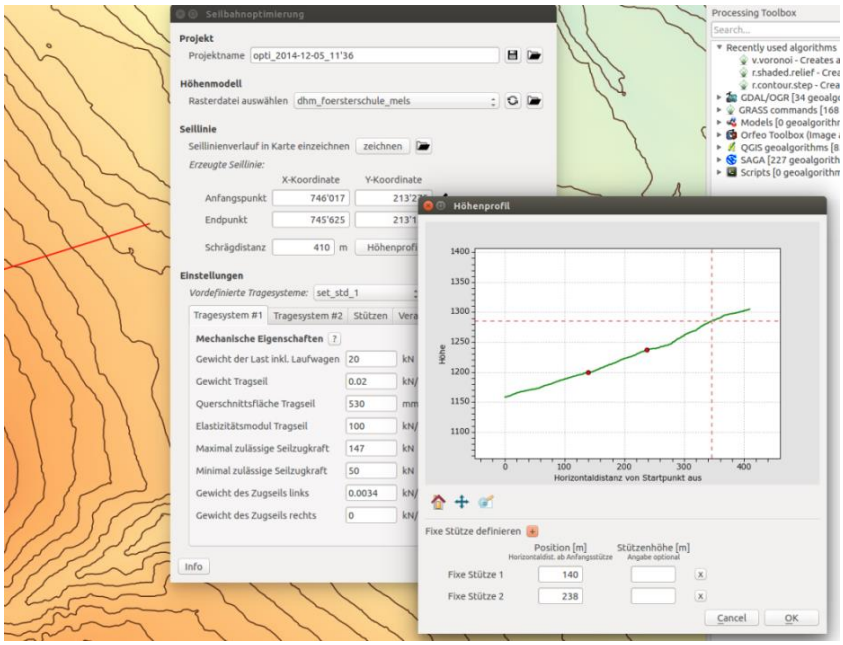
Technische Dokumentation

SEILAPLAN V3.6

QGIS Plugin zur Planung von Seilkran-Layouts



Inhaltsverzeichnis

1. Zweck des Plugins

Das Plugin implementiert Leo Bonts Optimierungs-Algorithmus zur Planung eines forstlichen Seilkran-Layouts. Das Programm ist fähig, auf Grund eines digitalen Höhenmodells (DEM) oder Profilpunkten aus Feldaufnahmen zwischen definierten Anfangs- und Endkoordinaten das optimale Seillinienlayout zu berechnen (Position und Höhe der Stützen).

Das Programm ist für mitteleuropäische Verhältnisse konzipiert und geht von einem an beiden Enden fix verankerten Tragseil aus. Für die Berechnung der Eigenschaften der Lastwegkurve wird eine iterative Methode verwendet, welche von Zweifel (1960) beschrieben und speziell für an beiden Enden fix verankerten Tragseilen entwickelt wurde. Bei der Prüfung der Machbarkeit der Seillinie wird darauf geachtet, dass 1) die maximal zulässigen Spannungen im Tragseil nicht überschritten werden, 2) ein minimaler Abstand zwischen dem Tragseil und dem Untergrund gegeben ist und 3) bei einem Einsatz eines Gravitationssystems eine minimale Neigung im Tragseil gegeben ist. Es wird diejenige Lösung gesucht, welche in erster Priorität eine minimale Anzahl an Stützen aufweist und in zweiter Priorität die Stützenhöhe minimiert.

Die Einbettung im kostenlos erhältlichen Geografischen Informationssystem QGIS (qgis.com) ermöglicht es, räumliche Daten zur Planung (Höhenmodell, Seillinie und Stützenpunkte) auf einer Karte darzustellen und die Geo-Funktionalität des GIS ins Tool einzubeziehen. So ist es beispielsweise möglich, die Seillinie in der Karte einzuzeichnen und sich das Höhenprofil anzeigen zu lassen.

2. Funktionsumfang und Bedienung

Für die Ausführung des Optimierungs-Algorithmus steht eine grafische Benutzeroberfläche (GUI = graphic user interface) zur Verfügung, mit deren Hilfe Start- und Endpunkt der Seillinie, die Eigenschaften des Tragesystems und weitere Einstellungsmöglichkeiten vom Benutzer spezifiziert werden können.

Ein weiteres Fenster nach Abschluss der Berechnungen erlaubt es, die Seillinie weiter zu verfeinern und manuelle Anpassungen anzubringen.

2.1. Kompatibilität

Das Plugin wurde unter Ubuntu Linux entwickelt und auf Windows und macOS getestet.

Das Plugin wurde ursprünglich unter QGIS Version 2.6 und 2.8 entwickelt und im Nachhinein auf QGIS Version 3.4 portiert. Das Plugin ist generell mit allen 3er Version von QGIS lauffähig mit Ausnahme der den Version 3.10.9 und 3.14.15, welche einen Fehler enthalten (Seilaplan informiert in diesem Fall beim Start, dass es nicht ausgeführt werden kann). Da der Wechsel auf Version 3 grosse Änderungen in der Python API mit sich gebracht hat, ist die Unterstützung von QGIS Versionen kleiner als 3.0 nicht mehr möglich.

Im Zuge verschiedener Weiterentwicklungen ist das Plugin seit Version 3.2 für QGIS 3.6 und höher verfügbar.

2.2. Installation

Informationen zur Installation von QGIS sind auf http://qgis.com/ zu finden.

Abb. 1
SEILAPLAN Icon

Das Plugins wird in QGIS über das Erweiterungs-Menü hinzugefügt. In *Erweiterungen verwalten und installieren*…kann unter *Einstellungen* ein neues Repositorium hinzugefügt werden. Die Adresse des Repositoriums lautet

https://raw.githubusercontent.com/piMoll/SEILAPLAN/master/plugin.xml

Im Feld *Namen* kann jeglicher Name eingesetzt werden. Nach Bestätigung des Dialogfensters kann im Reiter *Nicht Installiert* das Plugin per Suchabfrage (‚SEILAPLAN') gefunden und installiert werden. Bei der Installation kann es vorkommen, dass QGIS für eine kurze Zeit einfriert. Das Plugin Icon (Abb. 1) sollte nun im oberen Teil der QGIS Bedienoberfläche

erscheinen. Falls nicht, kann per Rechtsklick auf die Oberfläche und Aktivierung des Eintrags *Werkzeugkästen* > *Erweiterungswerkzeugleiste* die aktivierten Plugins sichtbar gemacht werden.

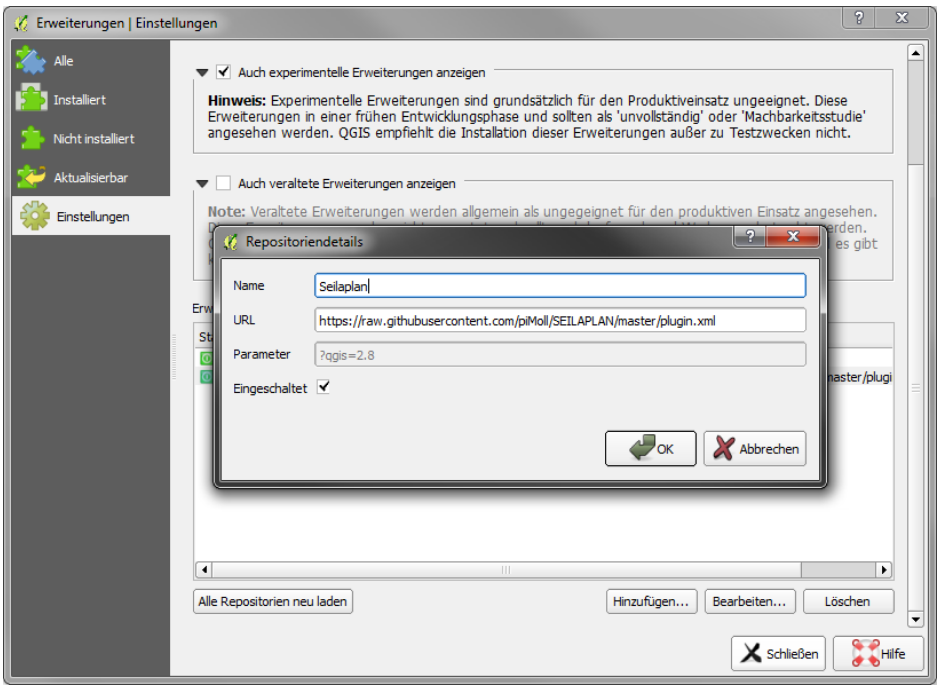


Abb. 2 Repositorium hinzufügen

2.3. Deinstallation

Das Plugin kann über das QGIS Menü *Erweiterungen verwalten und installieren...* deaktiviert oder deinstalliert werden. Im Reiter *Installiert* den Eintrag SEILAPLAN auswählen und auf deaktivieren, bzw. deinstallieren klicken.

2.4. Problembehebung

War die Installation oder Deinstallation nicht erfolgreich oder sind plötzlich zwei Plugins installiert, sollten die Plugin-Daten gelöscht und anschliessende eine Neuinstallation durchgeführt werden. Sofern keine speziellen QGIS-Profile angelegt wurden, befinden sich Plugins unter Windows im Ordner

C:\Users\<benutzername>\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins

<*benutzername*> muss mit dem korrekten Windows User ersetzt werden. Der Plugin-Ordner SEILAPLAN kann gelöscht werden, QGIS sollte dabei jedoch nicht laufen. Anschliessend QGIS öffnen und SEILAPLAN erneut installieren. Unter Linux sind die QGIS Plugins in

/home/<USER>/.local/share/QGIS/QGIS3/profiles/default/python/plugins zu finden (standardmässig versteckt).

3. Code bearbeiten

3.1. Allgemeines zur Arbeit mit QGIS-Plugins

Wenn Code in einem Plugin verändert wird, werden diese Änderungen erst mit einem Neustart von QGIS umgesetzt. Um das dauernde Neustarten zu umgehen, gibt es ein separates QGIS Plugin, welches das Neuladen des Codes übernimmt. Es heisst *Plugin Reloader* und kann über das Plugin Menü installiert werden (experimentelle Plugins aktivieren).

Wenn bereits beim Start von QGIS oder beim Neuladen mit dem Plugin Reloader eine Fehlermeldung gezeigt wird, ist der Ursprung des Fehlers meist einer der folgenden drei Möglichkeiten: eine der vielen __init__ Methoden enthält fehlerhaften Code, ein Import ist falsch, Bibliothek kann nicht gefunden werde oder im Code gibt es ein Syntaxfehler.

3.2. Einrichtung IDE

Das Plugin wurde in der IDE Jetbrains PyCharm Professional Edition entwickelt, da diese Software Remote Debugging erlaubt (siehe Kapitel 3.3). Erklärungen und Konfigurationen beziehen sich nachfolgend immer auf dieses Programm.

Das Zusammenspiel von QGIS und Python ist je nach Betriebssystem unterschiedlich. Damit in der IDE das Plugin ausgeführt werden kann, muss zuvor diejenige Python Distribution ausgewählt werden, welche auch von QGIS verwendet wird. Auf Windows und macOS wird QGIS mit einer eigenen Python Distribution ausgeliefert, auf Ubuntu wird das System-Python verwendet.

Ubuntu
Bei der Installation von QGIS unter Ubuntu wird kein eigenes Python mitgeliefert, stattdessen wird das bereits installierte System-Python benutzt. Um herauszufinden, welche Python Version von QGIS genutzt wird, kann man in der QGIS Python-Konsole folgende Befehle eingeben:

```
import os
os.__file__
>> /usr/lib/python3.7/os.py
```

Bei der Ausführung des Plugin-Codes sollte in diesem Beispiel als Interpreter also Python 3.7 ausgewählt werden (die Python Version ändert sich je nach QGIS Version).

Windows
Der korrekte Python Interpreter befindet sich im QGIS-Installationsverzeichnis, beispielsweise hier:
`C:\Program Files\QGIS 3.10\apps\Python37\python.exe`

Damit die QGIS-Python-Bibliotheken angesprochen werden können, muss zusätzlich der Python-Path (sys.path) mit diesem (oder ähnlichem) Pfad ergänzt werden:
`C:\Program Files (x86)\QGIS 3.10\apps\qgis\python`

Der Pfad kann direkt beim Einrichten des Projekt-Interpreters in PyCharm ergänzt werden, so sind Code Completions in der IDE möglich. Alternativ wird der Pfad bei der Ausführung des Standalone Skript automatisch ergänzt.

macOS
Auch unter macOS muss der Python Interpreter im QGIS-Installationsverzeichnis gesucht werden.
`/Applications/QGis.app/Contents/Resources/python`

Ansonsten sind keine weiteren Schritte notwendig.

3.3. Remote Debugging mit PyCharm und QGIS

Remote Debugging ist dann notwendig, wenn die grafischen Komponenten des Plugin Interface (z.B. Hauptfenster, Bearbeitungsfenster, Diagrammerstellung, etc.) oder die Interaktion mit QGIS (Layer-Erstellung, Zeichnen in Karte, etc.) analysiert werden sollen. Wenn nur der Optimierungsalgorithmus von Interesse ist, kann stattdessen das STANDALONE.py Skript verwendet werden, welches mit normalen Debugging-Werkzeugen ausgeführt werden kann. Genauere Erklärungen dazu im nächsten Kapitel.

Weil der Plugin Code nur aus QGIS heraus gestartet werden kann, ist ein normales Debugging in der IDE nicht möglich. Stattdessen muss in PyCharm „Remote-Debugging“ konfiguriert werden, wofür die PyCharm Professional Edition notwendig ist (in Community Edition nicht enthalten).

Beim Remote Debugging wartet die IDE auf ein Signal von QGIS, dass das Plugin gestartet wurde, worauf sie sich in die Code-Ausführung einklinkt und den Debug Modus aktivieren. Es ist dann möglich, so viele Breakpoints wie gewünscht zu setzen und mit den internen Debug-Werkzeugen anzusteuern. Um das Signal auszusenden, muss der nachfolgende Code im Plugin vorhanden sein. Der Code ist in seilaplanPlugin.py bereits vorhanden, jedoch auskommentiert.

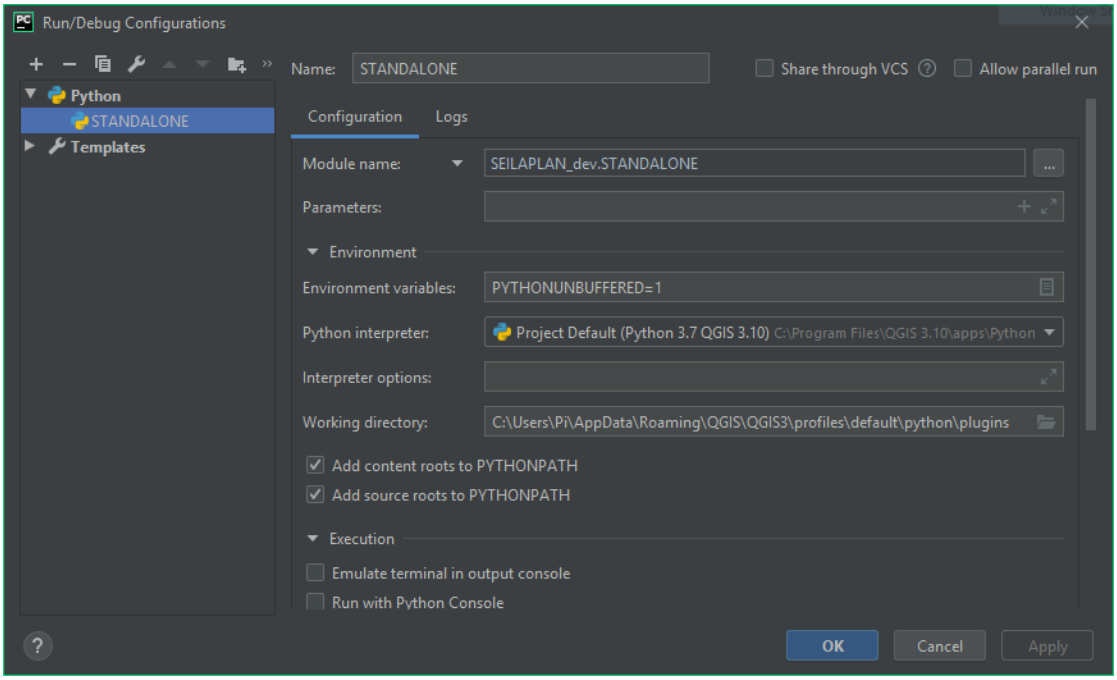
```
try:
   import pydevd_pycharm
   pydevd_pycharm.settrace('localhost', port=53100, stdoutToServer=True, stderrToServer=True)
except ConnectionRefusedError:
    pass
except ImportError:
    pass
```

Weitere Voraussetzungen für das Gelingen von Remote Debugging sind unter anderem die Python-Bibliothek pydevd. Ausführliche Anweisung zum Einrichten des Remote Debuggings können in der PyCharm Dokumentation nachgelesen werden.

3.4. Standalone-Skript

Soll nur der Optimierungsalgorithmus bearbeitet werden, kann das Plugin separat von QGIS ausgeführt werden. Damit alle Input-Daten und sonstige Voraussetzungen für den Algorithmus vorhanden sind, muss der Code über das Skript STANDALONE.py ausgeführt werden. Bei der Ausführung werden keine Dialogfenster gezeigt, die Benutzereingaben müssen deshalb über ein zuvor abgespeichertes Projektfile eingelesen werden. Das Standalone Skript enthält einen kurzen Hilfetext und Markierungen, wo Anpassungen notwendig sind.

Um das Debugging starten zu können, muss eine neue Debug-Konfiguration mit folgenden Einstellungen eingerichtet werden:



Der *Module name* entspricht dem Verzeichnisnamen + Standalone-Skriptnamen, könnte also je nach Verzeichnisname auch „SEILAPLAN.STANDALONE“ heissen. Der *Python Interpreter* muss unbedingt korrekt eingerichtet sein, siehe dazu Kapitel 3.2. Das *Working Directory* entspricht dem Pfad zu den QGIS-Plugins (nicht zum Plugin selbst!).

3.5. Code auf GitHub verwalten und versionieren

Der Plugin Code wird auf dem Repository https://github.com/piMoll/SEILAPLAN gehosted und weiterentwickelt. Weiterentwicklungen werden typischerweise in den dev-Branch commited, erst bei Abschluss aller Arbeiten und einem gründlichen Testing wird der dev-Branch zurück in den master-Branch gemerged. Zur automatischen Verteilung neuer Versionen des Plugins sind grundsätzlich zwei Aspekte notwendig:

- Aktualisierter Plugin Code in Zip „Seilaplan.zip“ auf Git vorhanden (am besten in einem Release)
- Plugin.xml aktualisiert: Plugin-Version und Downloadlink angepasst

Weitere Arbeiten vor einem Release werden in der build-Anleitung unter scripts > build_checklist.md beschrieben. In diesem Verzeichnis sind auch weitere Hilfestellungen für Entwickler zu finden

Falls beim Benutzer die automatische Überprüfung auf Updates in QGIS aktiviert wird, wird bei einem Neustart automatisch erkannt, dass eine neue Version des Plugins zum Download bereit steht. Per Klick auf die Benachrichtigung am unteren Rand wir das Plugin aktualisiert.

4. Struktur des Plugins

4.1. Plugin-Ablauf

Ausgangspunkt des Plugins ist die Datei seilaplanPlugin.py, in der sich die Funktion run() befindet. Die Funktion wird aufgerufen, wenn der Benutzer das SEILAPLAN-Icon in QGIS anklickt. Die run() Methode erstellt eine Instanz der Klasse seilaplanRun, welche den gesamten Ablauf des Plugins steuert, vom Hauptfenster über die Optimierung bis zum Bearbeitungsfenster. Jedes Mal wenn der Benutzer SEILAPLAN erneut öffnet, wird eine neue Instanz von seilaplanRun erstellt, dies erlaubt, das Plugin parallel mehrfach auszuführen.

Jede seilaplanRun Instanz initialisiert zu Beginn den ConfigHandler, eine wichtige Klasse, mit deren Hilfe alle Einstellungen, Parameter und Resultate organisiert werden.

Das Plugin zeigt beim Start ein erstes grosses Dialogfenster, das Hauptfenster. Hier können Projekteinstellungen geladen/gespeichert werden, die Terraininformationen ausgewählt und Anfangs- und Endpunkt definiert werden. Parameter der Seillinie können angepasst werden und in neuen Parametersets abgespeichert werden. Ist ein Anfangs- und Endpunkt definiert, kann man sich die Profillinie über den Button *Geländelinie* in einem Diagrammfenster darstellen lassen.

Vom Hauptfenster aus gibt es zwei mögliche Optionen: Man kann die Optimierung starten oder direkt ins Bearbeitungsfenster wechseln.

Wird die Optimierung ausgeführt, wird unter Einhaltung der gewählten Parameter die optimale Position und Höhe der Stützen zwischen Anfangs- und Endpunkt berechnet. Die Berechnung des optimalen Seillinienlayouts wird in einem separaten Thread ausgeführt, damit QGIS währenddessen ansprechbar bleibt. Ein kleines Fenster zeigt während der Berechnung den Fortschritt an. Das Resultat der Optimierung ist ein Seillinienlayout, welches im Bearbeitungsfenster dargestellt und weiter bearbeitet werden kann.

Es ist auch möglich, die Optimierung zu überspringen und direkt ins Bearbeitungsfenster zu wechseln. Der Benutzer hat dann die Möglichkeit auf der „grünen Wiese“ ein eigenes Seillinienlayout zu planen.

Im Bearbeitungsfenster wird bei jeder Anpassung einer Stütze oder den Seillinien-Kennwerten die Seillinie neu berechnet und im Diagramm dargestellt. Das Seillinienlayout kann in verschiedene Output-Produkte exportiert und abgespeichert werden und zu einem späteren Zeitpunkt wieder geöffnet werden.

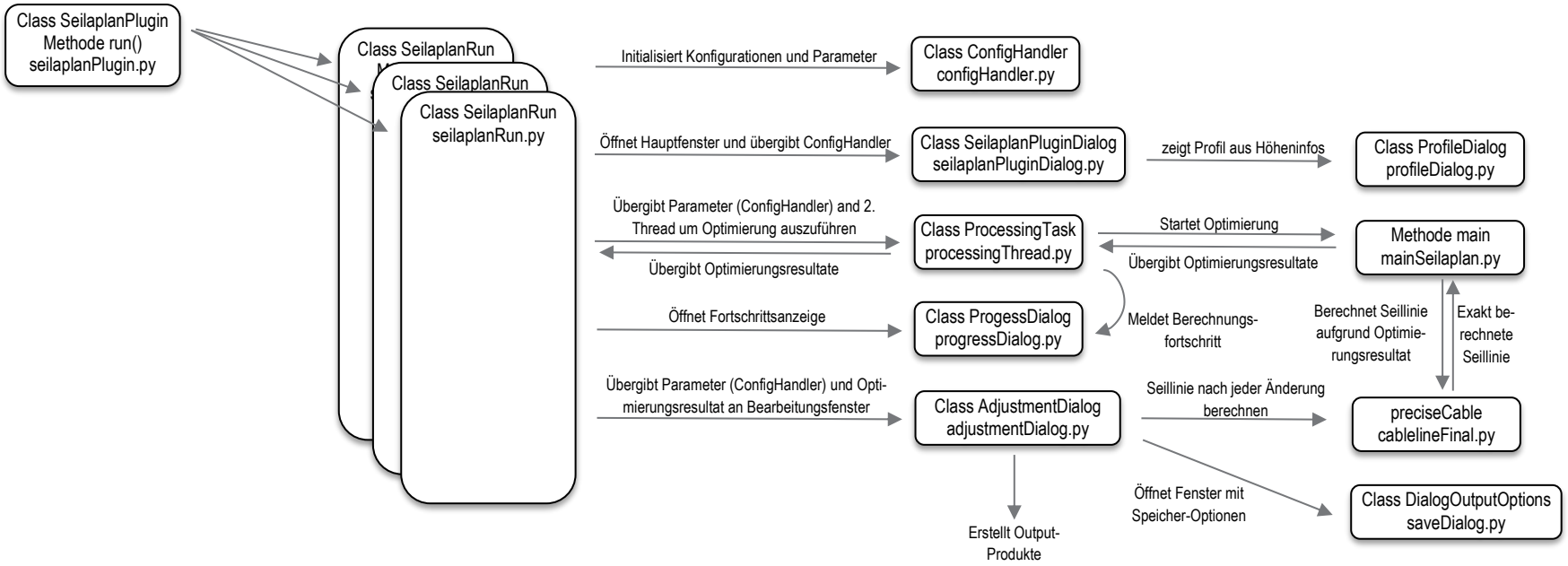


Abb. 3 Ablauf-Struktur des Plugins

4.2. Dateiverzeichnis

Einen allgemeinen Einstieg in QGIS Plugins und eine Übersicht über Grundkomponenten liefert die offizielle QGIS Seite: https://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/plugins/index.html

Das Seilaplan Plugin ist mittlerweile nicht mehr nur ein Optimierungsalgorithmus. Ein Grossteil der Skriptdateien befasst sich mit der Benutzerinteraktion, also dem Aufbau und der Funktionalität der verschiedenen Dialogfenster.

Nachstehend sind die Dateien im Skript-Verzeichnis und deren Zweck aufgelistet (nicht abschliessend).

	Skriptdatei	Beschreibung
config		Ordner mit Parameter-Dateien.
	Ordner parametersets	Enthält vordefinierte und vom Benutzer gespeicherte Parameter-Sets,
	Ordner birdView	Enthält Symbole für die Vogelperspektive im numpy Textformat. Die Objekte werden mit Hilfe der Python Bibliothek matplotlib als Path Objekte in tools > birdViewSymbol.py geladen. Die Datei _pole_symbols.svg (Inkscape Datei) enthält die SVG Pfade.
	params.txt	Enthält die Parameter-Definition: Erklärungen, Wertebereiche, Datentypen und Einheiten, die zur Prüfung der Werte verwendet werden.
	commonPaths.txt	Enthält die vom Benutzer gewählten Output Pfade und die Einstellungen für die Output Optionen (Checkboxen).
core		Sämtliche Scriptdateien des Kernalgorithmus.
	cableline.py	Berechnung der Seillinie basierend auf der Methode von Zweifel während des Optimierungsdurchlaufes.
	cablelineFinal.py	Berechnet den Seilverlauf, die exakte Seilmechanik und eine Vielzahl von Kennwerten des Tragesystems nach Fertigstellung des Optimierungsprozesses.
	mainOpti.py	Hauptskript für die Optimierung, hier wird das Optimierungsproblem formuliert und die Suche nach dem kürzesten Pfad ausgeführt.
	mainSeilaplan.py	Hauptskript zur Steuerung der Optimierung.
	optiSTA.py	Berechnet den Bereich der Anfangsseilzugkraft STA, welcher für den betrachteten Abschnitt möglich ist.
	peakdetect.py	Kleines Tool um Erhöhungen und Spitzen im Höhenprofil zu bestimmen.
	terrainAnalysis.py	Bestimmt das Bodenprofil durch Interpolation auf dem Höhenraster, berechnet die möglichen Stützenpositionen mit Berücksichtigung von Bodenabstand und Befahrbarkeit, bestimmt die Ankerparameter.
gui		Enthält alle Dateien, die für das Graphical User Interface zuständig sind
	Ordner icons	Enthält Icon-Bilder
	adjustmentDialog.py	Hauptklasse für das Bearbeitungsfenster, steuert Aufbau der GUI, Neuberechnung der Seillinie und Abspeichern der Daten.
	adjustmentDialog.ui	QtDesigner Datei des Bearbeitungsfensters.
	adjustmentDialog_pa..	Verantwortlich für den Inhalt im Reiter Kennwerte im Bearbeitungsfenster.
	adjustmentDialog_thr..	Verantwortlich für den Inhalt im Reiter Grenzwerte im Bearbeitungsfenster.
	adjustmentPlot.py	Zeichnet das Diagramm im Bearbeitungsfenster und die Vogelperspektive.
	birdViewWidget.py	GUI-Klasse baut und steuert Stützen-Zeilen im Tab Vogelperspektive.
	guiHelperFunctions.py	Einige kleine Funktionen, die das File selaplanPluginDialog.py ergänzen.
	mapMarker.py	Zeichnet die rote Profillinie und die Stützenstandorte in die QGIS Karte.
	plotting_tools.py	Funktionen, die von beiden Diagrammen (Geländelinie und Bearbeitungsfenster) verwendet werden.
	poleWidget.py	GUI-Klasse baut und steuert Stützen-Zeilen (Name, Distanz, Höhe, etc.). Die Klasse wird im Geländelinie-Fenster und Bearbeitungsfenster eingesetzt.
	profileDialog.py	GUI-Klasse baut das Geländelinie-Fenster auf.

	profilePlot.py	Zeichnet das Diagramm im Geländelinie-Fenster.
	progressDialog.py	Fortschrittsfenster, welches während der Berechnungen angezeigt wird.
	resources.qrc	XML-artiges File, das zu seilaplanDialog.py gehört und die benötigten Icons auflistet. Dieses File muss ergänzt werden, wenn weitere Icons in der GUI dargestellt werden sollen.
	resources_rc.py	Übersetzung des resources.qrc File nach Python. Die Übersetzung muss über das Terminal ausgeführt werden (siehe Kapitel 4.3).
	saveDialog.py	Enthält zwei kleinere Dialoge; Das Fenster um Parametersets abzuspeichern und das Speichern-Fenster.
	seilaplanDialog.ui	QtDesigner Datei des Hauptfensters.
	seilaplanPluginDialog.py	GUI-Klasse des Hauptfensters. Enthält (fast) alle Logik, die für das Einfüllen der Benutzerdaten notwendig ist.
	surveyImportDialog.py	Dialogfenster für den Import von Geländeprofilen
	surveyImportDialog.ui	QtDesigner Datei des Geländeprofil-Import Dialogfenster.
	ui_adjustmentDialog.p	Übersetzung des QtDesigner-Files des Bearbeitungsfensters nach Python. Die Übersetzung muss über das Terminal ausgeführt werden (siehe Kapitel 4.3).
	ui_seilaplanDialog.py	Übersetzung des QtDesigner-Files des Hauptfensters nach Python. Die Übersetzung muss über das Terminal ausgeführt werden (siehe Kapitel 4.3).
	ui_surveyImportDialog	Übersetzung des QtDesigner-Files des Geländeprofil-Import Dialogfenster nach Python. Die Übersetzung muss über das Terminal ausgeführt werden (siehe Kapitel 4.3).
help		Enthält PDF der Dokumentation
i18n		Enthält Übersetzungsdaten
img		PNG-Files für Info-Fenster
lib		Enthält die Bibliothek reportlab, die für die Erstellung von PDF-Exports notwendig ist (nicht in Standard-Python enthalten).
scripts		Verzeichnis mit einigen hilfreichen Notizen, Skripts und Terminal-Befehlen für die Arbeit mit Seilaplan.
templates		Enthält diverse Vorlagen für Geländeprofile. Diese Vorlagen können im Plugin heruntergeladen werden, dabei wird eine Verbindung zu Github hergestellt. Die Templates sind nicht im Release-Build enthalten.
test		Enthält verschiedene Test Cases
tools		Helferklassen und Funktionen, damit die Verbindung zwischen Kernalgorithmus und QGIS-GUI funktioniert. Hier sind z.B. das Handling von Projektfiles oder das Multithreading während der Berechnung zu finden.
	birdViewMapExtractor.py	Extrahiert den Ausschnitt, der in der Vogelperspektive-Darstellung hinterlegt ist.
	birdViewSymbol.py	Lädt die Stützensymbole, die in der Vogelperspektive dargestellt werden.
	calcThreshold.py	Berechnen und Darstellen von Kennwerten , welche überschritten werden (Bearbeitungsfenster > Reiter Kennwerte)
	configHandler.py	Schaltzentrale für alle Eingabeparameter. Erläuterungen siehe Kapitel 4.5.
	configHandler_params.py	Verwaltet Berechnungsparameter, überprüft und aktualisiert sie.
	configHandler_project.py	Verwaltet Projekteinstellungen, erstellt notwendige Objekte für das Anpassungsfenster (Profil, Stützen) oder die Optimierung.
	processingThread.py	Klasse vom Typ QgsTask; Übernimmt Inputdaten des Benutzers und startet einen zweiten Thread um den Optimierungsalgorithmus durchzuführen.
	fileFetcher.py	Kleine Methode um Daten aus dem Internet herunterzuladen. Wird verwendet um Vorlagen von Github zu holen.

	heightSource.py	Lädt und analysiert die Terraindaten: enthält die Klassen Raster und SurveyData (=Feldaufnahmen). Klasse Raster ist zuständig für den Zuschnitt des gewünschten Rasterausschnitts. SurveyData transformiert und interpoliert Felddaten entlang einer Geraden.
	importCsvVertex.py	Geländeprofile vom Typ Haglöf Vertex importieren
	importCsvXyz.py	Geländeprofile vom Typ XYZ importieren
	importExcelProtocol.	Geländeprofile vom Typ Feldprotokoll importieren
	poles.py	Speicherobjekt für alle Stützen des Stützenlayouts. Generiert die initialen Stützen aus den ausgewählten Parametern, übernimmt die aus der Optimierung hervorgehenden Zwischenstützen und kennt alle Regeln bezüglich Verankerungen und Bezeichnung der Stützen. Die Klasse hilft auch, das Bearbeitungsfenster korrekt aufzubauen.
	profile.py	Speicherobjekt für alle Eigenschaften des Profils, also des Terrainverlaufs. Enthalten sind unterschiedlichste Arrays der Terrainhöhe, X- und Y-Koordinaten, Befahrbarkeits-Informationen, usw.
	outputReport.py	Erstellt Ausgabeplot mit Seillinie und Höhenprofil und erzeugt den Bericht.
	outputGeo.py	Erstellt Shape-Daten und CSV-Koordinatentabellen.
seilaplanPlugin.py		Haupt-Plugin-Script, Einstiegspunkt für Ausführung des Plugins. Die Funktion run() ist der erste Code, der beim Klick auf das Plugin-Icon ausgeführt wird. In der Funktion run() wird eine seilaplanRun Instanz erstellt und gestartet.
seilaplanRun.py		Organisiert den Programmflow pro Plugin-Ausführung. Erlaubt es, das Plugin mehrfach parallel zu öffnen. Öffnet das Hauptfenster und steuert das weitere Vorgehen (Optimierung oder direkt zum Beabreitungsfenster), öffnet Dialogfenster und schliesst das Plugin.
README.md		Infotext für die Github Startseite.
metadata.txt		Beschreibt das Plugin kurz und enthält alle Kenndaten, welche für ein QGIS-Plugin angegeben werden müssen (Version, Autor, etc.). Dies sind die Texte, die im Pluginverzeichnis im QGIS Menü Erweiterungen angezeigt werden.
STANDALONE.py		Separates Skript, welches für die Ausführung des Optimierungsalgorithmus ohne QGIS notwendig ist. Genauere Erklärungen sind in Kapitel 3.4 zu finden.

4.3. GUI

Da die Oberfläche von QGIS in Qt geschrieben ist (http://qt-project.org), muss auch für das Plugin-Design Qt und damit die Python Bindings PyQt verwendet werden.

Für Qt steht der grafische Editor QtDesigner zur Verfügung, mit dem die GUI erstellt werden kann. Die Positionierung von Labels, Feldern und Buttons ist damit sehr einfach, die Funktionen einzelner Buttons muss aber nachträglich programmiert werden.

Beim Erstellen von neuen Eingabefelder sollte darauf geachtet werden, dass das Feld einen sinnvollen Objektnamen erhält. Dieser Name muss anschliessend im Code aufgerufen werden, wenn ein Feld ausgelesen oder befüllt werden soll.

Das QtDesigner-Projekt (*.ui) kann mittlerweile direkt im Python-Code geladen und interpretiert werden. Dazu muss das *.ui File vor der Definition der Dialogklasse mittel uic Utility geladen werden und dann als Klassenparameter übergeben werden:

```
import os
from qgis.PyQt import uic
UI_FILE = os.path.join(os.path.dirname(__file__), 'seilaplanDialog.ui')
FORM_CLASS, _ = uic.loadUiType(UI_FILE)
class SeilaplanPluginDialog(QDialog, FORM_CLASS):
        …
```

Früher war es notwendig, das File erst in Python Code zu übersetzen. Zur Vollständigkeit, hier das veraltete Vorgehen:

1. Konsole öffnen und mit Befehl `cd` zu Ordner wechseln, der die ui-Datei enthält:
   `cd /Path/to/Folder/SEILAPLAN/gui/`
2. Übersetzung mit Befehl `pyuic5` ausführen:
   `pyuic5 seilaplanDialog.ui -o ui_seilaplanDialog.py`
3. In der neu erstellen Datei ui_seilaplanDialog.py die allerletzte Zeile korrigieren, anstatt `import resources_rc` zu:
   `from . import resources_rc`

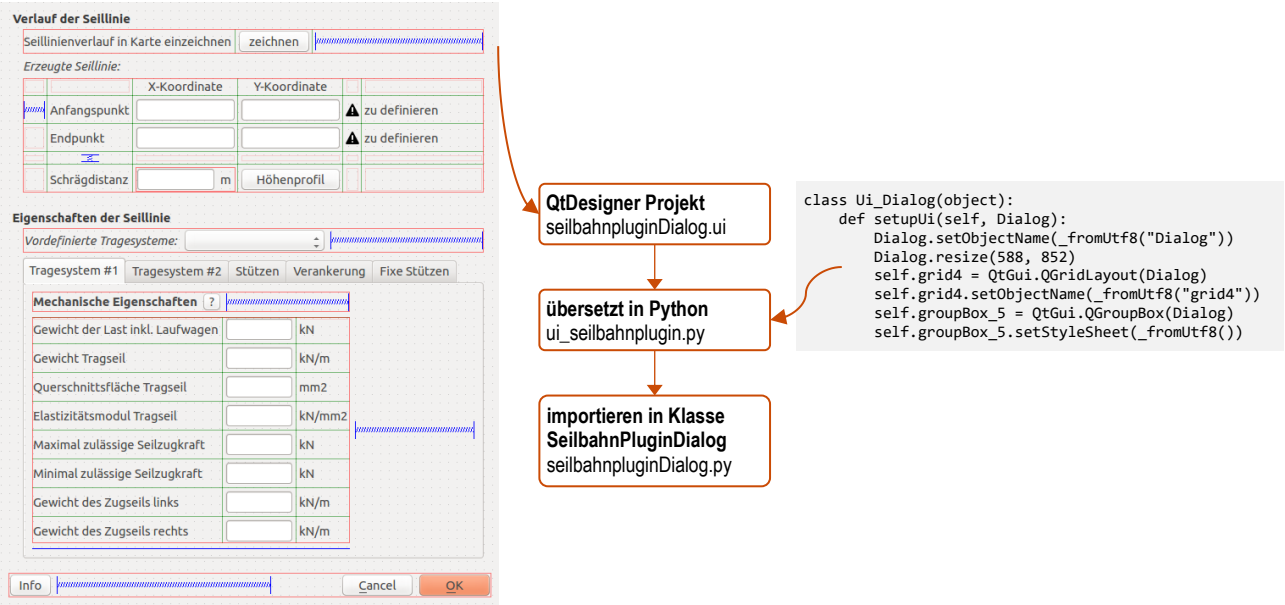


Abb. 4 Aussehen der GUI, erstellt in QtDesigner

Falls man ein neues Icon in der GUI benutzen möchte, müssen zuerst die Ressourcen ergänzt werden. In der Datei resources.qrc sind alle Icons aufgelistet, die in der GUI verwendet werden. Diese Liste kann mit dem neuen Icon ergänzt werden, womit das Icon in QtDesigner auswählbar wird.

Auch die Ressourcen-Datei muss nach einer Anpassung nach Python übersetzt werden, damit die Ausführung des Plugins funktioniert:

1. Konsole öffnen und mit Befehl `cd` zu Ordner wechseln, der die qrc-Datei enthält:
   `cd /Path/to/Folder/SEILAPLAN/gui/`
2. Übersetzung mit Befehl `pyrcc5` ausführen:
   `pyrcc5 –o resources_rc.py resources.qrc`

4.4. Übersetzung

Das Plugin beherrscht vier Sprachen: deutsch, französisch, italienisch und englisch. Je nach Spracheinstellungen von QGIS passt sich Seilaplan automatisch an die aktuelle Sprache an. Falls die Sprache von QGIS nicht übernommen werden kann, wird Seilaplan in Deutsch angezeigt.
Um die Sprache des Plugins manuell zu ändern, muss erst die Spracheinstellung in QGIS angepasst werden und das Plugin danach mittels *Plugin Reloader* neu geladen werden.
Die Übersetzungstexte werden beim Start des Plugins von der Bibliothek pyQt ausgelesen und je nach aktuell ausgewählter Sprache korrekt in der GUI dargestellt. Im Python-Code wird der benötigte Text mittels der pyQt-Funktion tr() / self.tr() angesprochen. Damit der benötigte Text geliefert wird, muss der Funktion der eindeutige Schlüssel des benötigten Textes übergeben werden.
Die Übersetzungsdaten befinden sich im Verzeichnis i18n, darin enthalten sind folgende Dateien:

- *.ts Datei pro Sprache: enthält alle Texte als Schlüssel-Wert Paare in einem lesbaren XML-Format
- *.qm Datei pro Sprache: enthält die gleiche Information wie im *.ts Datei, jedoch in binärer Form
- SeilaplanPlugin_i18n.pro: Konfigurationsdatei für das Übersetzen der .ts Files in das qm-Format

*Texte anpassen in der *.ts Datei*: Sollen einzelne wenige Begriffe angepasst werden, kann dies direkt in den *.ts Files durchgeführt werden.
Im ts-File befinden sich die Übersetzungstexte in `<message>` Blöcken. Der Schlüssel oder Identifikator ist im Tag `<source>` angegeben. Dieser sollte nicht verändert werden, da im Python-Code darauf verwiesen wird. Die Übersetzung folgt anschliessend im `<translation>` Tag, dieser Text kann nach Belieben angepasst werden. Die Angaben in `<location>` dienen als Hilfe zum Finden der Begriffe im Code, sind jedoch nicht unbedingt korrekt und können beim Anpassen ignoriert werden. Alle Übersetzungstexte einer Python-Klasse sind wiederum in einem `<context>` gruppiert.

```
<context>
    <message>
        <location filename="../tool/outputGeo.py" line="58"/>
        <source>stuetzen</source>
        <translation>Stützen</translation>
    </message>
</context>
```

Text anpassen in QtLinguist: Etwas übersichtlicher und angenehmer ist die Bearbeitung von Übersetzungstexten mit Hilfe des QtLinguist. Mit diesem Desktop-Programm lassen sich die ts-Dateien laden und übersichtlich in einer Tabelle darstellen. Das Programm kann Vorschläge machen und sogar die GUI-Oberfläche in einem kleinen Fenster darstellen.

Änderungen nach Bytecode übersetzen: Angepasste Übersetzungen, egal ob im QtLinguist oder direkt im *.ts File, werden in der GUI nur sichtbar, wenn die ts-Files mittels dem Konsolen-Tool `lrelease` in *.qm Files übersetzt werden. Dazu muss folgender Befehl im Seilaplan Home-Verzeichnis auf der Konsole ausgeführt werden:

```
lrelease i18n/SeilaplanPlugin_i18n.pro
```

Das Tool überschreibt die existierenden *.qm Files und gibt auf der Konsole einen Statusbericht der vorhandenen Übersetzungen aus. Achtung: Dieses Tool übersetzt nur Einträge, welche fertig übersetzt sind, also keinen Flag „unfinished“ besitzen.

Neu Begriffe ergänzen: Soll ein neuer Begriff ergänzt werden, kann im .ts File ein `<message>` Block kopiert werden und im korrekten `<context>` eingefügt werden. Die Reihenfolge der message-Blöcke ist nicht relevant. Es muss ein neuer Schlüssel `<source>` vergeben werden, dabei gilt, dass keine Umlaute oder Spezialzeichen verwendet werden dürfen. Der Message-Block muss in alle Sprachfiles kopiert werden und die korrekte Übersetzung im `<translation>` Teil ergänzt werden.

4.5. Konfigurationsmanager

Alle Einstellungen und Inputs an den Optimierungsalgorithmus werden vom Konfigurationsmanager in configManager.py und den untergeordneten Klassen (configHandler_*) organisiert. Der Konfigurationsmanager ist die Drehscheibe für alle Daten, die zwischen Benutzer und Optimierungsalgorithmus hin und her laufen. Die Hauptarbeiten leisten die zwei Unterklassen, der ParameterConfHandler und der ProjectConfHandler.
Der ProjectConfHandler kontrolliert die Eingabe von Anfangs- und Endpunkt, berechnet die Profillänge und das Azimut, speichert fixe Stützen oder den Punkttyp. Die Klasse initialisiert zudem einige wichtige Objekte:

- HeightSource lädt und analysiert die Terraindaten: enthält die Klassen Raster und SurveyData (=Feldaufnahmen). Klasse Raster ist zuständig für den Zuschnitt des gewünschten Rasterausschnitts. Klasse SurveyData transformiert und interpoliert Felddaten (CSV-Daten) entlang einer Geraden.
- Profile: Speicherobjekt für alle Eigenschaften des Profils, also des Terrainverlaufs. Enthalten sind unterschiedlichste Arrays der Terrainhöhe, X- und Y-Koordinaten, Befahrbarkeits-Informationen, usw.
- Poles: Speicherobjekt für alle Stützen des Stützenlayouts. Generiert die initialen Stützen aus den ausgewählten Parametern, übernimmt die aus der Optimierung hervorgehenden Zwischenstützen und kennt alle Regeln bezüglich Verankerungen und Bezeichnung der Stützen. Die Klasse hilft auch, das Bearbeitungsfenster korrekt aufzubauen.

Der ParameterConfHandler lädt Parameterdefinitionen, Parametersets, speichert Parameter und gibt sie mit korrekter Formatierung zurück, kontrolliert Parametereingaben und berechnet hergeleitete Parameter.
Der Konfigurationsmanager ist auch dafür zuständig, dass alle Projekteinstellungen abgespeichert und wiederhergestellt werden können. Dadurch ist es möglich, den Optimierungsalgorithmus ohne die Qt-Oberfläche ausführen zu können (mit Hilfe des Standalone Skripts). Alle Eingabewerte können aus Projekteinstellungen geladen und bereitgestellt werden.

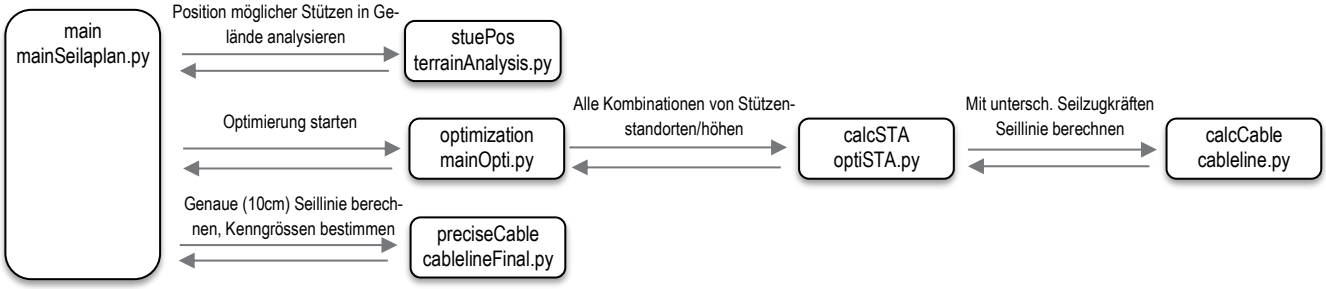
4.6. Optimierungsalgorithmus

Der eigentliche Optimierungsalgorithmus wird in einem separaten Thread in der Skriptdatei *processingThread.py* ausgeführt. Würde man die Optimierung in der Methode *run*() in seilaplanPlugin.py ausführen, würde bei der länger dauernden Berechnung QGIS in der Zwischenzeit einfrieren. Damit QGIS aktiv und ansprechbar bleibt, wird der Optimierungs-Algorithmus an den QGIS Taskmanager übergeben und separat ausgeführt.

Die Optimierung beginnt in der Skript-Datei mainSeilaplan.py. Diese führt zuerst eine Analyse des Terrains durch um mögliche Stützenstandorte entlang des Profils ausfindig zu machen. Anschliessend wird im Skript mainOpti.py das Optimierungsproblem formuliert, d.h. alle möglichen Varianten zusammengetragen. Die Varianten werden in optiSTA.py durchgerechnet und geprüft, d.h. es wird als erstes die Anfangsseilzugkraft bestimmt, welcher für den betrachteten Abschnitt möglich ist und dann die Seillinie im Skript cableline.py bestimmt.

Sind alle Varianten durchgerechnet, wird in mainOpti.py die Kombination der besten Varianten durch eine Least-Cost-Berechnung bestimmt. Die Lösung beinhaltet die Position und Höhe der Stützen und die optimale Seilzugkraft. Diese

Werte werden zurück an mainSeilaplan.py geschickt, wo die Seillinie ein letztes Mal mit hoher Auflösung (10cm) gerechnet wird (cablelineFinal.py).



4.7. Output-Erstellung

Beim Speichern der Resultate hat der Benutzer verschiedene Output-Produkte zur Auswahl. Das PDF-Diagramm zeigt denselben Plot wie im Bearbeitungsfenster. Für die Ausgabe wird die gleiche Klasse adjustmentPlot.py verwendet und nur einige Variablen angepasst.

Für die Erstellung des detaillierten Berichts sind zwei Schritte notwendig; erst werden die Texte und Werte in strukturierten Listen vorbereitet, dies geschieht in der Funktion *generateReportTxt*(). Anschliessend werden sie an die Funktion *generateReport*() übergeben, wo mit Hilfe der Bibliothek reportlab ein PDF mit Tabellen-Struktur aufgebaut wird. Die Texte werden in die Tabelle eingefüllt und die einzelnen Zeilen gestylt. Für den Kurzbericht werden neu alle Daten in derselben Funktion aufbereitet und gleich in die reportlab Tabellenstruktur abgefüllt.

Geodaten erstellen: Die Koordinatenlisten der Seillinie werden an die Methode *generateGeodata()* übergeben, die je ein Shapefile für das Leerseil, das Lastseil und die Stützenpunkte erzeugt. Die gleichen Daten können auch in ein KML File abgefüllt werden. In der Methode *generateCoordTable()* werden die csv-Tabellen erzeugt. Die Seildaten enthalten die Horizontaldistanz ab Anfangspunkt, X- und Y-Koordinaten, die Höhe des Last- und Leerseils, sowie das Höhenprofil. Die Stützeninformationen enthalten eine Bezeichnung, X- und Y-Koordinaten, die Z-Koordinate am Boden und an der Spitze der Stütze sowie die Höhe.

4.8. Vogelperspektive

Die Vogelperspektive wird mit Hilfe von matplotlib als zweiter Subplot unterhalb der Profilansicht erstellt. Für die Vogelperspektive wird ein Hintergrundbild aus der aktuellen QGIS-Karte extrahiert. Damit das Hintergrundbild exakt in den Subplot passt, muss in QGIS eine Druckvorlage erstellt, die Karte geladen und dann basierend auf dem Azimut der Seillinie gedreht werden.

Die Symbole der Stützen basieren auf SVG-Symbolen, die in der Inkscape-Datei config > _pole_symols.svg abgespeichert sind. Das SVG sollte nur mit Inkscape geöffnet und editiert werden.

Arbeitsschritte um Symbole zu aktualisieren:

Inkscape

1. Layer auswählen (z.B.: „Verstaerkter ankerbaum“, ohne „onepath“ postfix)
2. Linien, Kreise, etc. als Vektorgrafiken editieren (In der Inkscape Sprache sind dies „Strokes“ und „Objects“)
3. Layer duplizieren, Postfix „onepath“ anhängen (alter Layer löschen)
4. Zuerst alle Krise auswählen, Menü Path > Object to Path

5. Alle Linien auswählen, im Menü Path > Stroke to Path
6. Alle Paths auswählen, im Menü Path > Union → Mehrere Paths werden zu einem Path
7. Mögliche Translations und Scales entfernen: Rechtsklick > Group, dann Rechtsklick > Ungroup
8. Im XML-Editior den Pfad kopieren

In birdViewSymbol.py

9. SVG-Pfad zuoberst im Python File einfügen
10. Skript direkt laufen lassen: Sicherstellen, dass der Python Interpreter das Package „svg2mpl“ installiert hat
11. Die Symbole werden in einem plot dargestellt. Eventuell den Grössenfaktor (scale) in der Variable „SVG_PROPS“ anpassen, damit das Symbol gleich gross ist wie die bereits existierenden.
12. Das SVG Symbol wird zu einem matplotlib Path Objekt transformiert, dann als Numpy Text-File in ordner config > birdViewSymbols abgespeichert.
13. Wenn Seilaplan das nächste Mal ausgeführt wird, wird das aktualisierte Symbol im Numby Textfile benutzt.

Zusätzlich muss das angepasste Symbol in der Infografik aktualisiert werden. Dazu den Inkscape Layer als PNG mit Dimension 189x189 pixel / 96 dpi exportieren. Die Infografik ist ein Google Tabellendokument und im Seilaplan Google Drive abgespeichert.