

# Träddetektering i spårområde

## **Projektmedlem**

Niclas Hansson hansson.niclas.hansson@gmail.com

## **Projektledare**

Johan Östlund johan.ostlund@trafikverket.se

6 maj 2024

# Innehåll

<b>1</b>	<b>Introduktion</b>	<b>3</b>
1.1	Problembeskrivning . . . . .	3
1.2	Dataset . . . . .	3
1.3	Omvärldsbevakning . . . . .	3
1.3.1	Kommersiella lösningar . . . . .	4
1.4	Andra lösningar . . . . .	5
<b>2</b>	<b>Lösningsförslag</b>	<b>6</b>
2.1	Läsa datat . . . . .	6
2.2	Bygg bollträd . . . . .	6
2.3	Segmentera bort markpunkter, tygsegmentering . . . . .	8
2.4	Identifiera stampunkter . . . . .	10
2.5	Voxelisering . . . . .	11
2.6	Växtalgoritm . . . . .	13

# 1 Introduktion

Nedan följer en kort problembeskrivning vilket agerar som underlag till den slutgiltiga metod som valdes att implementeras.

## 1.1 Problembeskrivning

Varje år måste Trafikverket såga inspektera ett hundratal mil riskträd som ligger nära spårområdet. Riskträd bedöms vara de träd som är tillräckligt höga för att falla inom ett visst avstånd till spårområdet eller växer över spårområdet nära känsliga ledningar. Manuell inspektion är tidsödande och kräver mycket arbete i spårområdet. Att samla in data för automatisk detektion av riskträd som stöd för inspektionsarbetet kan korta ner den tiden som krävs att vara i spåret och kan därmed öka spårets totala tillgänglighet.

## 1.2 Dataset

Datasetet som denna studie utgår från är en Lidarscanning av en del av en längre järnvägssträcka. Algoritmen har inte testats på hela sträckan utan på ett litet delområde. Delområdet utmärks med en väldigt hög andel varierad vegetation och utöver järnvägen med tillbehör, inga eller väldigt få mänskotillverkade konstruktioner. Datasetet har en spatial 3D upplösning på ca. 5 cm, med RGBA-värden.

## 1.3 Omvärldsbevakning

En mindre litteraturstudie bedrevs för att få en fingervisning i potentiella lösningsmetodiker. Kontentan av litteraturstudien är att även om det är ett välstuderat problem så finns det fortfarande många olika lösningsmetodiker. Lösningsmetodik bör anpassas specifikt efter hur datasetet ser ut.

De forskningsartiklar som ligger till grund för detta case har ofta en av två typer av dataset:

- Träd i urbana miljöer
- Träd i odlad skog

Några utmärkande skillnader mot vårat dataset är följande:

- Odlad skog har ofta en låg andel sly. Till följd så är träden lättare att separera.
- Urban miljö har ofta tydligt separerade träd där problemet istället är omgivande konstruktioner (cyklar, bänkar, papperskorgar etc.)
- Både urbana miljöer och odlad skog har ofta relativt enhetlig uppsättning av träd, ex tallskog, ek-alle, etc.
- Data över stora skogsområden samlas ofta in med UAV, man får således en överblick av skogen där underliggande buskage och sly döljs av trädkronor i större utsträckning än de dataset som samlas in nära marken.
- Upplösningen är väldigt varierande mellan olika dataset.

### 1.3.1 Kommersiella lösningar

En av de mest populära lösningarna för liknande problem är TerraScan. TerraScan verkar lösa problemet med att identifiera träd på två sätt, trädtoppsidentifiering och stamidentifiering. Trädtoppsidentifieringen är relativt simpel: Skapa 2.5D heightmap från punktmolnet → lokalisera maxpunkter → kmeans-klustering eller vattensegmentering runt varje maxpunkt → projicera tillbaka resultatet på 3D-punktmolnet.

Identifieringen av maxpunkter kan göras på lite olika sätt. Morphologiska processer eller faltning med ett filter är vanligt. Nackdelen är att resultatet blir beroende av fönstersroeleken på filtret. D.v.s. att ett för stort filter kommer att undersegmentera slutresultatet och ett för litet filter kommer att översegmentera resultatet. M.a.o. så kan fel val av filterparametrar leda till

Stamidentifieringen har jag inte hittat något dokumentarat om men finns säkert om man rotar i deras funktioner.

## 1.4 Andra lösningar

Hittade en deep learning approach men den kräver en del träningsdata. Men man skulle kunna använde min metod som för att segmentera massa data, klicka genom resultaten och spara ned de träd som ser bra ut. Resultatet kan används som tränings/validerings-input till ett neuralt nätverk [1].

## 2 Lösningsförslag

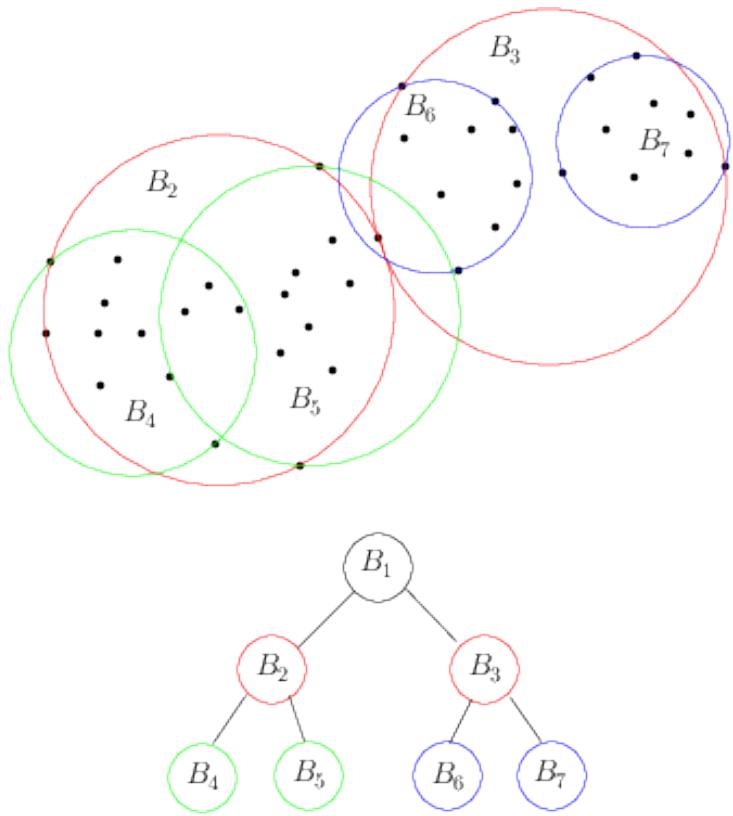
Följande lösningsförslag fokuserar på stamidentifiering. Begränsad träningsdata i liknande miljöer försvarar implementationen av mer komplexa algoritmer. Detta resultat kan istället ses som ett första segmenteringssteg för att få ut träningsdata eller som grund i en mer komplex algoritm. Algoritmen har en hel del rörliga delar men varje del är i grund och botten rätt simpel. Följande pseudokod/text sammanfattar händelseförloppet.

### 2.1 Läsa datat

LAS-data är enkelt att läsa och allt som behövs är en binär läsare för din specifika LAS-version [6]. Här har jag tidigare testat att sampla datat (för andra projekt där filerna var för stora), så det är ju något man kan prova om man vill öka hastigheten i algoritmerna då vissa skalar dåligt.

### 2.2 Bygg bollträd

Ett vanligt sätt att behandla punktmoln är att dela in dem i ett octgrid men i detta fallet har det varit fördelaktigt att använda sig av ett bollträd istället då vi gör många *närmsta-granne* och *punkter-inom-avstånd-r* sökningar [8]. Ett bollträd är en form av binärt sökträd. Du har en stor boll som innesluter hela datasetet. I den bollen finns två mindre bollar som omslutar 50% av punkterna var. I dessa bollar finns ytterliggare vars två bollar som inneslutar 25% av datasetet vardera o.s.v. Fördelen med denna metodik är att det går extremt snabbt att hitta närmsta grannar: Kolla bara i din boll, vill du ha fler grannar än de som finns i din boll: kolla de bollar som innesluter din boll. Att kolla om en boll innesluter en annan boll är väldigt simpelt. Har vi radie och position kan vi enkelt avgöra om en boll innehåller en annan boll genom en enkel vektor jämförelse.



Figur 1: Bollträde

Hur man splittar datasetet effektivt när alla punkter ligger i en oordnad lista är väl det kluriga i denna algoritmen men följande är förvånansvärt effektivt.

---

#### **Algorithm 1** Bygg bollträde

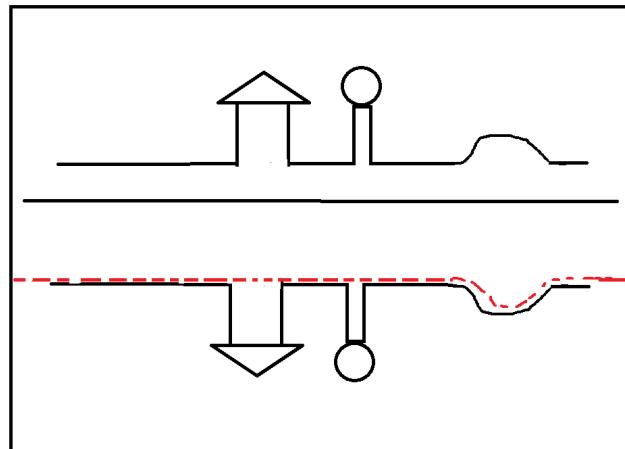
---

- 1: Ta en random punkt  $a \in \mathbb{P}$
  - 2: Hitta  $b \in \mathbb{P}$  med störst (kartesiskt) avstånd till  $a$ .
  - 3: Hitta  $c \in \mathbb{P}$  med störst (kartesiskt) avstånd till  $b$ .
  - 4: Dataspridning ligger förmögligen i riktning  $bc$
  - 5: Projicera alla punkter på vektor  $bc$
  - 6: Hitta mittpunkten
  - 7: Skapa två bollar som innesluter alla punkter
  - 8: Iterera över varje boll
- 

Algoritmen gör förvisso en alla-till-allा korrespondens, men endast en gång/boll som bildas. Datsetet blir dessutom succesivt mindre så färre och färre jämförelser görs för varje boll som byggs.

## 2.3 Segmentera bort markpunkter, tygsegmentering

Det finns många sätt att segmentera bort markpunkter på. Allra enklast är att sätta ett gränsvärde (säg 1 m över järnvägen) och plocka bort alla punkter under det gränsvärdet. Nackdel är att om geometrin inte är platt så kan det över/undersegmentera resultatet. Algoritmen jag valt här är snäppet mer komplex men rätt enkel att implementera. Tanken är att man vänder uppoch ned på datasettet och släpper ett ”tyg” över det. Tyget kommer att lägga sig utefter datasettets botten och man kan då segmentera bort alla datapunkter inom ett visst avstånd från tyget [7] [5].



Figur 2: Tygsegmentering

Fördelen med denna metod är att den är reaktivt enkel att implementera, den är väldigt effektiv och det finns gott om dokumentation över hur man kan optimera den. Parametrar bestämmer tygets ”styrhet” och hur väl den ska följa konturerr eller falla igenom hål. Algoritmen kan utökas om man är intresserad av att hitta t.ex. kantsten.

---

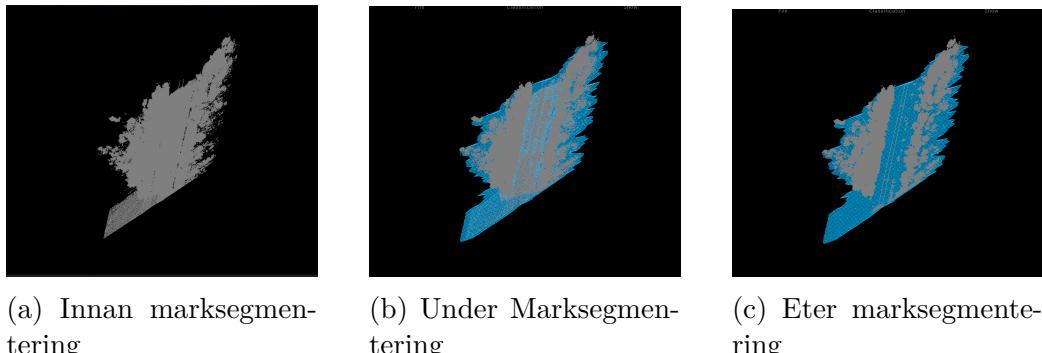
**Algorithm 2** Marksegmentering Tygsimulering

---

- 1: Skapa ett 2D-grid med godtycklig upplösning
- 2: Identifiera ett konvext hölje, ex Graham Scan
- 3: Ta bort alla gridpunkter utanför det konvexa höljet
- 4: **for**  $g \in \text{Grid}$  **do**
- 5:     Hitta en nära punkt  $p \in \mathbb{Punktmoln}$  och ta dess höjdsvärde
- 6:     Lokalisera alla grannar i  $\mathbb{G}$  inom 1 resp. 2 steg
- 7: **end for**
- 8: kör simulering
- 9: **for**  $g \in \mathbb{G}$  **do**
- 10:     Flytta gridpunkt  $g$  enligt acc.modell.
- 11: **end for**
- 12: **for**  $g \in \mathbb{G}$  **do**
- 13:     Kolla alla grannar och korrigera ditt eget höjdsvärde.
- 14:     Kolla att du inte överstiger ditt eget max/minvärde.
- 15: **end for**
- 16: iterera tills totala höjdskillnaden för alla punkter är under visst tröskelvärde

---

Steg 12 kan ses som att omgivande punkter *drar* i punkten. Hur stor dragkraften är påverkar styvheten i tyget.



Figur 3: Marksegmentering

## 2.4 Identifiera stampunkter

Här har jag valt att implementera ett beslutsträd som bygger på de statistiska egenskaper vi får genom att analysera egenvärdarna till kovariansmatrisen [3], [2]. Varje tröskelvärde i beslutsträdet motsvarar ett statiskt spridningsmått runt varje punkt i förhållande till omgivande punkter inom en viss radie. I detta fall så räknas spridningsmåtten fram genom att ta singulärvärdesdekompositionen av kovariansmatrisen inom ett område  $r$  runt punkt  $p$ . Singulärvärdesdekomposition kan man läsa mer om för att få intuitiv matematisk förståelse men det är utanför scope att återge det här. I detta fall har jag använt Eigen-bibliotekets implementering av SVD i C++. Detta ger oss egenvärdena som vi kan använda för att smälta upp några statistika spridningsmått så som planhet, linjäritet, varians, förändring av normalriktning, entropi och spridning vilka ses i tabell 1 nedan. Andra intressanta mått som ej använts kan vara densitet, färg, intensitet, laplacian osv.

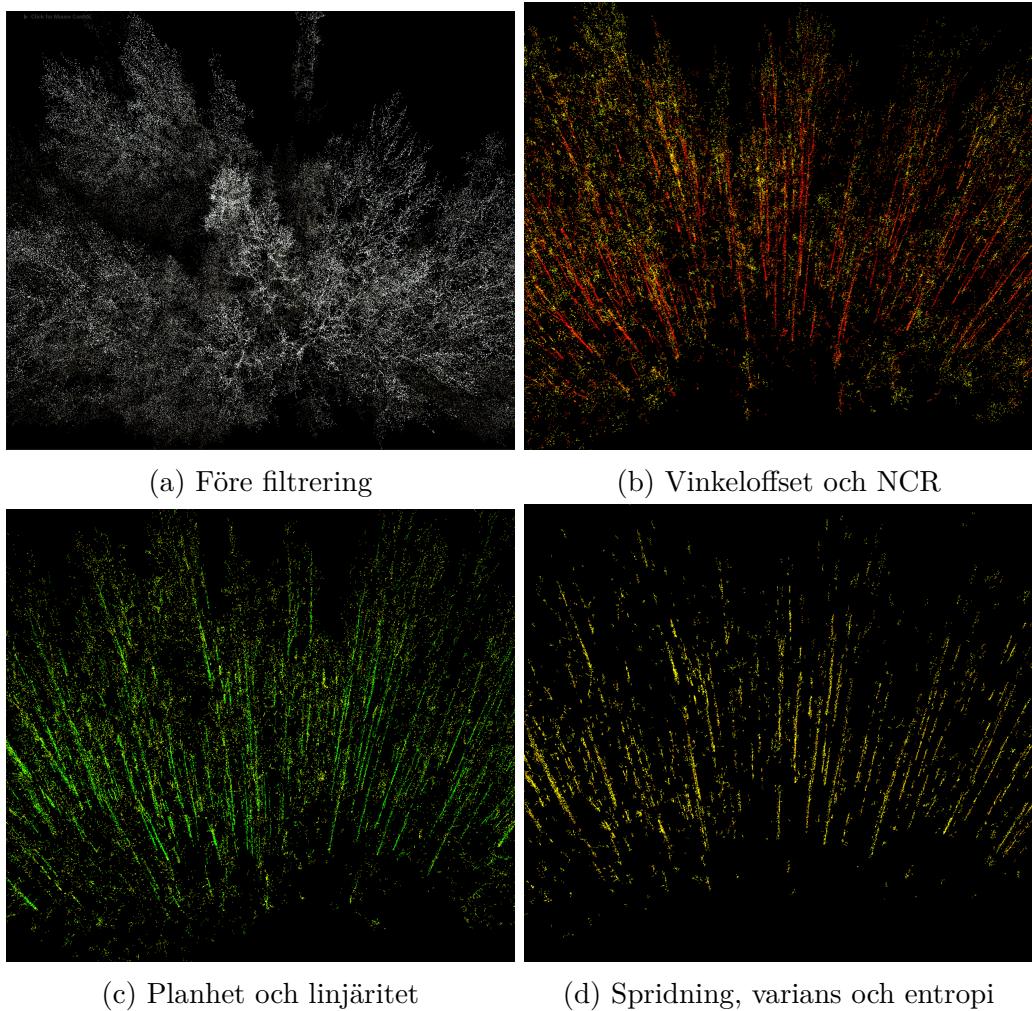
Ett annat värde om i detta fall fungerade bra var vinkelavstånd mellan första principalkomponent och z-axel. Under antagandet att en trädstam står rakt så kommer dataspridningen på stammen att vara parallel med z-axel. Huruvida det filtrerar bort lutande träd är något som bör testas, men för det lilla dataset jag testade på fungerade det väldigt bra (alla träd står relativt rakt).

Tabell 1: Statiska mått

Egenvärden	$\lambda_2 < \lambda_1 < \lambda_0$	Boxbredd (m)	Tröskelvärde
Alignment between N and z	$\ N \times Z\ $	0.3	>0.99
NCR	$\frac{\lambda_0}{\lambda_2 + \lambda_1 + \lambda_0}$	0.5	>0.6
Planhet	$\frac{\lambda_2 - \lambda_1}{\lambda_0}$	0.5	>0.1
Linjäritet	$\frac{\lambda_1 - \lambda_0}{\lambda_2}$	0.5	<0.6
Spridning	$\frac{\lambda_0}{\lambda_2}$	0.2	>0.2
Variation	$\lambda_0$	0.3	>0.15
Entropi	$-\sum_{n=0}^2 \lambda_i \log(\lambda_i)$	0.3	<0.2

Normalen  $N$  kan hämtas från sista kolumn ur U-matrisen från SVD-uppdelen, förutsatt att matrisen har en rang  $\geq 2$ . Boxbredd och tröskelvärdet är höftade och säkerligen inte optimala, men ger ett tillräckligt gott resultat för att bekräfta funktionaliteten. Beslutsträdet agerar som ett filter där vi fokuserar

på att behålla de punkter som förmögligen tillhör stammar. En nackdel här är att resultatet kan bli över/undersegmenterat beroende på hur hårt/slappt filtret är. En RF-algoritm där man hittar statiska spridningsmått i lådor av varierande storlek hade varit att föredra men det kräver träningsdata.

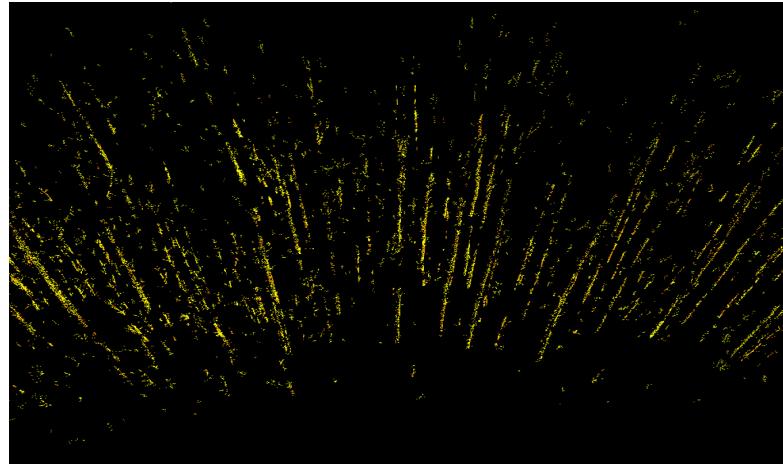


Figur 4: Filteringsresultat

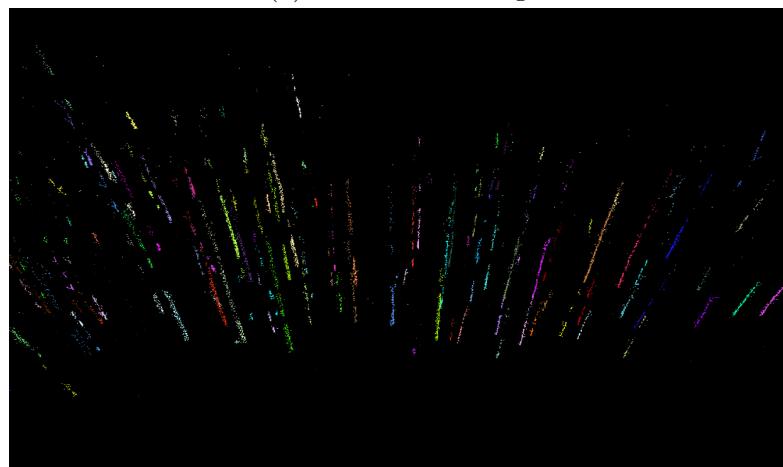
## 2.5 Voxelisering

När stampunkter har identifierats skapas ett 3D-histogram som omsluter alla kvarvarande punkter. Lådor som innehåller få punkter kan räknas bort som brus, med risk för undersegmentering. Punkter i samma låda klumpas ihop

med en närmstgranne-algoritm. För att klumpa ihop stammar kan man ta en låda och kolla i lådorna ovanför och under om de innehåller stamsegment inom ett visst xy-avstånd och med liknande riktning. Tanken är att en NN-algoritm kan klustra ihop punkterna lokalt och att voxeliseringsssteget kan klustra ihop segment som är långt ifrån varandra. Skuggning från träd närmare Lidarn gör att vissa delar på stammen inte upptäcks och voxelisering då ska lösa de problem som uppstår runt detta. Här finns risk om träd lutar mot varandra att vissa segment ”växer ihop” och det finns säkert stor utvecklingspotential för att skriva ad hoc-lösningar på detta problemet, men i det generella fallet bör stammarna vara så pass långt ifrån varandra att en NN-algoritm + voxelisering är tillräckligt bra för att klustra ihop majoriteten av§ punkterna.



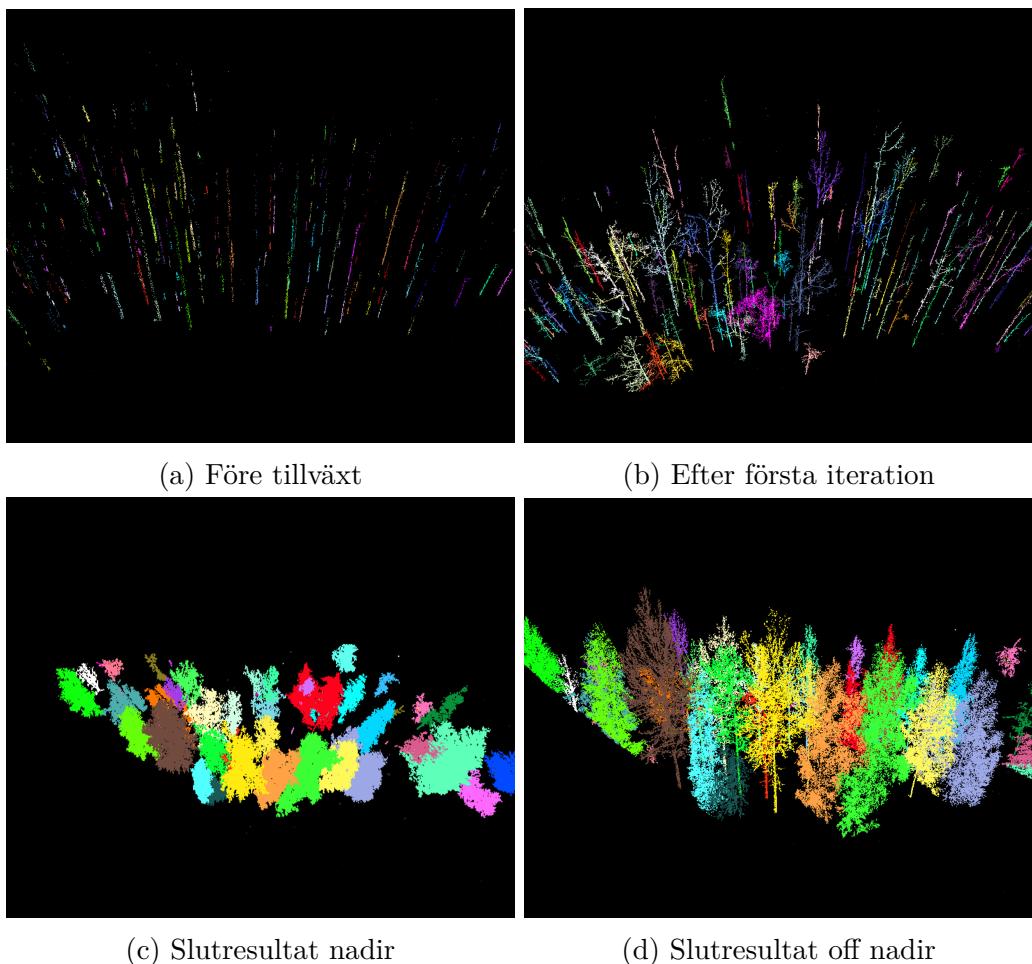
(a) Före voxelisering



(b) Efter voxelisering

## 2.6 Växtalgoritm

Växtalgoritmen utgår från samma typer av beräkningar som stamidentiferingen. Tanken är att vi identifierar närliggande punkter till varje stampunkt som vi också tror är stampunkter. Sedan letar vi kring dessa punkter o.s.v. När vi inte hittar fler så släpper vi lite på kraven och letar efter närliggande punkter inom de nya tröskelvärdena. På så vis växer trädet från stamtjockare grenar →tunnare grenar →löv allteftersom kraven på vad som tillåts tillhöra trädet släpps. Om en punkt vill tillhöra två träd så tillhör den det träd som är närmast granne.



Figur 6: Slutresultat

Jag hittade en mer strukturerad växtalgoritm som man kan prova att implementera, den bygger på samma principer men med lite mer motiveringar till

hur man ska tänka runt tillväxten [4]. Jag har en stark övertygelse om att man kan kombinera resultaten från en RF-algoritm med en tillväxtalgoritmen för att öka prestandan avsevärt.

## Referenser

- [1] Dong-Hyeon Kim et. al. *Automated Segmentation of Individual Tree Structures Using Deep Learning over LiDAR Point Cloud Data*. URL: <https://www.mdpi.com/1999-4907/14/6/1159>. (accessed: 05.06.2024).
- [2] H. Gross et. al. *SEGMENTATION OF TREE REGIONS USING DATA OF A FULL-WAVEFORM LASER*. URL: [https://www.isprs.org/proceedings/xxxvi/3-W49/PartA/papers/57\\_pia07.pdf](https://www.isprs.org/proceedings/xxxvi/3-W49/PartA/papers/57_pia07.pdf). (accessed: 05.06.2024).
- [3] Magnus Bremer et. al. *Eigenvalue and graph-based object extraction from mobile laser scanning point clouds*. URL: [https://www.researchgate.net/publication/264085076\\_Eigenvalue\\_and\\_graph-based\\_object\\_extraction\\_from\\_mobile\\_laser\\_scanning\\_point\\_clouds](https://www.researchgate.net/publication/264085076_Eigenvalue_and_graph-based_object_extraction_from_mobile_laser_scanning_point_clouds). (accessed: 05.06.2024).
- [4] Qianwei Liu et. al. *Point-cloud segmentation of individual trees in complex natural forest scenes based on a trunk-growth method*. URL: <https://link.springer.com/article/10.1007/s11676-021-01303-1>. (accessed: 05.06.2024).
- [5] Wuming Zhang et. al. *An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation*. URL: <https://www.mdpi.com/2072-4292/8/6/501>. (accessed: 05.06.2024).
- [6] ASPRS. *LAS file formats*. URL: [https://www.asprs.org/wp-content/uploads/2019/07/LAS\\_1\\_4\\_r15.pdf](https://www.asprs.org/wp-content/uploads/2019/07/LAS_1_4_r15.pdf). (accessed: 05.06.2024).
- [7] Mathew Fisher. *Cloth*. URL: <https://graphics.stanford.edu/~mdfisher/cloth.html>. (accessed: 05.06.2024).
- [8] wiki. *Ball tree*. URL: [https://en.wikipedia.org/wiki/Ball\\_tree](https://en.wikipedia.org/wiki/Ball_tree). (accessed: 05.06.2024).