Leopold-Franzens-Universität Innsbruck
Institut für Geographie
Innrain 52f
6020 Innsbruck

Geoinformatik: Python

Rasterkarte zur Kronendach-Durchlässigkeit im Wald

28. Februar 2020

Kursleitung:

PD Dr. Martin Rutzinger und Dipl.-Geogr. PhD Magnus Bremer

Wintersemester 2019/2020

Verfasser:

Alexander Höfner
Andreas Summer
Ludwig Hagelstein

1. Idee & Ausgangsdaten

Im Rahmen des Wahlmoduls ‚Geoinformatik: Python' soll die Kronendach Durchlässigkeit eines Waldstücks anhand von LiDAR Punktwolkendaten ermittelt und visualisiert werden. Hierfür wurde uns eine LiDAR Punktwolke zur Verfügung gestellt, welche eine Waldfläche von knapp 5,5 Hektar Wald oberhalb des Innsbrucker Stadtteils Hötting abdeckt. Die Daten sind im LAS-Format 1.1 vorklassifiziert (siehe Tabelle 1).

Tabelle 1 Punktwolkenklassifizierung im LAS-Format 1.1

KLASSIFIZIERUNGSWERT	BEDEUTUNG
0	Nie klassifiziert
1	Nicht zugewiesen
2	Erde
3	Niedere Vegetation
4	Mittlere Vegetation
5	Hohe Vegetation
6	Gebäude

2. Methodik

Die Untersuchungen basieren auf der Annahme, dass bei einem durchlässigen Kronendach viele Bodenpunkte und wenig Vegetationspunkte in der Punktwolke enthalten sind, bei einem undurchlässigen Kronendach dagegen genau umgekehrt. Die Punktwolke wird daher in Rasterzellen von 1 m Auflösung eingeteilt und mithilfe zweier Indizes untersucht.

2.1. Verwendete Python Module

Folgende Python Module wurden für die Untersuchungen verwendet:

- Gdal: Rasterverarbeitung und Rastererstellung
- Osr: Geodatenverarbeitung
- NumPy: Für mathematische Fragestellungen und für Arrays
- SciPy: Für die Interpolation der Bodenpunkte zu einem DGM

2.2. Script

Das Script wurde in Microsoft Visual Studio Code und mithilfe der Versionskontrolle von GitHub erstellt, sodass alle Gruppenmitglieder stets am neuesten Stand des Projekts weiterarbeiten konnten.

Eingangs wurden die zur Verfügung gestellten Punktdaten eingelesen (Abb. 1). Für Testzwecke wurden lediglich 1000 bis 10000 Punkte eingelesen, um den Rechenaufwand zu reduzieren.

```
#--------------------------------
#File einlesen:
#--------------------------------
fobj = open('Daten/Waldpunktwolke.txt', "r") #Input File
InputSize = "full"

pointlist = []

next(fobj)
for line in fobj:
    line = line.strip()
    line= line.split("\t")
    pointlist.append(line)
fobj.close
```

Abb. 1: Einlesen der Punktdaten

Anschließend wurden die eingelesenen Daten in ein Array gespeichert. Ein neuer Array, der lediglich die Vegetationsklassen beschreibt („vegarray“) und ein Bodenarray („bodenarray“) wurden erstellt (Abb. 2).

```
#------------------------------
#Daten einlesen und nach Typ Filtern:
#------------------------------

pointarray = np.array(pointlist)
pointarray = pointarray.astype(float)
print(len(pointarray))

#Neuer Array ohne Gebauede und Boden, Klassen 3, 4, 5
vegetationslist = []
bodenlist=[]

print("Vegetations-und Bodenliste einlesen:")
printProgressBar(0, len(pointarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
j=0
for i in pointarray:
    printProgressBar(j+1, len(pointarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
    j+=1
    if i[6] == 3 or i[6] ==4 or i[6] ==5:
        vegetationslist.append(i)
    if i[6] == 2:
        bodenlist.append(i)

vegarray = np.array(vegetationslist)
bodenarray = np.array(bodenlist)
```

Abb. 2: Erstellen eines Vegetations- und eines Bodenarrays

Als nächstes wurden die maximale und minimale Ausdehnung der Punktdaten in X-Y-Richtung zur Erstellung eines leeren Arrays mit der Dimension der Punktdaten (Abb. 3) ermittelt.

```
#-----------------------------------
#Extend Array with random numbers:
#-----------------------------------
#max values
maxvals = np.amax(pointarray,axis=0)
xmax=maxvals[0]
ymax=maxvals[1]
# print ("xmax:",xmax)
# print ("ymax:",ymax)

#min values
minvals = np.amin(pointarray,axis=0)
xmin=minvals[0]
ymin=minvals[1]
# print ("xmin:",xmin)
# print ("ymin:",ymin)

#empty array with extent dimensions
nrows= int(np.ceil(ymax-ymin))
ncols= int(np.ceil(xmax-xmin))
z=2
```

Abb. 3: X-Y-Ausdehnung der Punktdaten

Im Folgenden wurden die verschiedenen Arrays für weitere Berechnungen erstellt (Abb. 4).

```
# Erstellen der verschiedenen Arrays

bodencountout_array = np.empty((nrows,ncols))
bodencountout_array.fill(0)

bodenhoehencountout_array = np.empty((nrows,ncols))
bodenhoehencountout_array[:] = np.NaN

bodenhoehen_array = np.empty((nrows,ncols))
bodenhoehen_array[:] = np.NaN

vegcount_array = np.empty((nrows,ncols))
vegcount_array.fill(0)

vegcount_higher2_array = np.empty((nrows,ncols))
vegcount_higher2_array.fill(0)

veghohen_arry= np.empty((nrows,ncols))
veghohen_arry[:] = np.NaN

veghigher2_hoehen_array  = np.empty((nrows,ncols))
veghigher2_hoehen_array[:] = np.NaN

vegpoints_array = np.empty((nrows,ncols))
vegpoints_array[:] = np.NaN

bodenpoints_array = np.empty((nrows,ncols))
bodenpoints_array[:] = np.NaN

veghigher2_count_array = np.empty((nrows,ncols))
veghigher2_count_array[:] = np.NaN
```

Abb. 4: Erstellen der verwendeten Arrays

Danach wurden die einzelnen Bodenpunkte je Zelle gezählt und in einen neuen Array geschrieben (Abb. 5).

```
#-----------------------------------
#Bodenpunkte zaehlen:
#-----------------------------------
#Cellsize = float(raw_input("Bitte geben Sie die gewuenschte Rasterzellengroesse an:")) #Raw Input funktioniert in VisualStudio nicht
Cellsize = 1

print("Bodenpunkte in Rasterzellen aufteilen und Mittelwert pro Zelle berechnen:")
printProgressBar(0, len(bodenarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
j=0
for i in bodenarray:
    printProgressBar(j+1, len(bodenarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
    j+=1
    x=i[0]
    y=i[1]
    height = i[2]
    gx = int((x -xmin)/Cellsize)
    gy = int((y -ymax)/-Cellsize)
    bodencountout_array[gy][gx] +=1
    bodenhoehencountout_array[gy][gx] +=1
    if np.isnan(bodenhoehencountout_array[gy][gx]) == True:
        bodenhoehen_array[gy][gx] = i[2]
    else:
        bodenhoehen_array[gy][gx] = np.mean([bodenhoehen_array[gy][gx],i[2]])
```

Abb. 5: Zählen der Bodenpunkte

Die lückenhaften Bodenpunkte wurden als nächstes über die Fläche interpoliert, um ein lückenloses DGM zu erhalten (Abb. 6).

```
##### Boden interpolation
##Dokumention : "https://modelhelptokyo.wordpress.com/2017/10/25/how-to-interpolate-missing-values-2d-python/"
a = np.arange(0, ncols)
b = np.arange(0, nrows)
#mask invalid values
array = np.ma.masked_invalid(bodenhoehen_array)
xx, yy = np.meshgrid(a, b)
#get only the valid values
x1 = xx[~array.mask]
y1 = yy[~array.mask]
newarr = array[~array.mask]

GD1 = interpolate.griddata((x1, y1), newarr.ravel(),
                           (xx, yy),
                              method='cubic')

print (GD1)
```

Abb. 6: Interpolation der Bodenpunkte

Um lediglich die Vegetationspunkte über 2m Vegetationshöhe herauszufiltern, wurde die Höhendifferenz zwischen Punkt und interpolierter DGM-Höhe ermittelt (Abb. 7).

```
#-----------------------------------
#Mittlere Hoehe der VegPunkte:
#-----------------------------------
print("Vegetationspunkte in Rasterzellen aufteilen und erstellen eines Array mit Punkten uber 2 Meter:")
printProgressBar(0, len(vegarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
j=0
for i in vegarray:
    printProgressBar(j+1, len(vegarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
    j+=1
    x=i[0]
    y=i[1]
    height = i[2]
    #print(height-GD1[gy][gx])
    gx = int((x -xmin)/Cellsize)
    gy = int((y -ymax)/-Cellsize)
    vegcount_array[gy,gx]+=1
    veghohen_arry[gy,gx] = (i[2]-GD1[gy][gx])
    if height-GD1[gy][gx] >= 2:
        vegcount_higher2_array[gy,gx]+=1
        veghigher2_count_array[gy][gx]+=1
        if np.isnan(veghigher2_count_array[gy][gx]) == True:
            veghigher2_hoehen_array[gy][gx] = (i[2]-GD1[gy][gx])
        else:
            veghigher2_hoehen_array[gy][gx] = np.mean([veghigher2_hoehen_array[gy][gx],(i[2]-GD1[gy][gx])])
```

Abb. 7: Vegetationspunkte über 2m Höhe

Abschließend wurden zwei Indizes berechnet: Index 1 als Verhältnis aller Vegetationspunkte zu den Bodenpunkten und Index 2 als Verhältnis aller Vegetationspunkte über 2m zu den Bodenpunkten (Abb. 8).

```
#--------------------------------
#Index anzahl veg zu bodenpunkte veg/(veg-boden):
#--------------------------------
np.seterr(divide='ignore', invalid='ignore')
indexarray1 = np.empty((nrows,ncols))
indexarray1 = (vegcount_array/(vegcount_array+bodencountout_array))



# %%
#--------------------------------
#Index anzahl veg ueber 2 m zu bodenpunkte veg/(veg-boden):
#--------------------------------
np.seterr(divide='ignore', invalid='ignore')
indexhoehen2m = np.empty((nrows,ncols))
indexhoehen2m = (vegcount_higher2_array/(vegcount_higher2_array+bodencountout_array))
```

Abb. 8: Berechnung Index 1 und Index 2

Danach wurden die einzelnen Arrays für die Visualisierung in Rasterdaten konvertiert und georeferenziert (Abb. 9).

```
#--------------------------------
#OutRaster schreiben:
#--------------------------------


#Boden Raster
driver = gdal.GetDriverByName("GTiff")
dataset = driver.Create("Export/BodenCount_%s.tif"  % (InputSize), ncols, nrows, 1, gdal.GDT_Float32)
dataset.SetGeoTransform((xmin,1,0,ymax,0,-1))

#Koordinatensystem definieren:
dstSRS = osr.SpatialReference()
dstSRS.ImportFromEPSG(32632)
dest_wkt = dstSRS.ExportToWkt()

dataset.SetProjection(dest_wkt)

#Raster ausgeben:
bandout = dataset.GetRasterBand(1).WriteArray(bodencountout_array)
dataset.FlushCache()
```

Abb. 9: Export eines Arrays als Raster

Ladebalken

Als praktisches Feature wurde zudem ein Ladebalken integriert, der den Fortschritt der teilweise zeitaufwändigen Berechnungen zeigt (Abb. 10).

```
print("Vegetations-und Bodenliste einlesen:")
printProgressBar(0, len(pointarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
j=0
for i in pointarray:
    printProgressBar(j+1, len(pointarray), prefix = 'Progress:', suffix = 'Complete', length = 50)
```

Abb. 10: Ladebalken

Dieser wird bei der Ausführung des Skripts, wie in folgender Abbildung (Abb. 101) gezeigt, dargestellt

```
Vegetations-und Bodenliste einlesen:
Progress: |████------------------------------------| 20.4% Complete
```

Abb. 11:Darstellung Ladebalken

3. Ergebnis

In Abb. 12 wird das Verhältnis von Vegetation- zu Bodenpunkten pro Rasterzelle dargestellt. Bei Index 1 werden dabei alle Vegetationspunkte berücksichtigt, wohingegen bei Index 2 nur die Vegetationspunkte >2m Bodenhöhe einbezogen wurden. Man kann z.B. in der oberen rechten Ecke der Abbildung gut erkennen, wie bei Index 2 der Wald mit Bäumen deutlich besser abgegrenzt werden kann gegenüber niedrigerem Buschwerk, dies ist auch in den vergrößerten Ausschnitten in Abb. 13 und Abb. 14 zu erkennen.

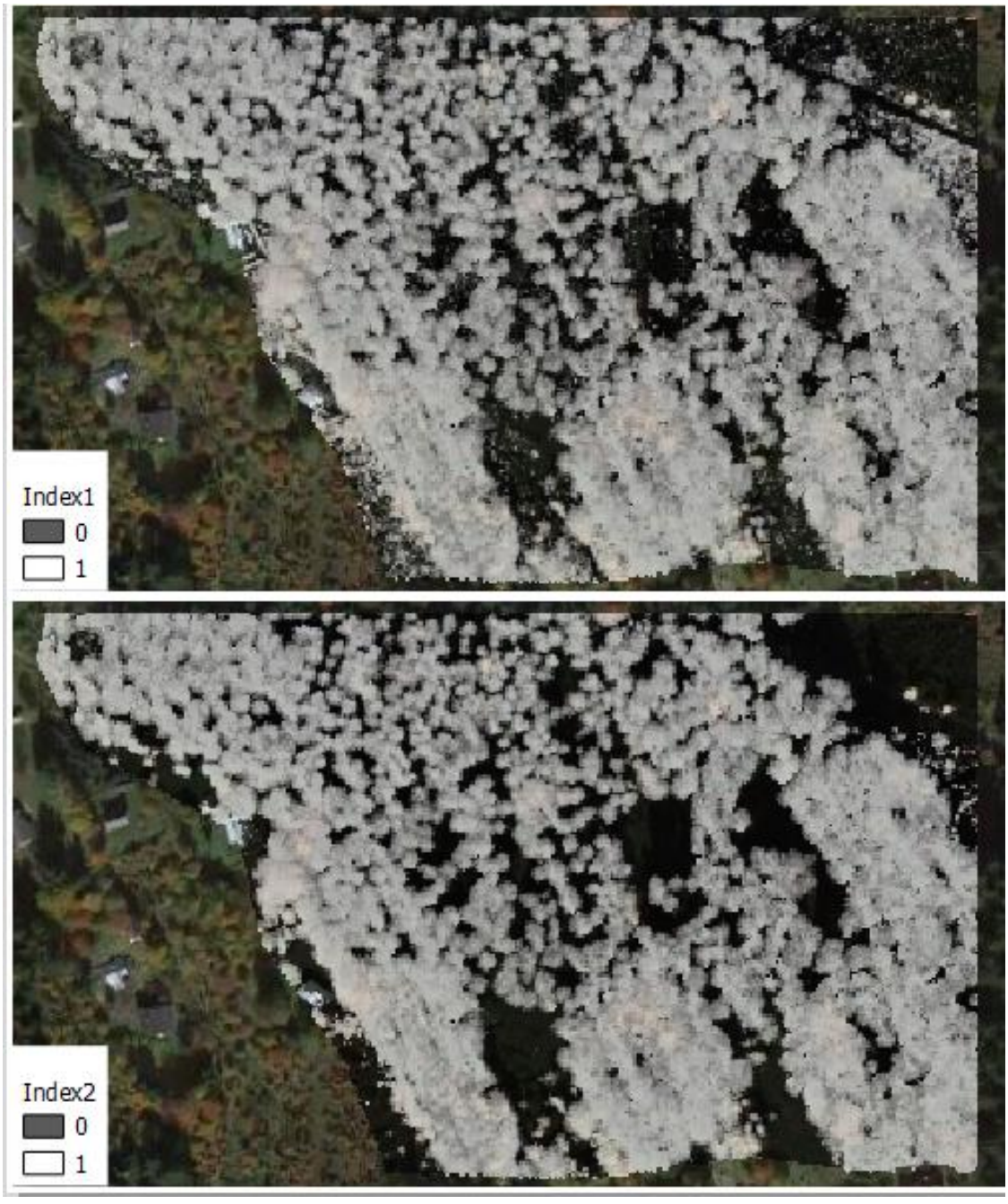

Abb. 12 Kronendachdurchlässigkeit dargestellt anhand von Index 1 (oben) und Index 2 (unten)

Abb. 13 Vergrößerter Ausschnitt Index 1

Abb. 14 Vergrößerter Ausschnitt Index 2

Um die Durchlässigkeit des Kronendachs der Bäume noch genauer abzugrenzen wurden lediglich Indexwerte >0.5 dargestellt, da nach optischer Überprüfung davon ausgegangen werden kann, dass bei Indexwerten < 0.5 die Durchlässigkeit so hoch ist, dass dort keine ernstzunehmende Abschattung durch Bäume zu erwarten ist.

Abb. 15 Vergrößerter Ausschnitt Index 2, >0.5

In Abb. 16 ist die Kronendach Durchlässigkeit mit Index 2 für die gesamte Punktwolke dargestellt, die Lichtungen im Wald sind deutlich zu erkennen, ebenso Flächen mit besonders undurchlässigem Kronendach. In Abb. 17 ist zudem die Bodenhöhenverteilung eingeblendet, ermittelt aus Interpolation der Bodenpunkte.

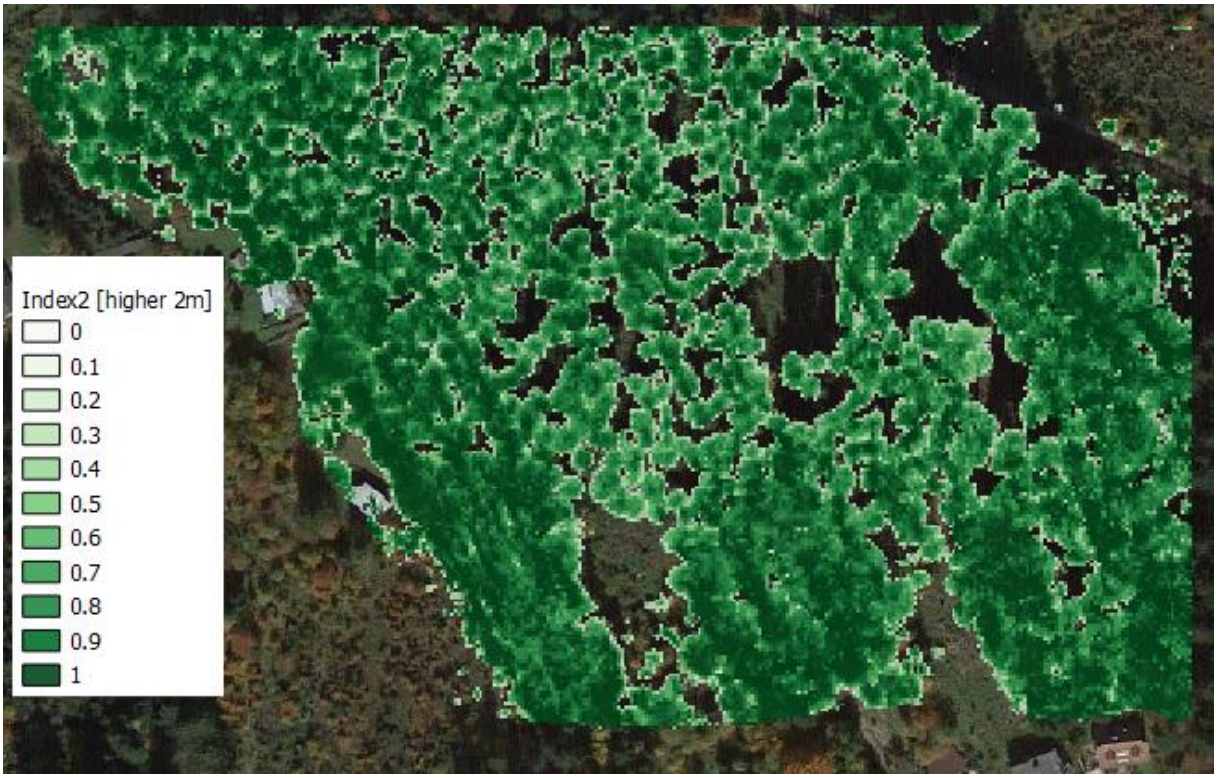

Abb. 16 Index 2 mit Orthophoto

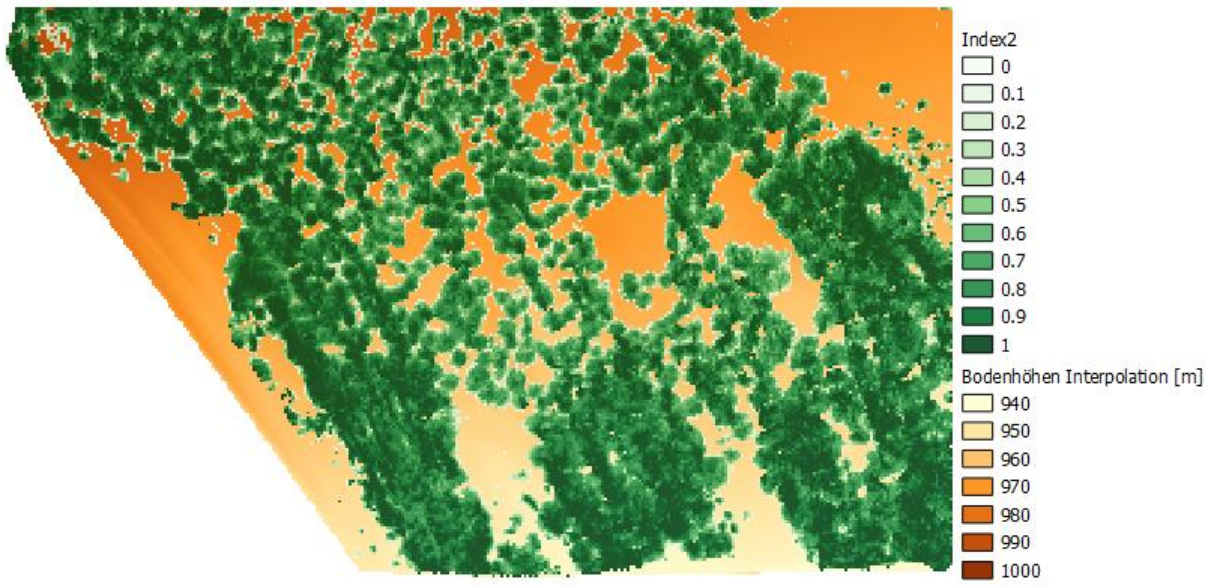

Abb. 17 Index 2 mit interpoliertem DGM

4. Diskussion & Fazit

Die Durchlässigkeit der Kronendachs kann mithilfe einer LiDAR Punktwolke gut quantifiziert und visualisiert werden. Wichtig ist jedoch eine Vorklassifizierung, um Bodenpunkte von Vegetationspunkten differenzieren zu können. Im Folgenden erwies es sich als hilfreich den Fokus auf Vegetation >2m Bodenhöhe zu legen, um ein klareres Abbild der Bewaldung zu erhalten.

Bei der Prozessierung großer Punktwolken, wie sie in diesem Projekt bearbeitet wurden erwies es sich als hilfreich den Code dahingehend zu optimieren, dass unnötige Doppelabfragen vermieden werden, um die Rechenzeit zu reduzieren.

Weiterhin wäre es interessant die Höhenverteilung der Vegetationspunkte zu untersuchen und so die Kronendachdurchlässigkeit zu ermitteln und möglicherweise Rückschlüsse auf Baumtyp ziehen zu können.

5. Literatur

Alonzo, M., Bookhagen, B., McFadden, J. P., Sun, A., & Roberts, D. A. (2015). Mapping urban forest leaf area index with airborne lidar using penetration metrics and allometry. *Remote Sensing of Environment*, *162*, 141–153.

Maltamo, M., Næsset, E., & Vauhkonen, J. (Hrsg.). (2014). *Forestry applications of airborne laser scanning: Concepts and case studies* (Softcover reprint of the hardcover 1st edition 2014). Springer.