Informatik II
SS19
GST.413UF

Programm 1
Drohnengetragenes Laserscanning
Technischer Bericht

Gruppe A
Paul Arzberger
00311430

Hiermit bestätige ich, die Ergebnisse nach besten Gewissen selbstständig erstellt und interpretiert zu haben.

Paul Arzberger

Graz am
30.04.2018

1. Entwicklungsumgebung

Die Software wurde unter den Betriebssystemen *Windows 7 64bit* und *Windows 10 64bit* mit der *Python Interpreter Version 3.7.0 64bit* entwickelt und getestet. Als IDE diente *PyCharm 2019.1.1 (Professional Edition)*. Zur Grafikerstellung wurde *matplotlib* unter der Version 3.0.2 verwendet. Weiters wurden *SciPy* 1.2.1 und *NumPy* 1.15.2 verwendet. Diese Pakete werden nicht vom Interpreter mitgeliefert und müssen seperat mittels *pip* installiert werden.

2. Aufgabenstellung

Ziel dieses Programmes ist es das Wissen über objektorierntierte Programmierung in der Sprache Python zu vertiefen und zu festigen. Hierfür soll ein Programm geschrieben werden welches Tachymeterbeobachtungen und die Daten eines drohnengetragenen Laserscanners auswertet und visualisiert. Der Tachymeter verfolgte das Drohnensystem, welches einer vorgegebenen Trajektorie folgte und nahm in regelmässigen Zeitabstände Winkel und Distanzmessungen zu diesem auf um die genaue Trajektorie dessen zu bestimmen. Der Laserscanner selbst nahm ebenfalls Distanz und Winkelmessungen nach untengerichtet von der Drohne aus, vom Gebäude der Steyrergasse 30 auf. Aus diesen Daten soll nun ein digitales Geländemodell des aufgenommenen Gebäudes erstellt werden.

3. Programmstruktur

Der Quellcode des Programmes befindet sich in einem Pythonfile namens *main.py*. Hier befinden sich alle Klassen- und Funktionsdefinitionen sowie der Code des Hauptprogrammes zur Datenauswertung und Visualisierung. Das Programm wurde unter der Python *Interpreterversion 3.7.0 64bit* geschrieben und auf *Windows 7 64bit* sowie *Windows 10 64bit* getestet

3.1 Verwendete Bibliotheken

Um das Programm zum Laufen zu bringen, werden einige zusätzliche Bibliotheken benötigt:

- Scipy 1.2.1
- Numpy 1.15.2
- matplotlib 3.0.2

3.2 Daten

Die Daten der Tachymeter- und Laserscannerbeobachtungen sind in zwei Textfiles *obsDrone.txt* und *obsTachy.txt* verspeichert und weisen beidemale die selbe Gliederung der Spalten auf:

Zeit in Sekunden, Distanz in Meter, Zenitwinkel in Radiant, Azimutwinkel in Radiant

3.3 Workflow und Vorgehensweise

3.3.1 Klassen und Datenauslese

Jede Beobachtung des Laserscanners und des Tachymeters sollen mit bestimmten Klassen abgebildet und mittels derer Memberfunctions und Attributen verarbeitbar gemacht werden. Ausgangspunkt ist die Klasse *Epoch* welche als Basisklasse für die weiteren Klassen *PolarEpoch* und *PositionEpoch* dient. Epoch besitzt, das an alle erbenden Klassen weitergegebene Attribut *time* vom Typ *datetime* des Pythonmoduls *datetime*, welches auch bei der Instanzierung eines neuen Objektes angegeben werden muss. Ist der übergebene Wert für *time* kein Wert des Types *datetime*, wird ein *ValueError* geworfen. Das Attribut *time* ist als *Property* realisiert. Die ausgelesene Daten der TXT-Files sollen als *PolarEpoch* verspeichert werden und die 3D Koordinatenpositionen der Drohnentrajektorie als Instanzen der Klasse *PositionEpoch* verspeichert werden. Das Auslesen und verspeichern der Messepochen auf Instanzen der Klasse *PolarEpoch* findet in einer Funktion names *read_obs_data* statt und verlangt nach den Übergabeparametern:

- Pfad des auszulesenden Files
- Beginn des Aufnahmezeitpunktes
- optional - einen boolschen Wert welcher defaultmässig auf False gesetzt ist um die Instanzierung und Elimination von der unterschiedlichen Instanzen auf den Schirm auszugeben oder auszublenden.

Bei der Instanzierung einer PolarEpoch-Instanz müssen der Aufnahmezeitpunkt vom Typ *datetime*, die Distanz als *float*, der Zenitwinkel als *float* und der Azimuthwinkel als *float* mit übergeben werden.

Der Aufnahmezeitpunkt einer Beobachtung leitet sich relativ vom Beginn der Beobachtungen ab. In den TXT Daten findet er sich in der ersten Spalte als Sekundenwert. Um nun einen eindeutig und vergleichbaren *datetime*-Value zu erstellen, wird bei der Auslese der Daten der Beginn der Aufzeichnung vom Typ *datetime* in die Funktion übergeben, dort in Sekunden umgerechnet und jeder Epoche hinzuaddiert und wieder in eine "menschlesbare" Form zurück verarbeitet. Dies ist insofern wichtig da jede *PositionsEpoch*-Instanz der Drohnentrajektorie eindeutig allen Bodenpunktmessungen des Laserscanners zugeordnet werden müssen.

3.3.2 Sortierung und 1. Hauptaufgabe

Die Tachymeterinstanzen müssen noch sortiert werden, da sie aufgrund eines Softwarefehlers nicht in der korrekten Reihenfolge in das TXT exportiert worden sind. Dies übernimmt ein *bubblesort* Algorithmus implementiter in der Funktion *bubble_sort()*. Diese sortierten *PolarEpoch* Instanzen sind als Liste verspeichert und werden mittels einer Listkomprehension in eine Liste PositionEpochInstanzen übergeführt. Für die Instanzierung einer PositionEpoch Instanz sind die Zeitinformation der zugehörigen PolarEpoch von Nöten, sowie die umgerechneten 3D Kooridinaten der PolarEpoch Instanz. Hierzu werden drei Memberfunktionen der PolarEpoch Instanz aufgerufen (*x_1ha*, *y_1ha*, *z_1ha*) um die X, Y

und Z Koordinaten relativ zum Ausgangspunkt (Standort des Tachymeter als PositionEpoch Instanz), welcher als Input übergeben wird zu berechnen.

3.3.3 Zuordnung der Tachymeterepochen den Laserscannerepochen

Um die gemessenen Bodenpunkte des Laserscanners nun den Trajektorienepochen zuzuordnen, wurden in einer Schleife die Trajektorienpositionen durchiteriert und die Liste mit den gemessenen Bodenpunkten nach ihrem Aufnahmezeitpunkt gefiltert. Geschehen ist mittels einer Function von Python namens *filter*. *filter* erzeugt eine Liste an Elementen für die eine bestimmte Bedingung oder Funktion als wahr angesehen wird. Die Funktion selbst ist eine *Lambda-Funktion*. Dadurch erhält man eine Subliste aller zutreffenden Instanzen des Laserscanners und kann falls mehr als eine Messung getätigt wurde eine Mittelung zum jeweiligen Messzeitpunkten vornehmen. Da in diesem Fall nur der gemessene Abstand vom Boden zum Laserscanner interessiert, wird nur der Teil der 1 geodätischen Hauptaufgabe ausgeführt der die Z Koordinate betrifft. Die korrespondierenden X und Y-Koordinaten der Trajektorie und der Z-Wert des gemessenen Bodenpunktes werden nun jeweils in eigene Listen gespeichert um für die Visualisierung herangezogen werden zu können.

3.3.4 Rasterisierung der Trajektorie

Die gemessenen Bodenpunkte sollen nun gegittert in der Auflösung 0,5 x 0,5 Meter abgespeichert und visualisiert werden. Da die Drohne einer gegebenen Trajektorie folgt und ab einem bestimmten Bereich eine 180° Kurve fliegt, um das Gebäude zeilenmässig abzuscannen, mussten all jene Laserscannerepochen von jenen unterschieden werden, bei denen der Laserscanner aktiv war. Dies geschieht in dem zwei boolsche Variablen (*start_recording, stop_recording*) auf True oder False gesetzt werden. Das Programm erkennt anhand der Anzahl in der gefilterten Subliste der Laserscanner PolarEpochen ob der Laserscanner aktiv ist oder gerade nicht aufnimmt. Hier zu wurde ein Logikteil eingebaut welcher den Aufnahme- oder Nichtaufnahmzustand logisch abbilden kann. Dies wird dazu verwendet alle betreffenden Z-Werte eines Bodenpunktes in eine Liste zu speichern und diese einem Dictionary anzuhängen. Mittels einem mitlaufendem Zähler und einer Modulu-operation auf gerade Zahlen, kann man recht einfach feststellen welcher Flugzeile der Trajektorie die Daten zugehörig sind und sie notfalls "umdrehen" da die Drohne nach jeder absolvierten Flugzeile eine 180° Kurve fliegt. Der mitlaufende Zeilenzähler dient auch als Key für das Dictionary, in welches nach Flugzeilen nummeriert die Bodenpunktdaten als Liste verspeichert werden. Die Daten des Dictionary werden mittels einem Schleifenkonstrukt in einen numpy array eingetragen und können nun zur Visulisierung als Raster herangezogen werden.

3.3.5 Resampling des Rasters auf 0,5x0,5 Meter Auflösung

Der Abstand der Laserscannerzeilen beträgt um die 5,5 Meter. Dies würde wenn man einen Raster aus den gescannten Zeilen erstellt, eine Auflösung von 5,5m in der horizontalen Richtung und 0,5m Auflösung in der Vertikalen sowie ein recht schwer interpretierbares Bild ergeben. Daher bedient man sich Interplationsmethoden um zwischen den getätigten Messpunkten zu interpolieren und die Auflösung zu erhöhen.

3.3.6 Plotten der Messungen und der Trajektorie

Für die Visualisierung wird *matplotlib* verwendet. Für die Darstellung der Drohnentrajektorie wird eine allgemeiner Plot verwendet. Das Höhenmodell wird als 3D Scatterplot, Trisurface sowie als Raster dargestellt.

3.3.7 Klasse Timer - Laufzeit stoppen

Mittels einer eigenen Klasse *Timer* welche die Kontextmanager Methoden*__enter__* und *__exit__* überschrieben hat, ist die Software in der Lage die Laufzeit des Programmes zu messen wenn man die Klasse mit dem Keyword *with* initialisiert und den restlichen Programmcode innerhalb des with-Blocks verfasst. Beim Verlassen des Blocks oder Beenden des Programms wird die Laufzeit des Programms am Schirm ausgegeben.

4. Implementierung

4.1 Klassen

4.1.1 Epoch

Die Klasse *Epoch* ist die Basisklasse der erbenden Klassen *PolarEpoch* und *PositionEpoch*. Sie besitzt ein Attribut *time* welches vom Python *datetime* Format stammen muss um einen gültigen Wert auf die Instanz zu schreiben. Dies wird auch mittels *isinstance* überprüft. Zusätzlich wurde ein weiteres privates Attribut implementiert - *__verbose*. Wenn eine Instanz instanziert wird sind einerseits bei PolarEpoch und PositionEpoch der Messzeitpunkt und Lageangaben ein Muss, man kann aber optional den Parameter verbose auf True setzen und somit Nachrichten über den Lebenszyklus einer Instanz am Bildschirm ausgeben lassen. per Default ist *__verbose* auf False gesetzt um ein Screenmasaker bei den doch tausenden PolarEpoch Instanzen zu vermeiden. *__verbose* und *__time* sind als Properties implementiert.

4.1.2 PolarEpoch

PolarEpoch erbt von der Basisklasse *Epoch*. Wird eine *PolarEpoch* Instanz instanziert benötigt man einen aktuellen Zeitstempel, eine Distanzinformation, eine Höhen-winkelinformation und eine Azimuthinformation. Da die Position der Drohne relativ zum Tachymeter (welcher einen festen Standort hat) berechnet werden sollen, wurde der Klasse drei Memberfunktionen (*x_ha1, y_ha1, z_ha1*) implementiert welche den X, Y und Z-Wert bei einer Ersten Hauptaufgabe ermitteln. Als Inputparameter dieser Memberfunktionen dient immer ein Objekt der Klasse PositionEpoch. Die Formeln wurden wie folgt verwendet (sowohl bei Tachy-->Drohne als auch Drone-->Bodenpunk)t:

$$x_2 = x_1 + r \cdot \sin\vartheta \cdot \cos\varphi$$
$$y_2 = y_1 + r \cdot \sin\vartheta \cdot \sin\varphi$$
$$z_2 = z_1 + r \cdot \cos\vartheta$$

4.1.3 PositionEpoch

PositionEpoch erbt von der Basisklasse *Epoch*. Die Klasse selbst benötigt bei der Instanzierung eine Zeitinformation im *datetime*-Format, eine X-, Y- und Z-Koordinate welche als Attribute verspeichert werden. Der Zugriff erfolgt über definierte Properties.

4.1.4 Timer

Die *Timer* Klasse wird dafür verwendet, um die Laufzeit des Programmes zu analysieren. Hierfür wurden die *__enter__* und *__exit__* Methoden überschrieben, welche unter Verwendung eines Contextmanagers mit dem Statement *with* aufgerufen werden. Wird ein Textblock mit dieser Klasse und dem with Statement erstellt, beginnt sobald die erste Zeile Code inerhalb des Textblockes ausgeführt wird die Zeitnehmung. Bei Verlassen des Textblockes wird die Zeit erneut genommen und eine Differenz von Anfangs und Endzeit gebildet und ausgegeben. Die Anfangs und Endzeit sind als private Attribute *__starttime* und *__endtime* in der Klasse zu finden.

5. Ergebnisse

5.1 Darstellung und Präsentation

Abbildung 1 zeigt ein abgeleitetes 3D Model des Gebäudes der Steyrergasse 30 ohne die Bodenpunkte zu mitteln. Eine Version des 3D Models mit gemittelten Werten zeigt nur geringfügig Unterschied zu jenem mit voller Auflösung. Auf der rechten Seite zeigt die Abbilung 1 eine Scatterdarstellung der gemittelten Bodenpunkte inkl einer Trajektoriendarstellung der Drohne.

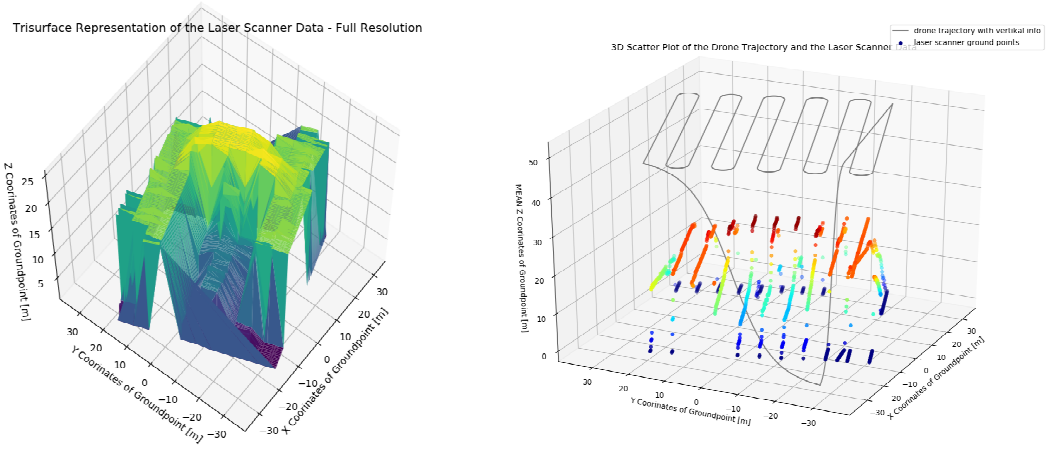

Abbildung 1 Links: 3D Modell Steyrergasse 30 - Rechts: Scatterdarstellung der Laserscannermessungen entlang der Dronentrajektorie

Die Rasterung der aufgenommenen Messungen des Laserscanners wurden mittels einem linearen Interpolationsverfahren in Spaltenrichtung auf 0,5m Auflösung hochgerechnet. Sofort sieht man den Informationsgewinn im Vergleich zum Ausgangsraster

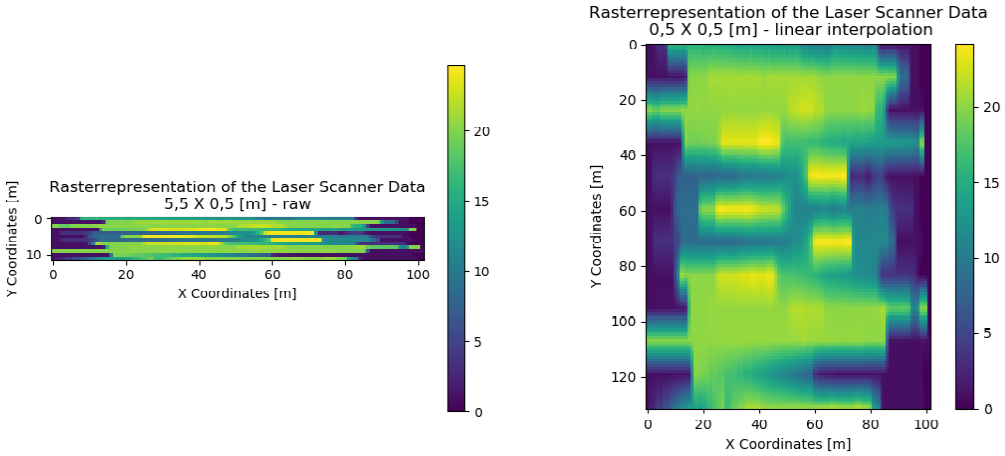

Abbildung 2 Links: 5,5x0,5m Auflösung - Rechts: Lineare Interpolation auf 0,5x0,5 m Auflösung

Eine cubic-spline Interpolation wurde ebenfalls durchgeführt, weist aber im Vergleich zur linearen Interpolation mehr Rauschen auf, eine Mittelung beider Verfahren kann auch herangezogen werden wurde aus Zeitgründen aber nicht mehr analysiert.

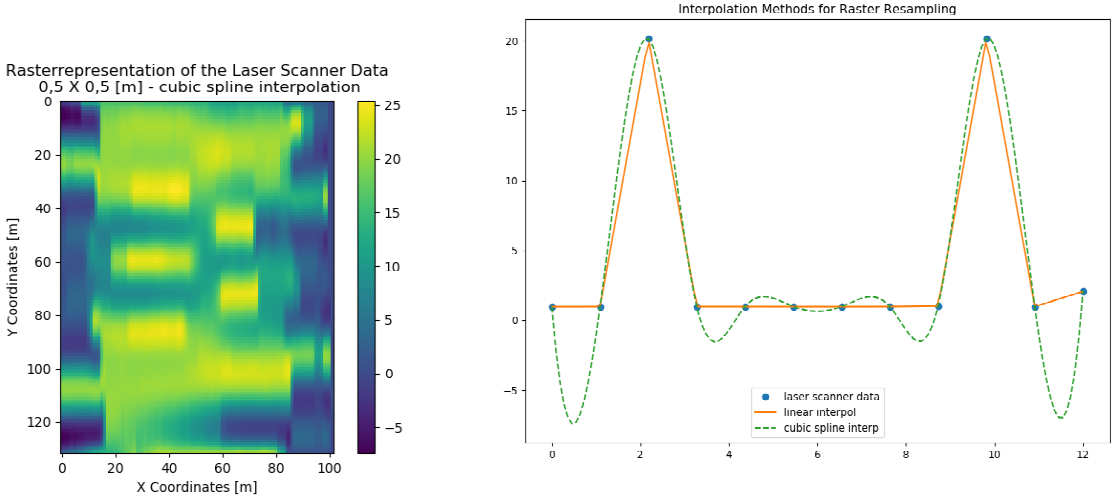

Abbildung 3 Links: Cubic Spline Interpolation der Laserscannermessungen auf 0,5x0,5m - Links: vergleich der Interpolationsverfahren anhand einer Spalte aus dem 5,5x0,5 m Raster

5.2 Zusammenfassung

Das Programm ist in der Lage aus einer Kombination von Tachymeter- und mobilen Laserscannerbeobachtungen ein digitales Oberflächenmodell zu erstellen, zu visualisieren und objektorientierte Programmierparadigmen umzusätzen.