Anwenderdoku zur Ermittlung der Kabellängen

Installation des Programmes und Schnellstart-Guide

Installation

Das Programm mit allen Quellen befindet sich auf dem hauseigenen git Server und kann mit einen beliebigen git client auf einem Anwenderrechner über `git clone http://gitea.schoenenberger.de/mistangl/kabellaengen.git` geholt werden. Dieser Ordner kann auch einfach mit allen Unterordnern gezippt und auf einem anderen Rechner entpackt werden.

Danach muss auf dem Zielrechner (z.B. per Windows APP) Python 3.X installiert werden. Alternativ kann ein Python Interpreter von einem Netzlaufwerk verwendet werden. Der Pfad zu diesem Interpreter muss dann über eine *Umgebungsvariable* mit dem Namen **NETWORK_INTERPRETER_PATH** auf der Maschine gesetzt sein. Grundsätzlich wird eine lokale Installation eines Python Interpreters (von https://www.python.org/downloads/) empfohlen.

Zur Installation des Kabeltools muss in dem geclonten Ordner `kabellaengen\bin` die Datei `install.bat` **als Administrator** ausgeführt werden. Hierfür *Rechtsklick* auf die Datei und *als Administrator ausführen*. Es wird in der Folge ein Ordner auf dem Desktop des Computers mit dem namen *Kabeltool* erstellt. In diesem Ordner ist eine Verknüpfung *create_cables*.

Ist Python installiert, werden per Doppelklick automatisch auf die `bin\install_py.bat` alle nötigen python Package heruntergeladen. Diese werden dann automatisch aus dem Netz in den .venv kopiert.

Schnellstart und Vewendung des Programms

Im auf dem Desktop erzeugten Ordner findet sich das eigentliche Programm *create_cables*. Um das Programm auszuführen, genügt es die zu verarbeitende .**dxf-Datei** per *Drag and Drop* auf das Programm zu ziehen. Das öffnen der *Command-Shell* bestätigt den Start des Programms und der Anwender wird über den Ablauf informiert.

Die Ausgabe schreibt:

```
folgende Datei wurde uebergeben: Pfad\zu\meiner\dxfdatei.dxf
-- hole Positionen
reading file ..done
writing results to a json file ...
done
-- erzeuge Graph mit Routing
writing results to a json file ...
done
--zeichne Kabel in dxf Datei
creating new .dxf ..
done
copying layers (Racks, Subdistributors, ...) from original .dxf into new .dxf ..
done
creating new .dxf ..
done
creating excel file with cable information ..
done
C:\10-Develop\kabellaengen\work\dxfdatei_cables.dxf
C:\10-Develop\kabellaengen\work\dxfdatei_cables.xlsx
C:\10-Develop\kabellaengen\work\dxfdatei_positions.json
C:\10-Develop\kabellaengen\work\dxfdatei_reduziert.dxf
C:\10-Develop\kabellaengen\work\dxfdatei_todraw.json
        5 Datei(en) verschoben.
Drücken sie eine beliebige Taste . . .
```

Das Drücken einer beliebigen Taste beendet das Programm und die Ausgabe kann verwendet werden.

! Hinweis !

Der Dateiname darf **keine Leerzeichen oder Sonderzeichen** enthalten. Verwende z. B. projekt_123.dxf statt mein projekt #1.dxf.

Standardmäßige Ausgabe des Programms

In den Ordner auf dem Desktop in dem das Kabeltool liegt werden nach Beendigung der Berechnungen die Ausgabedateien abgelegt. Standardmäßig sind dies vier Dateien, wovon letztlich zwei für den Anwender bestimmt sind. In der nachfolgenden Liste steht der name *dxfdatei* stellvertretend für die eingegebene Datei des Benutzers.

Dateiname	Details
`dxfdatei_cables.dxf`	`.dxf`-Datei, **mit Kabelwegen** (z. B. für Import ins Original-Layout als eigenes Layer) und gezeichneten Racks (ggf. in 3D)
`dxfdatei_cables.xlsx`	**Excel-Datei** mit Kabellängen, Längenübersicht und Fehler-Übersicht
`dxfdatei_positions`	**Zwischendatei** (nicht für Endnutzer bestimmt)
`dxfdatei_todraw`	**Zwischendatei** (nicht für Endnutzer bestimmt)

.dxf-Datei: *dxfdatei*_cables.dxf

Die standardmäßig erstellte und abgelegte Ausgabedatei `dxfdatei_cables.dxf` enthält die Kabelwege aller während des Programmablaufs behandelten Kabelverbindungen und kann beispielsweise als neues Layer in die .dxf-Datei der Anlage importiert werden. Neben den Kabelwegen werden auch die Kurzbezeichnungen (z.b. MA0060, BG3240, ...) sowie die Kürzel der Unterverteiler auf jeweils separaten Layern in die Zeichnungsdatei exportiert und ermöglichen damit die in sich schlüssige Verwendung dieser Datei. Desweiteren werden die von dem Programm erkannten Kabeltrassen eingezeichnet und nummeriert. Sofern diese im Ursprungslayout dreidimensional angelegt wurden, wird auch das in der Ausgabe berücksichtigt.

Excel-Datei: *dxfdatei*_cables.xlsx

Die standardmäßig erstellte und abgelegte Ausgabedatei `dxfdatei_cables.xlsx` enthält im fehlerfreien Fall drei Worksheets:

Length by ID:
zeigt die Kabel-ID, die tatsächliche Länge des Kabels, die Kabel-Atikelnummer und zuletzt den zugeörigen (aber gekürzten Bezeicher) für das Kabel:

Cable-ID	True Length (m)	Cable-ArtNr	Cable-Name (short)
UC0101-BG3241	2,9213	722001332	M12 St-0°/ M12 Bu-90° 3m
UC0101-BG3240	7,1784	722001334	M12 St-0°/ M12 Bu-90° 10m
UC0101-BG3270	15,4686	722001336	M12 St-0°/ M12 Bu-90° 20m
UC0101-BG3260	27,4471	722001339	M12 St-0°/ M12 Bu-90° 30m

Cable-ID	True Length (m)	Cable-ArtNr	Cable-Name (short)
UH01-MA0062	10,3015	725000015	4G1,5mm², Steuerleitung

Cables SIVAS:
zeigt die SIVAS-Nummer der Kabel und daneben die benötigte Anzahl an Kabeln in diesem Layout bzw. (für den Fall von Kabeln als Meterware, z.b. bei Motoren) die kummulierte benötigte Länge:

Cable-ArtNr	Amount (pcs)	Cumm. Length (m)
722001332	1	
722001334	1	
722001336	1	
722001339	1	
725000015		11

Cables SIVAS:
zeigt die Längen der einzelnen Kabeltrassen-Segmente anhand deren Nummerierung innerhalb der dxf Datei. Diese Ausgabe dient der Überprüfung der benötigten Gesamtlänge der Kabeltrassen.

Erweiterte Ausgabe im Fehlerfall:

Für den Fall, dass während des Programmablaufs Fehler auftreten bzw. Fehler im Layout erkannt werden, werden diese in weitere Worksheets geschrieben. Diese Worksheets tragen das Präfix *ERR*

- ERR-Equipment-Connection:
  - Zeigt alle nicht mit den Kabeltrassen verbundenen Elemente (Sensoren / Aktoren / Unterverteiler) an
  - Gibt neben dem Typen, den Bezeichner und die x, y-Koordinate des Elements zurück
  - Verbindungsfehler möglicherweise aufgrund zu großen Abstandes zur nächsten Kabeltrasse
- ERR-Routing:
  - Zeigt alle fehlgeschlagenen Kabelverbindungen an
  - Fallunterscheidung:
    - Unterverteiler nicht in Layout vorhanden: Unterverteiler ist in der Kennzeichnung der Sensor-Blöcke im Layout erwähnt aber nicht im Layout selbst positioniert. Es wird nur der betroffene Unterverteiler angezeigt. Keine explizite Auflistung aller damit ebenfalls nicht verbundenen Sensoren / Aktoren
    - Unterverteiler **und** Sensor / Aktor nicht angebunden: Sowohl Sensor als auch Aktor sind nicht mit den Kabeltrassen verbunden. Beide damit auch in *ERR-Equipment-Conncetion* aufgeführt
    - Unterverteiler nicht angebunden: siehe oben.
    - Sensor / Aktor nicht angebunden siehe oben.
- ERR-Attributes:
  - Zeigt alle Blöcke an, welche Fehler in deren Attributen aufweisen
  - Mögliche Fehler:

- Kein Attribut "KENNZEICHNUNG": keine Zuweisung zu Unterverteiler möglich. Element wird nicht verkabelt und nicht gezeichnet.
- Ungültiger Pfad in "KENNZEICHNUNG": Eingegebenes Attribut in Kennzeichnung entspricht nicht dem normalen Muster. Element wird nicht verkabelt und nicht gezeichnet.

Excel-Datei: *dxfdatei*_BOM.xlsx

Diese Datei enthält eine vollständige **Stückliste (BOM)** aller im Layout erkannten Komponenten – also sowohl aller Sensoren/Aktoren als auch der Kabel, die zum Einsatz kommen. Die Datei besteht aus zwei Tabellenblättern:

1. BOM (Gesamtstückliste):

Zeigt für jede verwendete Artikelnummer den Namen laut bezeichner.cfg, die Gesamtanzahl bzw. Gesamtlänge und unterscheidet dabei automatisch zwischen Sensoren und Kabeln.

Art.-Number	Amount (pcs)	Length (m)	Name (SIVAS)
722001330	3		M12 Sensorleitung 1 m, gewinkelt
725000015		24	Steuerleitung MA, 4G1,5mm²
839220147	5		Reflexionslichttaster XYZ

Sensoren (bzw. andere Komponenten mit Artikelnr) werden über ihre Artikelnummer gezählt. Kabel werden – sofern Meterware (z.B. Steuerleitungen für Motoren) – über die summierte Länge dargestellt, andernfalls über Stückzahl.

2. BOM by UV (Stückliste pro Unterverteiler):

Dieses Worksheet gibt für jede Artikelnummer aus, **welchem Unterverteiler (UV)** sie zugeordnet ist und wie viele davon jeweils dorthin gehören. Kabelmeter werden wie im anderen Arbeitsblatt getrennt ausgewiesen.

UV	Art.-Number	Amount (pcs)	Length (m)	Name (SIVAS)
UH01	722001330	1		M12 Sensorleitung 1 m, gewinkelt
UC01	722001330	2		M12 Sensorleitung 1 m, gewinkelt
UC01	725000015		12	Steuerleitung MA, 4G1,5mm²

Hinweis: Diese Ansicht ist besonders nützlich zur **vorbereitenden Bestellplanung pro UV**.

Anpassung des Verhaltens des Programms:

Verwendete Konfigurationsdateien

Zum aktuellen Zeitpunkt verwendet bzw. berücksichtigt das Programm **vier** Konfigurationsdateien (Dateiendung .cfg). Diese steuern das verhalten des Programms und **können vom Nutzer bei Bedarf geändert** werden bzw. **müssen vom Nutzer im Falle von betrieblichen Ändeurngen aktualisiert werden** (z.B Änderung von Sachnummern), um die Funktion des Programms zu gewährleisten. In der folgenden

Tabelle sind die Aufgaben bzw. inhalte der einzelnen Konfigurationsdateien aufgezeigt und in den darauffolgenden Abschnitten der jeweilige Aufbau und besonderheiten erläutert.

Datei	Inhalt
`allgemein.cfg`	Layernamen der Kabelpritschen Layenamen der Unterverteiler Layernamen der Tunnel Layernamen der Sensoren / Aktoren Toleranz der Verbindung zw. Sensor und Kabeltrasse Toleranz des "Anpinnens" zweier Kabeltrassen untereinander
`BMK.cfg`	Betriebsmittelkennzeichnung der Elemente, die im Routing berücksichtigt bzw. ignoriert werden sollen Zuweisung der Kabelkennzeichnungen zur jeweiligen Betriebsmittelkennzeichnung Längenanpassungen für einzelne Kabelkennzeichnungen
`kabel.cfg`	SIVAS-Nummern für die einzelnen längenabhängigen Kabel der jeweiligen Kabelkennzeichnung
`bezeichner.cfg`	Speicherung der Bezeichner zu den SIVAS-Nummern sowie automatische Pflege fehlender Artikelnummern

❗ Hinweis ❗

Änderungen an den .cfg-Dateien werden erst beim nächsten Programmlauf wirksam. Stelle sicher, dass die Datei korrekt gespeichert wurde und keine doppelten Abschnitte enthält.

Zusammenfassung und Struktur der Konfigurationsdateien

Nachfolgende Aufzählung zeigt zeigt den Aufbau und Zweck der `.cfg`-Dateien für das Routing-Tool. Jeder Abschnitt ist dokumentiert mit:

- Konfigurations-Block (Abschnittsname)
- Beschreibung der Funktion
- Beispielhafte Einträge

allgemein.cfg

Diese Datei steuert das Einlesen von Layern und definiert geometrische Toleranzen.

`[GetPos-Layer_Racks]`

Was? Layernamen, in denen Kabelpritschen (Racks) enthalten sind.

Layername (Beispiel)
PRITSCHE_200-60_OMNIFLO
0-0_ILS_Pritsche_200-60_Inbound

Layername (Beispiel)
0-0_Omniflo_Pritsche_200-60_AMR

`[GetPos-Layer_Distributors]`

Was? Layernamen, die Unterverteiler beinhalten.

Layername (Beispiel)
0-0_ILS_UNTERVERTEILER
UNTERVERTEILER

`[GetPos-Layer_Tunnel]`

Was? Layernamen für Tunnel.

Layername (Beispiel)
Busverteiler-Kennzeichnung

`[GetPos-Layer_Equipment]`

Was? Layernamen von Sensoren, Motoren, IOs etc.

Layername (Beispiel)
0-0_ILS_MOTOR
MOTOR
REALE_POSITION_IO

`[GetPos-Geom-Sensor]`

Was? Geometrische Maße von Sensor-Markern zur Mittelpunktsberechnung.

Attribut	Wert (Beispiel)
Breite	80
Hoehe	90

`[Racks]`

Was? Maximale Distanz (Toleranz) zwischen benachbarten Racks zum "Snappen".

Attribut	Wert (Beispiel)

Attribut	Wert (Beispiel)
SnapTolerances	200

`[Sensoren]`

Was? Maximale Distanz zwischen Sensoren/Motoren und Racks für Verknüpfung.

Attribut	Wert (Beispiel)
ConnectionTolerances	3000

BMK.cfg

Diese Datei steuert, **welche Betriebsmittelkennzeichen (BMK)** beim Routing berücksichtigt werden und welche **Kabeltypen** zugewiesen sind.

`[Routing-Include]`

Was? BMK-Kürzel, die beim Routing **berücksichtigt** werden.

Kürzel	Bedeutung / Beispiel
MA	Motoren
QM	Ventile
BX	Scanner

`[Routing-Ignore]`

Was? BMKs, die ignoriert werden (z.B. Schaltschrank-Innenleben).

Kürzel	Beispielkomponente
FC	Frequenzumrichter
QA	Sicherungen / Schütze

`[Cable-Mapping]`

Was? Verknüpfung von BMKs zu Kabelgruppen (aus `kabel.cfg`).

BMK	Zugewiesene Kabelgruppe(n)
MA	MA
MB	WD_Q
BX	WF_B, WD_I

BMK	Zugewiesene Kabelgruppe(n)
BG-829422026	WD_I-829422026

`[Length-Adjustments]`

Was? Korrektur von Kabellängen (z.B. vorkonfektionierte Sensorleitungen).

BMK	Abzuziehende Länge (m)
BX	4

---

kabel.cfg

Diese Datei enthält die **SIVAS-Nummern je Kabellänge und -typ**, sowie die **Bezeichnungen** der SIVAS-Nummern.

Kabelgruppen (Beispiel: `[WD_Q]`, `[WD_I]`, `[WF_B]`)

Was? Zuordnung von Kabellängen zu SIVAS-Sachnummern.

Länge (m)	SIVAS-Nr.	Beispielgruppe
1.0	722001300	WD_Q (Ventile)
2.5	722001338	WD_I (Sensoren)
5.0	726001045	WF_B (Scanner)

Sensorspezifische Gruppen

Was? Besondere Kabellisten für spezifische Sensor-Artikelnummern.

Gruppe	Länge (m)	SIVAS-Nr.
WD_I-829422026	5.0	722001353
WD_I-720002003	10.0	722001354

---

bezeichner.cfg

Diese Datei enthält Bezeichner zu den verwendeten SIVAS-Artikelnummern. Sie beinhaltet sowohl die Bezeichner (laut SIVAS) für die Bauteil-Artikelnummern als auch für die Kabel. Die Datei wird im Laufe eines Routing-Prozesses bei Bedarf angepasst und mittels einer SIVAS-Datenbankabfage (bei bestehender VPN-Verbindung bzw. im lokalen Firmennetz am Standort) erweitert.

`[Sivasnummern]`

Enthält die bekannten SIVAS-Artikelnummern mit zugehörigen Bezeichnern, z.B.:

Sivasnummer	Bezeichner
725000015	4G1,5mm², Steuerleitung
722001300	Verb.-Leitung M12 St-0°/ M8 Bu-0° PUR UL/CSA 1m
722001301	Verb.-Leitung M12 St-0°/ M8 Bu-0° PUR UL/CSA 2m

`[Missing]`

Fehlende Nummern, die im Layout verwendet wurden, aber noch keinen Bezeichner besitzen. Diese werden automatisch per SIVAS-Datenbank ergänzt (sofern erreichbar). Hierzu wird im Kommandofeld eine entsprechende Benachrichtigung angezeigt.

! Hinweis !

Die Einträge in den `.cfg`-Dateien müssen **laufend gepflegt** werden, wenn sich:

- Layernamen in CAD-Dateien ändern
- Neue BMK-Kürzel eingeführt werden
- Neue SIVAS-Artikel hinzukommen

Nur so kann eine korrekte automatische Kabelberechnung und Zuordnung gewährleistet werden.

Anwenderhinweise zur Layouterstellung

Im nachfolgenden Abschnitt sind die **wichtigsten** Punkte zur Erstellung der Layoutzeichnungen aufgeführt. Die Beachtung dieser Hinweise stellt sicher, dass die automatische Auswertung durch das Programm fehlerfrei und zuverlässig erfolgen kann.

! Hinweis !

Dieser Abschnitt ist durch die Fachabteilung regelmäßig zu pflegen und analog zu Konfigurationsdateien stets aktuell zu halten.

Verwendung der Layer

- **Erfasst werden ausschließlich Elemente auf Layern**, die in der Konfigurationsdatei (`cfg`) unter dem jeweiligen Abschnitt (`GetPos-Layer_...`) aufgeführt sind.
- Nicht konfigurierte Layer oder falsch benannte Layer werden vollständig ignoriert.
- Für jede Objektklasse (z. B. Racks, Sensoren, Unterverteiler) sollte ein eigener dedizierter Layer verwendet werden.
- Layer-Namen dürfen **keine Sonderzeichen oder Leerzeichen** enthalten, und sollten **eindeutig** benannt sein.

Zeichnen von Kabeltrassen

- **Verbindungspunkte von Racks sollten präzise aneinander gezeichnet werden**, idealerweise mit Fangmodi (Snapping) im CAD.

    - Das Programm erkennt und verbindet nahe beieinanderliegende Racks automatisch, jedoch nur mit begrenzter Toleranz.
    - Eine saubere Planung der Trassengeometrie erleichtert die automatische Graphenerstellung erheblich.
- **2D-Kabeltrassen (LWPOLYLINE)** sollten verwendet werden für Strecken in einer Ebene bzw. auf einer Höhe
- **3D-Trassenführung** bei Höhendifferenzen (z.B entlang von Förderern) erfolgt in drei Schritten:
    1. Untere Ebene als `LWPOLYLINE` mit Z=0.
    2. Obere Ebene als `LWPOLYLINE` mit Z=Erhebung (über DXF-Attribut `Erhebung`).
    3. Verbindung über `3DPOLYLINE`, welche die Steigung darstellt.

Attribute der Blöcke (Bauteile)

- **Pro Bauteil darf nur ein Block vorhanden sein.** Alle relevanten Informationen (z. B. Artikelnummer, Bezeichner, Position) müssen diesem einen Block als Attribute zugeordnet sein.
- **Keine übereinanderliegenden Blöcke!**
    - In der Vergangenheit wurden häufig zwei Blöcke übereinandergelegt, um Kabel- und Sensorinformationen zu trennen. Dies ist künftig nicht mehr zulässig.
- **Beispiel für frühere Aufteilung (nicht mehr erlaubt):**
    - *Rahmen innen* (Angaben zum Kabel):
        - A:
        - B: Kabel-ID
        - C: lange Nummer (?)
        - ArtiNr: Kabelartikelnummer
        - Beschr: Kabelname (SIVAS)
        - Menge, Position, Gruppe, Etikette, Auflöse, etc.
    - *Rahmen außen* (Angaben zum Bauteil selbst):
        - A:
        - B: Bauteil-Kurzbezeichnung
        - C: lange Nummer (?)
        - ArtiNr: Sensorartikelnummer
        - Beschr: Kurzbeschreibung
        - sowie gleiche Attribute wie oben.
    - *Text*
        - IO: Bauteil-Kurzbezeichnung
        - id
        - VERW
        - BEZEICHNUNG
        - TEXT-D, TEXT-E, TEXT-ES, TEXT-F
        - SPS
- **In Zukunft**:
    - Ein einziger Block enthält alle Informationen des Bauteils.
    - Die Informationen zu den benötigten Kabeln werden **allein** vom Kabeltool bereitgestellt
    - Die konkrete Struktur der Attribute wird noch festgelegt (To Be Defined).
    - Ziel ist es, sowohl die Verarbeitung im System als auch die manuelle Pflege im Layout zu vereinfachen.

Positionierung von Bauteilen

- In Zukunft wird das Attribut `REALE_POSITION` eingeführt. Damit kann die **exakte physische Einbauposition** eines Bauteils (z. B. Sensors) unabhängig von der tatsächlichen Platzierung des Blocks im Layout definiert werden.
    - Ermöglicht eine saubere, leserliche Platzierung der Blöcke im Plan – z. B. seitlich der Anlage (dort wo Platz ist)
    - Der Block selbst kann damit **optisch gut positioniert**, aber technisch korrekt ausgewertet werden.
- Bauteile ohne `REALE_POSITION` werden standardmäßig an der **gezeichneten Blockposition** verortet.
- **Anbindung an Kabeltrassen erfolgt immer zur nächstgelegenen Rack-Geometrie** im definierten Toleranzbereich.
    - Wird ein Block zu weit entfernt vom Rack gezeichnet (bzw. `REALE_POSITION` liegt zu weit weg), erfolgt **keine automatische Verbindung**.
    - Die maximale Toleranz ist über die Konfiguration `allgemein.cfg[tol_connect]` steuerbar.
- Es ist daher darauf zu achten, dass Bauteile (bzw. deren reale Positionen) **räumlich korrekt und in sinnvoller Nähe** zu den Kabeltrassen platziert sind.

Detaillierte Informationen zum Programmablauf und Code-Struktur

Zweck des Programms

Dieses Toolset dient der **automatisierten** und **komfortablen** Ermittlung von Kabellängen zwischen Unterverteilern und deren zugeordneten Sensoren/Aktoren entlang definierter Kabeltrassen. Als **Eingabe** dient eine 2D-Layout-Zeichnung im `.dxf`-Format. **Ausgaben** sind u.a.:

- eine strukturierte `.json`-Datei mit Kabellängen und Pfadkoordinaten
- eine visuelle `.dxf`-Datei mit eingezeichneten Kabelwegen
- eine weitere .dxf-Datei mit einem reduzierten Layout (Kabeltrassen + Kabel + Unterverteiler)
- zukünftig: tabellarische Aufbereitung der Kabellängen mit SIVAS-Nummern (z. B. `.xlsx`)

Allgemeiner Programmablauf

Das Toolset besteht aus drei zentralen Skripten, die automatisiert über **ein** *Batch-Skript* aufgerufen werden. Der interne Programmablauf ist folgender:

`getpositions.py` -> `routing.py` -> `drawdxf.py`

Das erste Programm extrahiert aus der .dxf-Datei sämtliche Informationen über die Positionen von Sensoren / Aktoren und Unterverteilern sowie den Kabeltrassen. Das zweite Programm baut aus den Informationen ein Modell der Anlage auf und verknüpft die Elemente entlang der kürzesten Wege. Das letzte Programm zeichnet eine neue .dxf-Datei, in welcher die Kabelwege von den Unterverteilern zu den Sensoren dargestellt sind.

Aufruf des Programms

Der einfachste Weg, das Toolset zu starten, ist per *Drag-and-Drop*. Hierfür muss lediglich die .dxf-Datei, für die die Kabelführung und -auswertung durchgeführt werden soll, auf die Verknüpfung namens *Kabeltool* gezogen werden. Es öffnet sich eine sogenannte *Command-Shell*, die den Benutzer über den aktuellen Zustand des Programmablaufs informiert.

Die Ausgabe schreibt:

```
C:\10-Develop\kabellaengen\bin>getexdraw.bat easy.dxf
--hole Positionen
reading file ..done
writing results to a json file ...
done
--erzeuge Graph mit Routing
writing results to a json file ...
done
--zeichne Kabel in dxf Datei
done
```

Ausgabe des Toolsets

Alle drei oben genannten Teilprogramme erzeugen eine eigene Ausgabedatei. Die Ergebnisse der ersten beiden Programme dienen als Eingabe für die folgenden. Jedes der genutzten Programme besitzt eine eingebaute Hilfe, wenn man das Programm mit dem Schalter --help aufruft.

Die Ausgaben des letzten Teilprogrammes `drawdxf.py` sind für den Anwender bestimmt. Zu Informationszwecken können die Zwischenergebnisse im "Work"-Ordner geöffnet werden (genauere Informationen zu den Ausgaben in den jeweiligen Abschnitten zu den Teilprogrammen). Als wesentliche Ausgabe nach Aufruf des Gesamtprogramms dienen:

- Die .dxf-Datei mit den Kabelwegen: `NamederEingabedatei_cables.dxf`
- Die tabellarische Aufbereitung der Kabellängen mit zugehörigen Sachnummern für jeden Sensor / Aktor: `NamederEingabedatei_cables.xlsx`

Informationen zu den Einzelprogrammen

Nachfolgend sind Infomationen zu den Einzelprogrammen aufgeführt. Diese enthalten teils wichtige Hinweise zur Aufbereitung der Eingabedateien, Informationen zu den einzelnen Ausgabedateien sowie immer ein Beispiel anhand eines einfachen Layouts `easy.dxf`. Dieses wird verwendet, um den Programmablauf

beispielhaft zu zeigen. Nachfolgend ein Screenshot des Layouts für die weiteren Beispiele.

Details zu `getpositions.py`

Hier die vorhandenen Schalter des Programms:

```
usage: getpositions [-h] -f myfile.dxf [-s] [-r] [-w WRITE] [-c] [-e ERRORS] [-n]

fetches the x/y positions from a dxf file
```

Schalter:

Schalter (kurz)	Schalter (lang)	Argument (notwendig!)	Hilfe
`-h`	`--help`		show this help message and exit
`-f`	`--filename`	`myfile.dxf`	file to be analyzed
`-s`	`--sensors`		fetch all positions of sensors, actors and subdistributors
`-r`	`--rack`		fetch all positions of all cable racks
`-w`	`--write`	`myfile_positions.json`	write results into a json file
`-c`	`--console`		print results to output
`-e`	`--errors`	`myfile_errors.json`	write an error file in case of double defined items in layout
`-n`	`--scan`		print all layer of racks, distributors and equipment not empty

Das erste Programm im Ablauf bestimmt maßgeblich die Ausgaben der weiteren Programme und ist weiterhin maßgeblich von der Eingabedatei (.dxf-Datei der Anlage) abhängig.

Eine .dxf-Datei (mit standardisierten Merkmalen!) wird in `getpositions.py` eingelesen. Das Programm extrahiert aus der 2D-Zeichnungsdatei alle Anfangs- und Endpunkte der Kabeltrassen sowie alle Positionen der Sensoren / Aktoren und Unterverteiler innerhalb des 2D-Layouts der Anlage. Diese werden gesammelt und in einer temporären Datei abgespeichet (`NamederEingabedatei.json`).
Bei der Erstellung der .dxf-Datei, welche vearbeitet werden soll, ist es wichtig, dass die Kabeltrassen konsistent auf den gleichen Layern gezeichnet werden und bei der Erstellung / Positionierung der Sensoren und Aktoren folgendes beachtet wird:

- Sensoren / Aktoren müssen als Block in der dxf Datei definiert sein und das Attribut "REALE_POSITION=x" enthalten. Das x dieses Attributs kann dann unabhängig von der Beschreibung im Kasten verschoben werden, so dass man dieses auf der echten Sensorposition verschieben kann. Das Kabel wird dann vom Unterverteiler bis zur Position des x erzeugt.
- Sensoren / Aktoren, die keine solche Markierung enthalten, werden auf die Mitte des Textblockes mit einem Kabel versehen

Die Ausgabe des Programms ist eine .json-Datei, welche **strukturiert, in Textform** die Informationen aus der Zeichnungsdatei weitergibt. Nachstehend ein Auszug der .json-Ausgabe des oben gezeigten Layouts.

```
{
  "sensors": {
    "BG3241": {
      "IO": "BG3241",
      "ID": "",
      "VERW": "Jam detector",
```

```
      "BEZEICHNUNG": "Stausensor 1 (ILS-CV M0108)",
      "KENNZEICHNUNG": "=A01+UC0101-KF1DI1",
      "SPS": "1",
      "REALE_POSITION": "x",
      "pos": [38.8, 1280.2]
    },
    "MA0062": {
      "IO": "MA0062",
      "ID": "",
      "VERW": "CV-M0062_0,75",
      "BEZEICHNUNG": "Motor MA0062",
      "KENNZEICHNUNG": "=A01+UH01-KF1DQ04",
      "TEXT-E": "Motor MA0062",
      "SPS": "1",
      "REALE_POSITION": "x",
      "pos": [4967.0, 8072.5]
    },
  },
    ....

  "distributors": {
    "UC0101": [0.0, 4162.8]
  },
  "racks": {
    "Rack_1": [
      [4946.5, 15774.4],
      [4946.5, 3879.4]
    ],
    "Rack_2": [
      [0.1, 57.6],
      [0.1, 3777.6],
      [14755.1, 3777.6]
    ],
    "Rack_3": [
      [185.1, 15865.5],
      [12450.7, 15865.5]
    ]
  },
  "mapping": {
    "UC0101": [
      "BG3241",
      "MA0062"
    ]
    }
}
```


Die Ausgabedatei ist gegliedert in vier große Blöcke:

- **Sensoren und Aktoren** ("sensors"): Enthält alle Sensoren und Aktoren der Anlage mit allen zugewiesenen Informationen.
- **Unterverteiler** ("distributors"): Enthält die Positions-Infomation der Unterverteiler

- **Kabeltrassen** ("racks"): Enthält die Koordinaten der Anfangs und Endpunkte einer Kabeltrasse. Wenn Kabeltrasse als *Polylinie* gezeichnet ist, sind mehrere einzelsegmente aufgeführt (siehe bspw. "Rack_2")
- **Zuweisung von UV zu Sensoren / Aktoren** ("mapping"): Enthält die Zuweisung von einem Unterverteiler zu allen daran angeschlossenen Sensoren und Aktoren

Details zu `routing.py`

Hier die vorhandenen Schalter des Programms:

```
usage: routing.py [-h] -f my_positions.json [-c] [-g] [-w WRITE]

Berechne Wege von Sensoren zu Verteilern über Kabeltrassen
```

Schalter:

Schalter (kurz)	Schalter (lang)	Argument (notwendig!)	Hilfe
`-h`	`--help`		show this help message and exit
`-f`	`--filename`	`myfile_positions.json`	file positional information
`-c`	`--console`		print result to output
`-g`	`--graph`		draw graph and display in separate window
`-w`	`--write`	`myfile_todraw.json`	write results-file to use for cable-drawing

Die .json-Ausgabe, welche oben in einem Auszug gezeigt ist, stellt die Daten der .dxf-Datei in einer maschinen- und menschenlesbarer Fassung dar. Die ausgegebenen Daten sind jedoch weiterhin im weitesten Sinne "Rohdaten", welche noch weiter behandelt werden müssen. In dem Einzelprogramm `routing.py` wird aus den Daten mithilfe der Funktionen, welche in `plant.py` implementiert sind, eine "virtuelle" Anlage (eng.: *plant*) aufgebaut. Als Eingabe dient lediglich die .json, welche von `getpositions.py` ausgegeben wird. Hauptfunktionen von `routing.py` bzw. `plant.py` sind:

- Aufbauen des "Grundgerüstes" der Anlage bestehend aus Kabeltrassen (Racks)
  - Finden von Schnittpunkten einzelner Racks und Erstellung von expliziten Knoten dort
  - Finden von "eigentlichen" Schnittpunkten und Anpinnen von nahezu verbundenen Racks aneinander
- Verknüpfen von Sensoren / Aktoren / Unterverteilern mit dem Grundgerüst der Anlage
  - Finden des nächstgelegenen Racks zu jedem **S**ensor / **A**ktor / **U**nter**V**erteiler
  - Erstellen eines Aufpunktes für die Strecke vom Rack zum S / A / UV
  - Verknüpfen des jeweiligen S / A / UV mit dem Aufpunkt über den kürzesten Weg
- Erstellen eines Graphen (=mathematisches Modell für netzartige Struktur) zur Anwendung von Wegfindungs-Algorithmen zur Bestimmung der kürzesten Kabelwege von Sensor / Aktor zu Unterverteiler entlang der Racks

Die Ausgabe des Einzelprogramms `routing.py` ist im wesentlichen die Infomationen über die Kabel, welche von einem Sensor / Aktor zu dem zugehörigen Unterverteiler laufen. Die Informationen beinhalten den Pfad

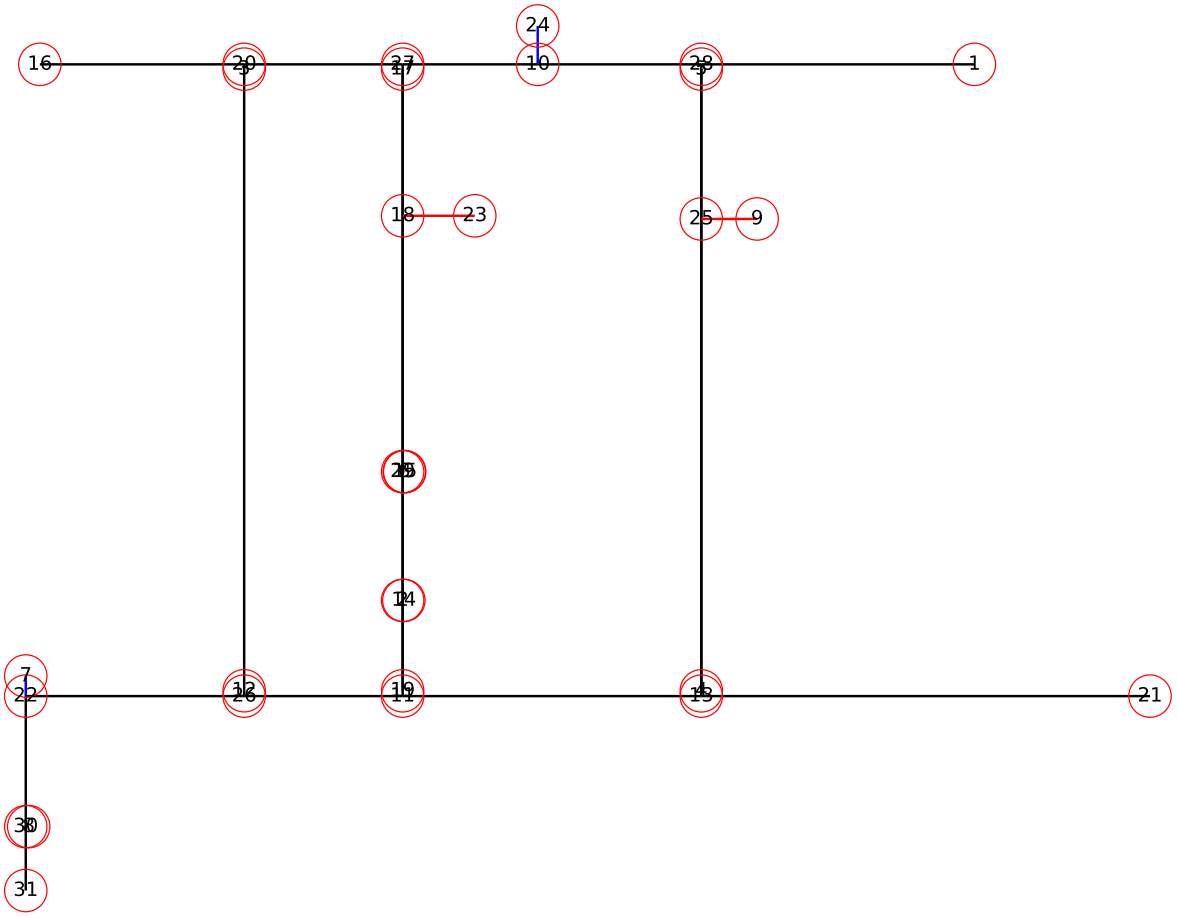
(= die einzelnen Koordinaten in x,y über welche der Pfad verläuft) und die Länge des Pfades (und damit die Länge des Kabels). Die Ausgabe erfolgt erneut in einer strukturierten .json-Datei. Die Ausgabedatei an dieser Stelle trägt stets den namen `todraw.json`. Sie dient als Eingabe für das Einzelprogramm `drawdxf.py`. Nachfolgend ein Ausschnitt der Ausgabedatei anhand des oben gezeigten Beispiels:


```
{
  "kabel": [
    {
      "id": "UC0101-BG3241",
      "coords": [
        {"x": 0.0, "y": 4162.8},
        {"x": 0.1, "y": 3777.6},
        {"x": 0.1, "y": 1280.2},
        {"x": 38.8, "y": 1280.2}],
      "length": 2921.3,
      "nodes": [31, 17, 18, 12]
    },

    ....

    {
      "id": "UC0101-BG3240",
      "coords": [
        {"x": 0.0, "y": 4162.8},
        {"x": 0.1, "y": 3777.6},
        {"x": 2866.6, "y": 3777.6},
        {"x": 4946.5, "y": 3777.6},
        {"x": 4946.5, "y": 3879.4},
        {"x": 4946.5, "y": 5609.0},
        {"x": 4961.9, "y": 5609.0}],
      "length": 7178.4,
      "nodes": [31,17,15,29,21,7,14]
      ]
    }
```


Die Ausgabedatei gliedert sich in die Blöcke, welche im Mapping (siehe Ausgabedatei unter `getpositions.py`) aufgeführt sind. Im obigen Beispiel sind demnach die Kabel von Unterverteiler *UC0101* zu Sensor *BG2341* bzw. von *UC0101* zu *BG2340* dargestellt. Neben der ID des Kabels ist aufgeführt:

- Die Knotenpunkt-Koordinaten entlang welchen das Kabel verläuft
- Die exakte Länge des Kabels
- Die Knotenpunkte (nodes) des Graphen, entlang welchen das Kabel verläuft

Als weitere Ausgabedatei kann in der Konfigurationsdatei `routing.cfg` die Ausgabe des Graphen eingestellt werden. Der Graph, welcher sich anhand des verwendeten Beispiels ergibt ist nachfolgend gezeigt. Die Knotenpunkte können zur einfachen Überprüfung der Kabelwege anhand des Graphen verwendet werden (ohne die Verwendung der teils unhandlichen Koordinaten x,y). Der Graph wird standardmäßig als Vektorgrafik (.svg) gespeichert, sodass ohne Qualitätsverlust an sich überlappende Knoten gezoomt werden kann, um diese genauer zu betrachten.



- Schwarze Linine stellen Racks dar
- Rote Linine sind Verbindungen zwischen Racks und Sensoren / Aktoren
- Blaue Linien sind Verbindungen von Unterverteilern zu Racks

Details zu `drawdxf.py`

Hier die vorhandenen Schalter des Programms:

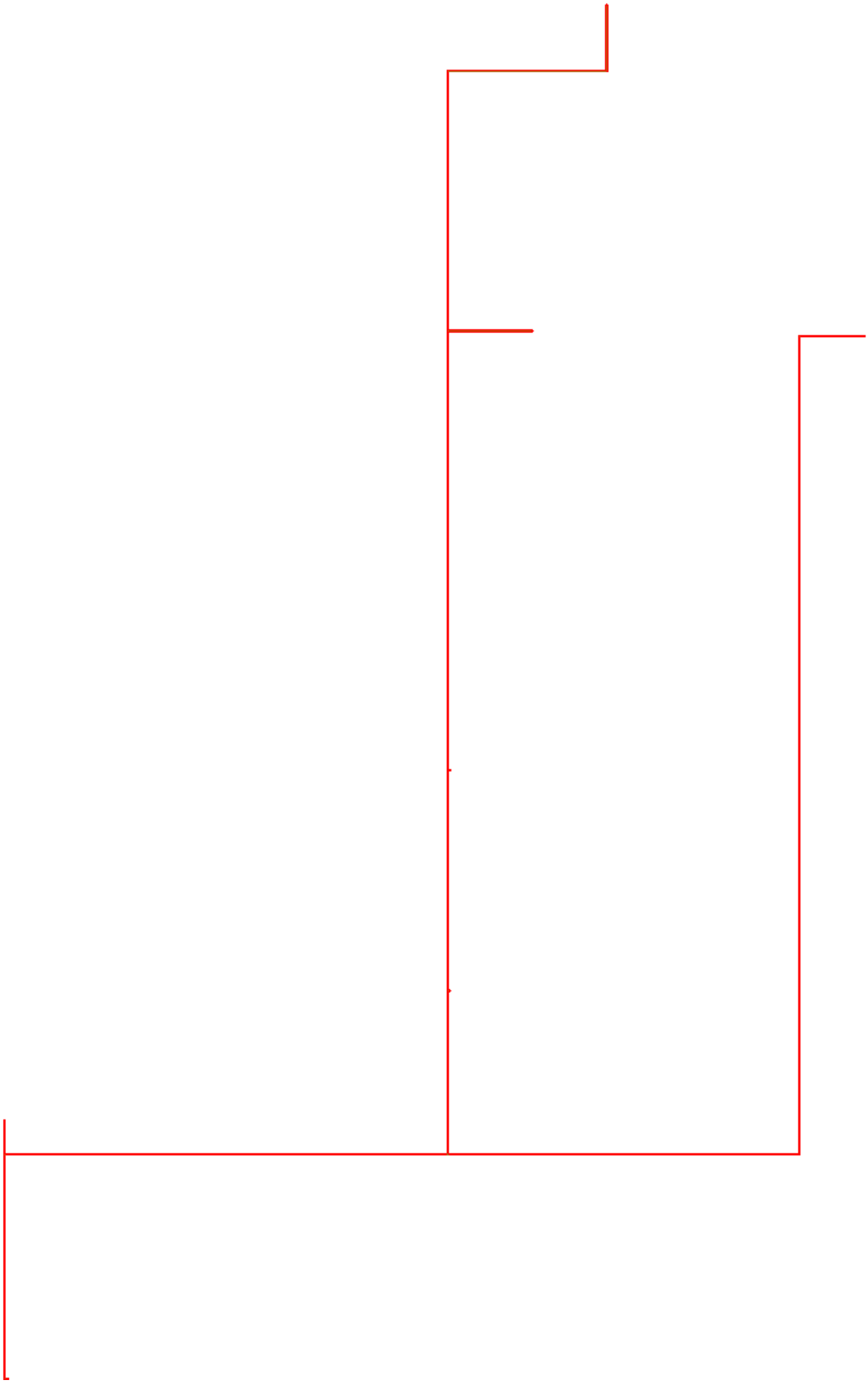
```
usage: drawdxf [-h] -f myfile_todraw.json [-d myfile.dxf] [-n NEW] [-x
myfile_cables.xlsx] [-o myfile.dxf] [-l]


draws a dxf file with the given cable coordinates
```

Schalter:

Schalter (kurz)	Schalter (lang)	Argument (notwendig!)	Hilfe
`-f`	`--filename`	`myfile_todraw.json`	this json file contains all cables and its coordinates which should be drawn. Saved with a unique timestamp

Schalter (kurz)	Schalter (lang)	Argument (notwendig!)	Hilfe
`-d`	`--dxf`	`myfile.dxf`	this dxf drawing will be copied and the new layer with the cables will be added. Requires `--origin`
`-n`	`--new`	`myfile_cables.dxf`	create a new dxf file only with cables in it. Name is basename and a timestamp
`-x`	`--excel`	`myfile_cables.xlsx`	create a xlsx file with cables data
`-o`	`--origin`	`myfile.dxf`	name of original .dxf file used by `--dxf`
`-l`	`--local`		using only local data for naming of article numbers. If not set: fetching names from SIVAS

Das letzte Einzelprogramm, welches in der Routine aufgerufen wird, dient der Erstellung einer eigenen .dxf-Datei, welche die Kabelwege dastellt. Diese .dxf Datei trägt stets den Namen der Eingabedatei mit der Ergänzung `..._cables.dxf`. Die Datei kann als neue Layer in das bestehende Anlagen-Layout importiert werden, um die Kabelwege zu verifizieren. Die sich aus dem behandelten Beispiel ergebende Datei ist nachfolgend alleine sowie importiert in das Layout dargestellt:



Die in den Details zu `getpositions.py` beschriebene Handhabung der Verbindung der Sensoren / Aktoren wird ersichtlich:

- Sensoren mit einem Marker "x" zur genauen Positionierung werden an diesen verbunden (BG3260, BG3240, ...)
- Sensoren, welche über keinen Marker verfügen, werden anhand des Textblocks verbunden (z.B: BG3270)
- Die Unterverteiler werden stets auf die Mitte ihres Blocks verbunden.

Das letzte Programm gibt standardmäßig zudem eine Excel-Übersicht der Kabel zurück. Diese beinhaltet die jeweilige ID des Kabels ("von wo nach wo") und die tatsächliche (genaue) Länge des Kabels. Daneben ist die gerundete Länge (aktuell auf volle 10 m) aufgeführt. In einem weiteren Arbeitsblatt sind die Längen gesammelt aufgeführt (z.B. 10x 10m Kabel, 5x 20 m Kabel, 1x 50 m Kabel). In einer Fehlerübersicht sind fehlgeschlagene Verbindungen zwischen Sensoren / Aktoren / Unterverteilern zu Kabeltrassen aufgeführt. In einer weiteren Übersicht fehlgeschlagene Kabelverbindungen (*Routings*) und der Grund für das Fehlschlagen.

Häufige Fagen

Wo stelle ich ein, was das Toolset am Ende ausspuckt?

Hier müssen wir das mit der einzelnen Config noch implementieren

Warum werden manche Kabeltrassen nicht mit anderen verbunden?

Mittels der Methode `join_racks()`, welche innerhalb `plant.py` definiert ist und von `routing.py` aufgerufen wird, werden die Rack-Segmente, welche von `getpositions.py` übergeben werden, auf echte Schnittpunkte, sowie beinahe Schnittpunkte überprüft und an den jeweiligen Punkten verknüpft. Beinahe Schnittpunkte werden durch Entlanglaufen der einzelnen Racks und "Absuchen" eines Toleranzfeldes ermitteln.

Sobald ein Schnittpunkt des Toleranzkreises mit einem angrenzenden Rack festgestellt wird, wird dieses Rack mit dem ausgehenden Rack verbunden.

Sollten einzelne Kabeltrassen nicht miteinander verbunden werden, ist die eingestellte Toleranz in der Konfigurationsdatei sowie der tatsächliche Abstand des Endpunktes einer Kabeltrasse und der naheliegenden

zu überprüfen. Die Toleranz muss stets größer sein als dieser Abstand. Bei zu groß gewählter Toleranz kann unerwünschtes Anpinnen von zu weit entfernten Kabeltrassen auftreten. Der Standardwert ist auf **200** zu setzen.

Warum verbindet das Programm den Sensor mit genau dieser Kabeltrasse?

Bei der Verbindung von Sensoren / Aktoren mit den Kabeltrassen findet noch keine Wegsuche zum zugehörigen Unterverteiler statt. Die Sensoren / Aktoren tragen darüber hinaus keinerlei Information, mit welcher Kabeltrasse sie verbunden werden sollen. Der Algorithmus sucht daher ausgehend von der Position des Sensors das am nahesten liegende Rack und stellt im Anschluss die kürzestmögliche verbindung zu einer Kabeltrasser her.

Situation: Horizontales und vetikales Rack (R1 bzw R2), Sensor (S). Erstellen eines Kreises um den Sensor und Überprüfung auf Schnittpunkte mit Kabeltrassen

Schritt 2: Bei Ausbleiben eines Schnittpunktes -> Vergrößern des Kreises. Bei Ermittlung eines Schnittpunktes -> Erzeugen eines Aufpunktes auf der geschnittenen Kabeltrasse und Verbinden von Sensor mit Kabeltrasse.

Ideen für "Umbau" der Programme

- Eine Config in der unter mehreren Blöcken ## Routing ##, ##Ausgabe## die einzelnen Schalter 1 / 0 gesetzt werden
- Getpositions läuft immer
  - -> scheibt output in positions.json in work
  - -> damit werden alte positions.json überschrieben und work ordner nicht zugemüllt
- Routing läuft immer -> schreibt output in routing.json
  - wenn schalter in Config für Graph gesetzt ist wird Graph als .svg in work gespeichert
    - Name der Datei *eingabefile*_graph.svg
- Drawdxf umbennen in sowas wie *cables.py* oder so

- Schalter steuern was alles ausgegeben wird (Excel, dxf, ... was könnte man noch machen)
- hier muss man entscheiden ob Cabel-dxf immer gleich heisst oder so wie Eingabefile (Gefahr der Zumüllung)
- selbes gilt für excel-File