinientool.py.

4.3.2 Beschreibung der Konstruktoren und Destruktoren

__init__()

```
def luftlinientool.LuftlinienCalculator.__init__ (
              self,
              source,
             str  attr_vfs = "TypeNo",
             dict  dict_vfs = {"VFS 0": 0, "VFS 1": 1, "VFS 2": 2, "VFS 3": 3, "VFS 4": 4, "VFS 5"↩
: 5},
              max_entfernung = 1,
              anz_versorger = 0,
              attr_quelle = None,
```

```
 attr_ziel = None,
bool  use_filter = False,
str  formula_distance = "euclidean",
 path_output = None )
```

Konstruktor.

Parameter

source	Dateiname (str) oder Visuminstanz
attr_vfs	Name des Bezirkattributs, das die Kategorisierung in OZ,MZ,UZ ... enthält. Default: TypeNr
dict_vfs	Dictionary, das die Attributwerte für die jeweiligen VFS enthält
max_entfernung	Angabe, bis zu welcher Entfernung, Nachbar angebunden werden
anz_versorger	Angabe, an wie viele höherrangige Zentren ein Bezirk angebunden werden soll
attr_quelle	Name des Attributs, das angibt, ob der Bezirk als Quelle berücksichtigt wird. Default: None
attr_ziel	Name des Attributs, das angibt, ob der Bezirk als Ziel berücksichtigt wird. Default: None
use_filter	gibt an, ob nur aktive Bezirke berücksichtigt werden. Kann nur verwendet werden, wenn source = Visuminstanz
formula_distance	definiert die Distanzfunktion für die Ermittlung der Versorgungszentren. Anmerkung: Für die Triangulation werden die Luftlinienverbindungen anhand der euklidischen Distanz ermittelt. Delaunay-Triangulation funktioniert nur bei einer Projektion der Lat/Lon Koordinaten.
path_output	optionale Möglichkeit einen Pfad für den Dateiexport anzugeben. Default: None. Dann wird bei bedarf der aktuelle Ordner verwendet.

Definiert in Zeile 165 der Datei luftlinientool.py.

4.3.3 Dokumentation der Elementfunktionen

adj_matrix_to_links()

```
def luftlinientool.LuftlinienCalculator.adj_matrix_to_links (
              self,
              list_vfs = None )
```

Übersetzt die Adjazenzmatrizen der gewünschten VFS in eine Streckenliste.

Parameter

list_vfs	Liste der VFS. Falls nicht gegeben, werden alle VFS der Instanz verwendet

Rückgabe

df_edges: DataFrame mit allen Strecken und ihrer VFS. Achtung: Duplikate werden nicht entfernt

Definiert in Zeile 282 der Datei luftlinientool.py.

adj_matrix_to_set_of_connected_zones()

```
def luftlinientool.LuftlinienCalculator.adj_matrix_to_set_of_connected_zones (
              self,
              vfs,
              use_zone_names = True )
```

Wandelt die Adjazenzmatrix in eine Liste der verbundenen Bezirke je Bezirk um.

Parameter

vfs	str, Name der zu betrachtenden VFS
use_zone_names	bool, falls True werden die hitnerlegten Bezirksnamen verwendet

Rückgabe

df_set_zones: DataFrame mit list Objekt je Bezirk und einer Spalte, die die Anzahl enthält

Definiert in Zeile 319 der Datei luftlinientool.py.

calculate_main()

```
def luftlinientool.LuftlinienCalculator.calculate_main (
              self )
```

Berechnet für jede hinterlegte VFS der Instanz die Adjazenzmatrix.

Rückgabe

Keine Rückgabe. Die Ergebnisse werden intern gespeichert.

Definiert in Zeile 350 der Datei luftlinientool.py.

calculate_reachability_max_steps()

```
def luftlinientool.LuftlinienCalculator.calculate_reachability_max_steps (
              self,
              max_steps,
              vfs )
```

Berechnet, welche Nachbarn innerhalb von n Schritten erreicht werden können.

Parameter

max_steps	maximale Entfernung (Schritte)
vfs	zu untersuchende VFS

Rückgabe

matrix: Adjazenzmatrix für die Erreichbare Nachbarn innerhalb der max-steps

Definiert in Zeile 340 der Datei luftlinientool.py.

calculate_vfs()

```
def luftlinientool.LuftlinienCalculator.calculate_vfs (
              self,
              vfs )
```

Berechnet die Verbindungen einer VFS.

Parameter

vfs	die Verbindungsfunktionsstufe, für die Verbindungen ermittel werden

Definiert in Zeile 366 der Datei luftlinientool.py.

delete_added_links()

```
def luftlinientool.LuftlinienCalculator.delete_added_links (
              self )
```

Löscht die Strecken der VFS.

Rückgabe

Keine Rückgabe. Die Visuminstanz wird verändert.

Definiert in Zeile 865 der Datei luftlinientool.py.

delete_unused_nodes()

```
def luftlinientool.LuftlinienCalculator.delete_unused_nodes (
              self )
```

Löscht Knoten in Visum, die keine Strecken anbinden.

Alle Knoten ohne Strecken werden gefiltert & die aktiven Knoten werden gelöscht. Anschließend wird der Filter zurückgesetzt.

Rückgabe

Keine Rückgabe. Die Visuminstanz wird verändert.

Definiert in Zeile 513 der Datei luftlinientool.py.

export_matrix()

```
def luftlinientool.LuftlinienCalculator.export_matrix (
              self,
              list_vfs = None )
```

Exportiert die gewünschten Adjazenzmatrizen entweder direkt nach Visum (falls Visuminstanz verknüpft) oder als .mtx datei.

Vorhandene Matrizen werden überschrieben.

Parameter

visum	optionale Übergabe einer Visuminstanz. Default None
list_vfs	optionale Übergabe einer Menge an VFS. Default: None (alle des Objekts)

Definiert in Zeile 542 der Datei luftlinientool.py.

export_net()

```
def luftlinientool.LuftlinienCalculator.export_net (
                self,
                links_additive = True,
                list_vfs = None,
                create_connectors = True )
```

Exportiert eine Netzdatei falls eine Visuminstanz übergeben wird, wird die Netdatei in Visum geladen.

Parameter

visum	optionale Übergabe einer Visuminstanz. Default None
links_additive	falls False werden die existierenden Strecken in Visum gelöscht
list_vfs	Liste der VFS, die berücksichtigt werden sollen. Default: Alle des Objekts

Definiert in Zeile 693 der Datei luftlinientool.py.

export_zones_uda_connections()

```
def luftlinientool.LuftlinienCalculator.export_zones_uda_connections (
                self,
                vfs )
```

Exportiert die Verbindungen sowie die Anzahl der Verbindungen als Bezirk UDAs nach Visum.

Rückgabe

Keine Rückgabe. Die Visuminstanz wird verändert.

Definiert in Zeile 799 der Datei luftlinientool.py.

extract_net()

```
def luftlinientool.LuftlinienCalculator.extract_net (
```

```
            self )
```

Erstellt die Infrastrukturobjekte als Vorbereitung für den Export der Infrastruktur in Form von dicts für Knoten, Strecken, Streckentypen.

Wird aufgerufen, falls beim Export ein Objekt nicht in den dicts vorhanden ist. Verhindert die Mehrfachanlegung von Strecken und Knoten.

Rückgabe

Keine Rückgabe. Die Ergebnisse werden intern gespeichert.

Definiert in Zeile 628 der Datei luftlinientool.py.

filter_links_vfs()

```
def luftlinientool.LuftlinienCalculator.filter_links_vfs (
            self )
```

Filtert die Strecken der eingefügten Streckentypen in Visum.

Rückgabe

Keine Rückgabe. Die Visuminstanz wird verändert.

Definiert in Zeile 843 der Datei luftlinientool.py.

filter_zones_source_targets()

```
def luftlinientool.LuftlinienCalculator.filter_zones_source_targets (
             self,
            bool  filterFromZones = True )
```

Filtert die Bezirke, für die das gegebene Attribut größer 0 ist.

Rückgabe

Keine Rückgabe. Die Visuminstanz wird verändert.

Definiert in Zeile 851 der Datei luftlinientool.py.

init_results()

```
def luftlinientool.LuftlinienCalculator.init_results (
              self )
```

Initialisiert die Adjazenzmatrizen.

Rückgabe

Keine Rückgabe. Die Ergebnisse werden intern gespeichert.

Definiert in Zeile 832 der Datei luftlinientool.py.

4.4 Dokumentation llt_GUI.LLTFrame

Definiert das komplette Fenster, erzeugt die einzelnen Bestandteile und verbindet diese mit der Logik.

Abgeleitet von Frame.

Öffentliche Methoden

- def **__init__** (self)
- def **__set_properties__** (self)
- def **__set_layout__** (self)
- def **__bind_events__** (self)
- def **__set_values_vfs_buttons__** (self)
- def **event_set_default** (self, event=None)
- def **event_choose_attr** (self, event)
- def **event_calculate** (self, event)
- def **event_quit_button** (self, event)
- def **event_import_data** (self, event)
- def **event_info** (self, event)
- def **event_reset** (self, event)
- def **event_export_results** (self, event)
- def **event_export_net** (self, event)
- def **event_export_mtx** (self, event)

- def **event_export_master** (self, event)
- def **event_filter** (self, event)
- def **event_delete_links** (self, event)
- def **update_param_vfs** (self)

Öffentliche Attribute

- **buttons_value_n_versorger**
- **cb_quelle**
- **cb_ziel**
- **cb_vfs**
- **buttons_vfs_value**
- **buttons_value_k_nachbar_vfs**
- **button_vfs_active**
- **llt_calculator**
- **default_k_nachbar**
- **default_anz_vf**
- **attr_vfs**
- **attr_quelle**
- **attr_ziel**
- **visum**
- **list_attr**
- **attr_dist_fcn**
- **panel**
- **notebook**
- **tabMain**
- **tabLog**
- **menu_bar**
- **menu**
- **toolbar**

4.4.1 Ausführliche Beschreibung

Definiert das komplette Fenster, erzeugt die einzelnen Bestandteile und verbindet diese mit der Logik.

Definiert in Zeile 21 der Datei llt_GUI.py.

4.5 Dokumentation llt_GUI.MainTab

4.6 llt_GUI.MainTab Klassenreferenz

Spezifiziert & verwaltet den Tab mit den Eingabe- und Aktionsmöglichkeiten.

Abgeleitet von Panel.

Öffentliche Methoden

- def **__init__** (self, parent)
- def **__set_layout__** (self)
- def **__bind_events__** (self)

Öffentliche Attribute

- **cb_vfs**
- **cb_quelle**
- **cb_ziel**
- **button_vfs_active**
- **buttons_vfs_value**
- **buttons_value_k_nachbar_vfs**
- **buttons_value_n_versorger**
- **buttons_export_mat**
- **buttons_export_net**
- **btn_export_master**
- **cb_dist_fcn**

4.6.1 Ausführliche Beschreibung

Spezifiziert & verwaltet den Tab mit den Eingabe- und Aktionsmöglichkeiten.

Definiert in Zeile 349 der Datei llt_GUI.py.

4.7 Dokumentation llt_GUI.LogTab

Spezifiziert den Tab, der die Lognachrichten ausgibt.

Abgeleitet von Panel.

Öffentliche Attribute

- **logger**
- **log**
- **handler**

4.7.1 Ausführliche Beschreibung

Spezifiziert den Tab, der die Lognachrichten ausgibt.

Definiert in Zeile 522 der Datei llt_GUI.py.

Index