CHALMERS

GÖTEBORGS UNIVERSITET

Utveckling av algoritmer för generering av takmodeller för digitala tvillingstäder

Development of algorithms for generating roof models for digital twin cities

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Ivan Flensburg
Linda Hoang
Holger Johansson
Alex Matsson
Mats Richardson
Alexander Samuelsson

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

Utveckling av algoritmer för generering av takmodeller för digitala tvillingstäder

Kandidatarbete i matematik inom civilingenjörsprogrammet Maskinteknik vid Chalmers

Alex Matsson

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk fysik vid Chalmers

Linda Hoang

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers

Mats Richardson Alexander Samuelsson

Kandidatarbete i matematik inom civilingenjörsprogrammet Samhällsbyggnadsteknik vid Chalmers

Ivan Flensburg Holger Johansson

Handledare:
Anders Logg
Vasilis Naserentin

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

Förord

Inledningsvis vill vi rikta ett stort tack till våra handledare Anders Logg och Vasilis Naserentin samt hela *Digital Twin Cities Centre* (DTCC). Vi vill även passa på att tacka: Ruben Seyer, Michal Palak, Alexander Thorén och Jesper Söderberg som bidragit med värdefulla synpunkter vid läsutbyten.

Vi som varit en del av detta projekt kommer från fyra olika program och har således olika akademiska bakgrunder. På grund av detta har vi kunnat bidra med olika saker och hela tiden haft flera olika synsätt på saker vilket har gynnat oss. Samarbetet har fungerat bra trots att det mesta av kontakten skett digitalt.

En loggbok, bestående av en gemensam dagbok samt individuella tidsloggar har förts genom hela arbetet. De tre inledande veckorna bestod av en ingående litteraturfördjupning för att bekanta sig i ämnet med litteratur som DTCC tillhandahållit. Efter dessa veckor delades gruppen i två olika grupper. En delgrupp för maskininlärning-, bildbehandling- och segmenteringsprocessen (ML-gruppen) samt en delgrupp med ansvar för genereringen av 3D-geometrin (GEO-gruppen). ML-gruppen har arbetat med bakomliggande material till avsnitt 2.3-2.6 i metoden. Linda Hoang har i huvudsak lagt mycket tid på undersökning av maskininlärningsmodellen: hur den fungerar, vilka parametrar som kan ändras och andra förfiningar för att få fram bästa möjliga resultat. Ivan Flensburg har haft huvudsakligt ansvar för bildbehandlingen av bilderna som maskininlärningsmodellen producerat. Det arbetet innebar mycket efterforskningar om digital bildbehandling och test av befintliga algoritmer för att hitta den optimala behandlingen. Holger Johansson har i huvudsak arbetet med skapandet av polygoner i 2D. Det arbetet innebar mycket kodande med verktyg för GIS och beräkningsgeometri.

GEO-gruppen, som bestått av: Mats Richardson, Alexander Samuelsson och Alex Matsson, har arbetat med materialet bakom avsnitt 2.7-2.12 i metoden. I denna delgrupp har alla tre arbetat tätt tillsammans med de flesta delar. Även om vi jobbat med samma saker har vi ofta sökt lösningar på egen hand först för att sedan välja vad som ska implementeras. På så sätt har arbetet kunnat vara effektivt samtidigt som vi fått fram flera alternativa lösningar och då minimerat riskerna att missa bra alternativ.

För rapporten finns huvudförfattare till varje avsnitt presenterat i tabell 1. Det ska även nämnas att alla i gruppen bidragit med synpunkter på varje avsnitt och på så sätt varit till stort hjälp för huvudförfattaren till ett avsnitt.

Tabell 1: Huvudförfattare till rapportens avsnitt

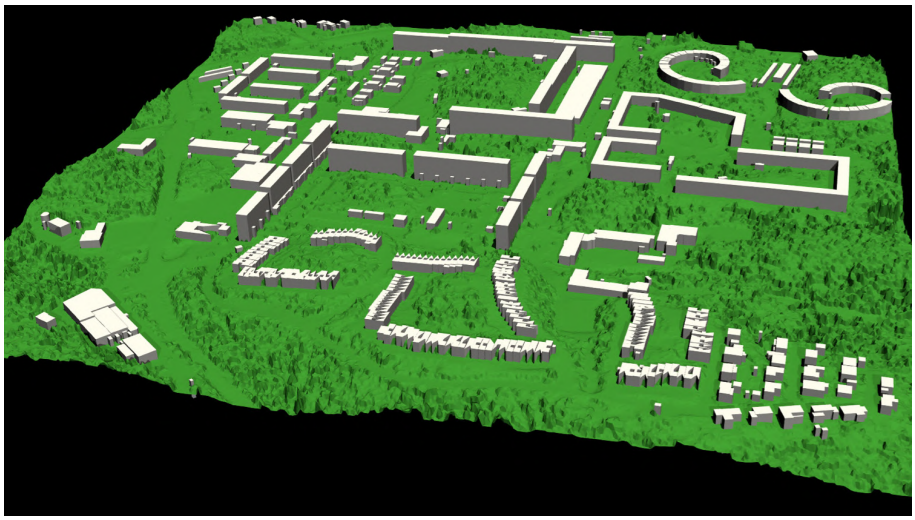
Kap.	Avsnitt	Huvudförfattare
	Förord	Alex
	Populärvetenskaplig presentation	Linda, Holger
	Sammanfattning och abstract	Alexander
1	Inledning	Alex, Holger
1.1	Syfte	Alexander
2	Metod	
2.1	Översiktlig beskrivning	Alex, Linda
2.2	Data	Holger
2.3	Skapande av maskininlärningsmodell för segmentering av bild	Linda
2.4	Träning av maskininlärningsmodell för segmentering av bild	Linda
2.5	Bildbehandling	Ivan
2.6	Takplan i 2D	Holger
2.7	Utvärdering av punktmoln	Alex
2.8	Para ihop datapunkt med polygon	Alex
2.9	Plangenerering med singulärvärdesuppdelning, SVD	Alexander
2.10	Filtrering av avvikande datapunkter	Mats
2.11	Sammanslagning av gemensamma hörnpunkter	Mats
2.12	Optimering för planarisering	Alexander
3	Resultat	
3.1	Generering av 2D-geometri	Ivan,Holger,Linda
3.2	Generering av 3D-geometri	Alexander
4	Diskussion	
4.1	Utvärdering av bildsegmentering till polygoner i 2D	Ivan,Holger,Linda
4.2	Resultat av generering av geometri i 3D	Alexander, Alex
4.3	Framtida arbete	Ivan
4.4	Samhälleliga och etiska aspekter	Mats
4.5	Slutsats	Mats
	Appendix	
A	Maskininlärningsmodell	Linda
B	Bildbehandling, utförlig beskrivning	Ivan
C	Takplan i 2D, utförlig beskrivning	Holger
D	Generering av polygon i 2D, utförlig beskrivning	Holger
E	Utvärdering av punktmoln	Alexander
F	Quadrant Angle Algorithm	Alex
G	Optimeringsproblem för planarisering	Alexander
H	Ett urval av genererade tak	Mats
I	Kod	

Populärvetenskaplig presentation

Har du någonsin funderat på hur framtidens städer kommer att se ut? Kommer de se likadana ut som de gör idag eller kommer de vara helt annorlunda? Även fast vi inte kan spå framtiden så är det en sak som är säker; världens städer växer så det knakar och andelen av världens befolkning som bor i städer ökar ständigt. Det leder till att det blir ännu viktigare att städerna uppfyller behoven av bra infrastruktur, miljö, social trivsel och låg klimatpåverkan.

Nya digitala verktyg kan underlätta i planeringen och underhållet av städer. Bland annat så har utvecklingen av digitala tvillingstäder accelererat de senaste åren. En digital tvillingstad är en datorsimulering av den verkliga staden som uppdateras i realtid. Den digitala tvillingstaden ska innehålla så mycket information som möjligt om allting som sker i staden. Det är allt från den fysiska miljön som byggnader och vägar till den ekologiska och sociala miljön med luft och informationsflöden. Digitala tvillingstäder ska till exempel kunna användas för att snabbt överblicka och koordinera hjälpinsatser vid olyckor men också simulera påverkan av planerade byggnadsprojekt innan de faktiskt byggs.

Innan det går att testa och simulera händelser i en stad så måste först själva stadsmiljön skapas digitalt. Ett viktigt steg i den processen är att skapa en realistisk 3D-modell av byggnader. För att kunna skapa digitala tvillingstäder i en stor skala så måste detta ske automatiskt och med lätthet kunna uppdateras i takt med att staden förändras. Detta arbete genomförs i samarbete med *Digital Twin Cities Centre* (DTCC) vilka har målet att skapa digitala tvillingstäder innan år 2030 [1]. DTCC har hittills lyckats skapa den digitala 3D-modell som visas i figur 1 men vill öka detaljrikedomen.



Figur 1: Den modell av stadsmiljön DTCC har i dagsläget [1]

Målet med detta projekt är att utifrån flygfoton, fastighetskartor och laserskanningar från flygplan skapa verklighetstrogna 3D-modeller av byggnadernas tak och sedan utvärdera denna process. För att uppnå detta krävs flertalet delsteg vilka redovisas i rapporten. Sammanfattningsvis är första delen att känna igen taklinjer på flygfoton med hjälp av maskininlärning, bildbehandling och hörnigenkänning. Resultatet dessa steg blir en svartvit bild som visar takets linjer från ett fågelperspektiv. Genom beräkningsmatematik och optimering kan sedan ritningen användas för att generera 3D-modellen.

Det slutliga utfallet av arbetet är 3D-takmodeller som har genererats genom att följa processen som beskrivs i rapporten. Trots att det endast var några få takmodeller som kunde tas fram genom att följa hela processen så har projektet kunnat dra viktiga slutsatser för framtida utveckling av detta spännande område. Så vad tror du? Hur kommer framtidens städer att se ut? Vi tror i alla fall att digitala tvillingstäder kommer underlätta livet i dem.

Sammanfattning

En digital tvilling är en modell av en stad som i varje ögonblick ska spegla staden och dess rörelser. Digitala tvillingar förutspås ha en stor mängd användningsområden i framtiden, exempelvis inom stadsplanering eller koordinering av hjälpinsatser vid olyckor. Tidigare har det krävts mycket manuellt arbete för att ta fram digitala modeller vilket gjort processen tidskrävande. Ny teknologi har möjliggjort automatisering av processen, något som idag är föremål för forskning, bland annat på *Digital Twin Cities Centre* (DTCC) på Chalmers.

I detta arbete undersöks algoritmer för att automatiskt generera takmodeller för digitala tvillingstäder. Med hjälp av befintlig data i form av ortofoton, fastighetskartor, laserdata samt en implementerad maskininlärningsmodell från DTCC har gruppen tagit fram en metod för att generera takmodeller. Först segmenteras ortofoton för att hitta takens kanter och taknockar. Sedan behandlas bilderna för att jämna ut segmenteringen och hitta hörnpunkter. Från detta kombineras fastighetskartorna, de segmenterade bilderna och hörnpunkterna för att bestämma takplanen i 2D. Nu kan en 3D-geometri genereras genom att laserdatan kopplas till rätt takplan med *quadrant angle-algoritmen*. Därefter genereras plan med *singulärvärdesuppdelning* och laserdatan filtreras för avvikande punkter. Avslutningsvis sammanfogas gemensamma hörnpunkter och ett linjärt optimeringsproblem appliceras för att planarisera takplanen.

Resultaten är lovande men visar också på utmaningen i att automatisera denna komplexa process. Det är uppenbart att mycket arbete kvarstår innan gruppens metod kan appliceras på stor skala. Däremot tas ett antal korrekta takmodeller fram vilket visar på potential. Framtida förbättringsområden inkluderar att hantera förskjutningar som uppkommer hos segmenteringarna och förbättra planariseringen av takplanen. Fortsatta frågeställningar kan vara att undersöka hur byggnadspolygoner och punktmoln kan utnyttjas under bildsegmenteringsfasen samt att utreda huruvida en kombination av en data- och modellbaserad metod ger bättre resultat.

Abstract

A digital twin is a model of a city that, in each moment, should accurately reflect the city and its movements. Digital twins are predicted to have a large number of real-world applications, for example in urban development or coordination of relief efforts during times of crisis. In the past, extensive manual labour has been required to create digital models which has made the process time consuming. New technology has enabled automation of the process, which currently is a subject of research on *Digital Twin Cities Centre* (DTCC) on Chalmers.

This thesis investigates algorithms to automatically generate roof models for digital twin cities. With the help of existing data in the form of orthophotos, building footprints, LiDAR-data and an implemented machine-learning model from DTCC the group has created a method for generating roof models. First orthophotos are segmented to find the edges and ridges of roofs. Then the images are processed to smooth out the segmentation and find the cornerpoints. From this the footprints, segmented images, and cornerpoints are combined to determine the roofplanes in 2D. Now 3D-geometry can be created by combining the LiDAR-data to the right roofplane with the *quadrant angle algorithm*. Thereafter planes are generated with *singular value decomposition* and the LiDAR-data is filtered for outliers. Finally common cornerpoints are merged and a linear optimization problem is applied to planarize the roofs.

The results are promising but also show the challenges involved in automating this complex process. It is clear that much work still remains before the method created by the group can be applied in a large scale fashion. However some correct roof models are generated, which shows the potential of the method. Future improvements include handling displacements that arise during the segmentation process and improving the planarization of the roofplanes. Future work could involve research in how buildingpolygons and point clouds could be used during image segmentation as well as investigating wheter a combined model- and datadriven approach yields better results.

Ordlista

Artificiellt neuronnät - Datorprogram som kan lära sig att dra slutsatser på data genom att först träna på liknande data. Namnet kommer från att programmets uppbyggnad är inspirerat av hur hjärnan fungerar [2].

Byggnadspolygon - Även kallat byggnadsarea. Avser den markyta som upptas av en byggnad.

CNN - *Convolutional Neural Network*, eller faltningsnätverk, är en klass neurala nätverk som ofta används inom bildanalys [3].

DTCC - *Digital Twin Cities Centre*, ett forskningsprojekt på Chalmers med syfte att skapa virtuella tvillingar av städer [1].

Geodetiskt referenssytem - Ett system för att bestämma punkter på jordytans position och hur de ändras med tiden [4].

Laserskanning - Från engelska: *Light detection and ranging* (LiDAR), ett mätinstrument för att finna avstånd. Används bl.a för att skapa punktmolnen.

LOD - *Level of detail*, en term som beskriver detaljnivån på den virtuella tvillingen [5].

Ortofoto - En flygbild som är geometriskt korrigerad för höjdskillnader för att bli skalriktiga.

Punktmoln - En datastruktur bestående av ett moln av koordinater i tre dimensioner.

Spatial - Rumslig, har att göra med utsträckning i rummet.

SVD - *Singulärvärdesuppdelning*, en matrisfaktorisering som ofta används inom dataanalys.

UUID - *Universally unique identifier*, en bit-kod som används för att ge objekt en unik etikett.

Innehåll

1 Inledning

Moderna städer består av långa komplexa flödeskedjor och aktörer sammankopplade i komplicerade nätverk. En stad är alltid i rörelse, både inom dess egna nätverk av infrastruktur men också med avseende på invånarantal och yta. Detta tillsammans med att klimatförändringar ändrar de fysiska förutsättningarna gör att dagens städer är mycket komplexa att planera, utveckla och förvalta [6].

Samhällen runt om i världen har länge använt abstraktioner av städer som beslutsunderlag och för att få en överblick. I takt med teknikens utveckling har dessa övergått från enkla kartor på papyrus, till mer avancerade och informationsrika kartor på papper och sedan till dagens digitala och interaktiva kartor och modeller med flera dimensioner av information. En flerdimensionell modell kan innehålla både 3D-geometrier men också annan information såsom trafikflöden [7]. Idag krävs det i regel mycket manuellt arbete för att skapa användbara 3D-geometrier av städer, eftersom alla byggnader måste ritas upp för hand i en dator. I takt med att datorers prestanda ökas och metoder förfinas har efterfrågan av att automatisera denna process för att skapa digitala tvillingar av städer ökat.

En stads digitala tvilling ska i varje ögonblick spegla hela staden och dess rörelser i en digital sfär. Den ska alltså spegla stadens fysiska utseende i 3D och därtill innehålla flerdimensionell information om flöden och händelser. Denna information bör uppdateras i realtid för att alltid kunna återge staden i dess aktuella tillstånd. Digitala tvillingar av städer förutspås ha en stor mängd användningsområden som skulle vara till nytta för hela samhällen och dess invånare. Digitala tvillingar skulle kunna användas som bland annat virtuell testbädd för innovation, stadsplanering och som verktyg för att förutse och hantera kriser [8]. Konkreta exempel på användningsområden är till exempel att vid stora olyckor snabbt kunna utvärdera omfattningen och organisera hjälpinsatser. Det kan också vara att simulera och visualisera hur nya stadsplaneringsprojekt påverkar staden med allt från vind och trafikflöden till sociala aspekter såsom flyttmönster och segregation [8].

Ett viktigt steg i processen att automatiskt skapa digitala tvillingar av städer är att skapa en realistisk 3D-geometri. Kvaliteten på de genererade husmodellerna av staden delas upp i olika LOD (Level of Detail), där de olika nivåerna av detaljrikedom definieras enligt figur 2 [5]. En husmodell i LOD0 består endast av dess byggnadspolygon. Husmodeller i LOD1 byggs på med en utsträckning i höjd från marken. I LOD2 bör alltid den huvudsakliga takformen återges korrekt. Med huvudsaklig takform menas i detta fall om huset har till exempel valmat tak: tak som sluttar åt alla sidor eller sadeltak: tak som sluttar åt två sidor och möts i en taknock (taken som syns i figur 2 för LOD2, LOD3 och LOD4). För LOD3 ökar detaljrikedomen då takkupor, takutskjut och fönster kan inkluderas. Den sista nivån, LOD4, kan även inkludera byggnadens interiör [5].

Figur 2: Beskrivning av LOD där LOD0 är den enklaste och LOD4 den mest detaljrika modellen [5]. CC BY-NC-ND 4.0.

Det här projektet utförs inom ramen för det pågående forskningsprojektet Digital Twin Cities Center (DTCC), från vilka en maskininlärningsmodell samt indata tillhandahållits. Deras vision är att skapa praktiskt användbara digitala tvillingar av städer innan år 2030 [1]. För närvarande har forskningsprojektet en modell av LOD1 där alla huskroppar har platta tak. Målet för DTCC är nu att höja detaljrikedomen i modellen och skapa huskroppar med LOD2. Ett flertal olika metoder har testats och det forskas kontinuerligt inom området [8][9].

Vid generering av takgeometri finns det två välkända tekniker, modellbaserad och databaserad [10]. Den modellbaserade tekniken utgår ifrån en fördefinierad katalog med takmodeller. Katalogen innehåller oftast de primitiva takformerna: plana tak, sadeltak och valmat tak. Denna teknik fungerar bra för enklare takformer och garanterar att de genererade taken har en regelbunden

och komplett struktur. För mer komplexa former approximeras taket med en enklare variant från modellkatalogen. Med den databaserade tekniken används inte någon fördefinierad katalog för takmodeller utan denna teknik utgår direkt från datan och kräver en segmentering (uppdelning) av takets geometri. Alla tak hanteras på samma sätt och det krävs inte lika många förenklingar av komplexa geometrier. Tekniken i sig är dock ofta mer komplex att implementera och mer beroende av kvaliteten på punktmolnet [11]. I detta projekt används en databaserad teknik.

1.1 Syfte

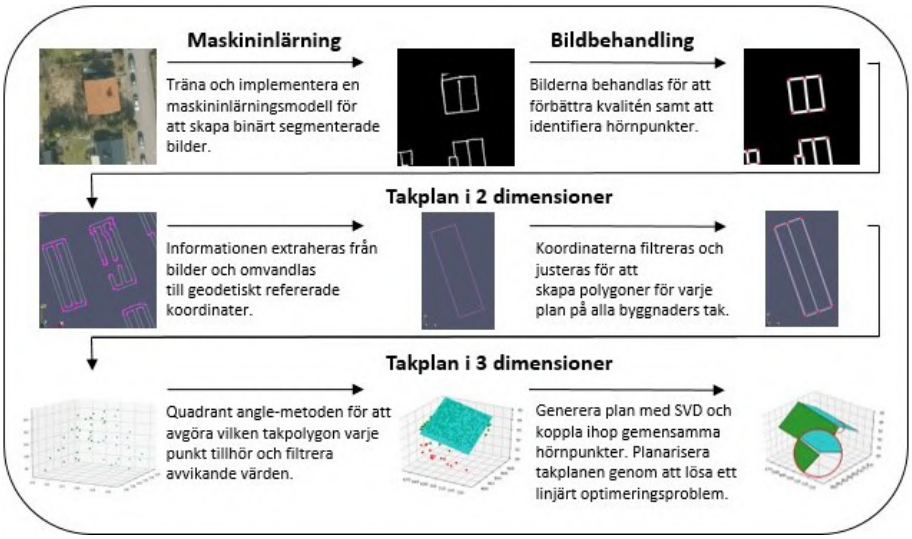
Syftet med projektet är att undersöka, utveckla och utvärdera metoder för att automatiskt generera digitala tvillingar av städer med detaljnivå LOD2. Projektet fokuserar på modelleringen av tak till byggnader i städer och kan delas upp i två huvudområden. Först segmenteras taken med hjälp av maskininlärning, bildbehandling och hörnigenkänning. Segmenteringen används sedan för att generera geometri för taken. Då används algoritmer och metoder från beräkningsgeometri. Målet är att utifrån ortofoton, fastighetskartor och laserdata skapa korrekta virtuella modeller av tak tillhörande många olika typer av byggnader.

2 Metod

Detta avsnitt beskriver metoden som använts för att generera virtuella modeller av hustak. I avsnitt 2.1 ges en översiktlig beskrivning av hela metodprocessen. Indatan består av punktmoln, ortofoton och fastighetskartor enligt beskrivningen i avsnitt 2.2. Delavsnitt 2.3-2.6 handlar om segmenteringen av taken och delavsnitt 2.7-2.12 om generering av geometri utifrån segmenteringen.

2.1 Översiktlig beskrivning

Hela metodprocessen kan sammanfattas enligt flödesschemat i figur 3. Inledningsvis tränas en maskininlärningsmodell för att få ut binärt segmenterade bilder. Därefter sammanfogas de små segmenterade bilderna och bildbehandling påbörjas för att sedan kunna identifiera hörnen i bilderna. Vidare omvandlas bilder som beskrivs som en matris med pixlar till en lista med slutna polygoner. Polygoner och dess hörn kopplas sedan ihop med respektive byggnad. Vid detta stadie var generering av 2D-geometri klar. Generering av 3D-geometri inleds med en utvärdering av punktmolnet för en mängd byggnader. Därefter används *quadrant angle-algoritmen* för att ge varje takpolygon ett eget punktmoln och *singulärvärdesuppdelning* för att generera initiala takplan. Utifrån dessa takplan genomförs en filtrering av avvikande punkter för att sedan på nytt generera bättre anpassade plan. Slutligen åtgärdas glapp mellan takplan och det genomförs ett försök att planarisera takplanen genom att lösa ett linjärt optimeringsproblem.



Figur 3: Övergripande sammanfattning av hela metodprocessen.

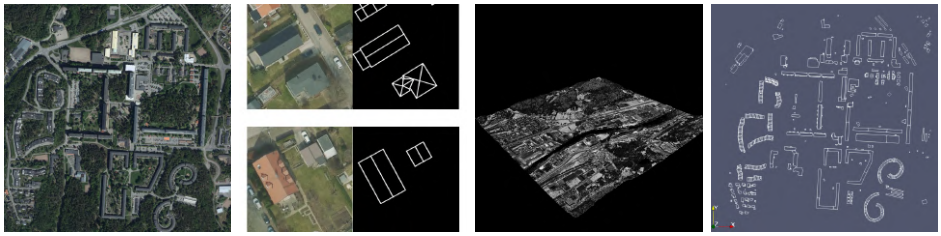
2.2 Data

All data som projektet utgår ifrån är tillhandahållen av DTCC [1]. Projektet använder data över stadsdelarna Eneborg, Fältbacken och Husensjö i Helsingborg samt Hammarkullen i Göteborg. Datan består av tre olika typer vilka är ortofotografier, fastighetskartor och laserdata i form av punktmoln.

Ortofotografi är flygfotografier som korrigerats för att vara ortogonala och skalriktiga, se figur 4a. Det betyder att avståndet mellan punkter i fotografiet inte påverkas av markens höjdskillnader. Med andra ord så betraktas alla punkter i fotografiet rakt uppifrån. Ortofotografierna är även geodetiskt refererade vilket betyder att varje pixel i fotografiet har en geografisk koordinat tilldelad. Det möjliggör omvandling från fotografi till koordinatsystem i 2D [12]. Den fotografiska datan som används lagras på matrisform där varje pixel motsvarar ett element i matrisen. Varje pixel har ett tilldelat värde för att beskriva vilken färg den ska representera [13]. Ortofotografierna som använts har delats upp i ett rutnät bestående av flera mindre kvadratiska bilder för enklare hantering. Till varje bild i rutnätet finns även en svartvit bild där varje byggnads takplan har markerats med vita konturer, se figur 4b. Dessa används som träningsdata vid bildsegmenteringen som beskrivs i avsnitt 2.3.

Även laserdata samlas in genom flygningar över ett område, se figur 4c. Laserdata består av ett punktmoln där varje punkt har koordinater i 3D. Punktmolnet som används i projektet har en ungefärlig densitet på 1,5 punkter per kvadratmeter markyta. Även punktmolnet är geodetiskt refererat i ett bestämt referenssystem [14].

Punktmolnet som används är filtrerat och en del avvikande punkter är borttagna genom DTCC:s algoritmer. Även fastighetskartorna är processade och filtrerade. Fastighetskartorna som används består därför bara av områdets byggnadspolygoner. Med byggnadspolygoner menas den yta på marken som huset står på, se figur 4d. Alla byggnadspolygoner är beskrivna med geodetiskt refererade koordinater. Laserdatan och byggnadspolygonerna är lagrade i tabellform där en kolumn består av varje byggnadspolygon som har ett unikt en *Universally unique identifier* (UUID) och en kolumn innehåller punkterna i punktmolnet som ligger i eller intill den byggnadspolygonen.



(a) Ortofotografi (b) Del av ortofotografi med tillhörande segmenteringar (c) Punktmoln (d) Byggnadspolygoner

Figur 4: Datatyper.

2.3 Skapande av maskininlärningsmodell för bildsegmentering

I följande avsnitt behandlas processen för att skapa en maskininlärningsmodell för bildsegmentering. Inledningsvis beskrivs maskininlärning och dess användning inom bildsegmentering. Sedan presenteras den valda maskininlärningsmodellen.

2.3.1 Grunden för maskininlärning och bildsegmentering

Maskininlärning kan brett förklaras som databaserade metoder som automatiskt förbättras genom erfarenhet, för att öka prestanda eller ge korrekta förutsägelser [15]. Mängden data och dess kvalitet är avgörande för hur lyckat resultatet blir. Tillämpningar för maskininlärning är exempelvis att klassificera dokument genom att avgöra vilket ämne en text handlar om, röstigenkänning eller för objektigenkänning inom datorseende. Oavsett applikation består den generella metoden av följande steg: 1. Insamling av data. 2. Rensning av data genom borttagning av saknade eller brusiga element. Sedan indelas datan i en träningsdatamängd och en testdatamängd. 3. Träning av modellen. 4. Utvärdering av modell med testdatamängden. 5. Förbättra modell genom att ändra algoritmer eller parametrar.

Grunden till många maskininlärningsmodeller är så kallade artificiella neuronnät (ANN). ANN är en typ av självlärande algoritmer för maskininlärning som är inspirerade av arkitekturen och dynamiken i de biologiska neuronnätverken i hjärnan [16]. I en biologisk hjärna överförs signaler eller information till och från neuroner genom kopplingar. Detta kan modelleras av ett artificiellt neuronnät, där nätverket lär sig genom att ändra kopplingarna som även kallas vikter mellan neuroner. Neuronerna är uppdelade i lager, där informationen färdas mellan lagrena. Signalerna rör sig från det första lagret (inmatningsskiktet) till det sista lagret (utgångsskiktet). Om informationen endast rör sig framåt och inte har några loopar kallas detta framåtkopplat nätverk.

För att nätverken ska lära sig, tränas de utefter önskat syfte. I detta arbete behandlas ett inlärningsätt som kallas övervakad inlärning (*supervised learning*), vilket innebär att nätverket matas med känd träningsdata och kända svar. Ett exempel är nätverk som tränar på kategoriserade bilder för att sedan själv kunna kategorisera nya bilder.

Bildsegmentering kan användas för att veta var ett objekt ligger i bilden. Vid segmentering tilldelas varje pixel i en bild en klass. Metoden används ofta för att klassificera bilder för applikationer som behöver hög noggrannhet då det är svårt att åstadkomma detta vid manuell klassificering [17]. Det finns flera olika typer av bildsegmentering. Semantisk segmentering innebär att varje pixel klassificeras som en del av ett objekt som när alla människor i en figur är segmenterade som ett objekt och bakgrund som ett objekt [18]. Om en pixel enbart kan tillhöra en av två klasser är bildsegmenteringen binär. Vid semantisk binär segmentering av tak innebär detta att takets definerade linjer såsom taknocken och ytterkanterna representeras med vit pixel och övriga pixlar tillhör klassen svart. Resultatet av denna pixelvis svartvita bild kallas för en mask av bilden.

En teknik för att applicera bildsegmenteringen är genom att träna ett neuralt nätverk. Indata till nätverket består av en bild samt tillhörande sanna mask och efter träning utmatas en ny mask av bilden. För att en modell ska lära sig beräknas en förlustfunktion där den utmatade bilden jämförs med den sanna masken. Denna förlust vill nätverket minimera. Efter att förlusten har beräknats uppdateras vikterna mellan neuronerna, och på så sätt förbättras nätverket. För att få

en bättre uppfattning om hur bra nätverket presterar, mäts förlust under träning, dels mäts så kallad träningsförlust men också valideringsförlust. Validering görs på en separat datamängd som nätverket inte har tränats på och ingen uppdatering görs med hjälp av den beräknade valideringsförlusten. På detta vis kan man testa hur bra nätverket fungerar på okänd data.

2.3.2 Val av modell för bildsegmentering

Nätverket som har presterat bäst inom bildigenkänning är faltningsnätverk (CNN) [19]. CNN är en metod som har stor inlärningskapacitet. Istället för att varje neuron representerar en pixel, som i motsvarande framåtkopplat nätverk, så representerar varje neuron lokala huvuddrag som exempelvis kanter eller hörn från ett område med flera pixlar. Således reduceras antalet kopplingar och parametrar, vilket minskar komplexiteten vid träning.

En av de mest framgångsrika metoderna för bildsegmentering är U-Net [20] som har en CNN-arkitektur, vilket är modellen som används i detta arbetet. En bild på nätverkets arkitektur finns i appendix A. Följden av nätverkets olika lager har formen av ett ”U”. Indata ges till lagret längst upp till vänster på ”U:et” och resultatet hämtas i lagret längst upp till höger på ”U:et”. Vänstersidan fungerar som en kodare och högersidan som en avkodare. I kopplingen mellan kodaren och avkodaren så extraheras huvuddragen. Enkelt förklarat är kodarens funktion att reducera indatabilden och avkodaren ska bygga ihop bilden igen med hjälp av de extraherade huvuddragen. För att viktig information inte ska gå förlorad i kodaren så finns det även ”skip”-lager mellan varje kodare och avkodare som kopplas till utdatalagret.

2.4 Träning av maskinlärningsmodell för bildsegmentering

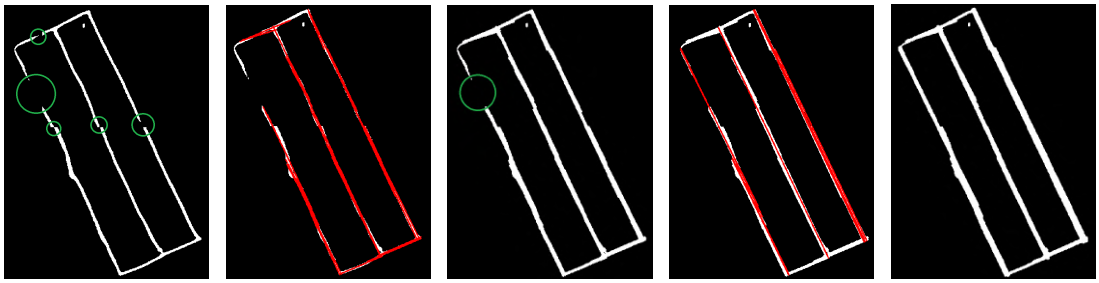
För att generera binärt segmenterade bilder över hustaken implementeras en maskininlärningsmodell där en datamängd av hustak inmatas. Sedan tränas en modell som förutspår segmenterade bilder från nya okända tak. Datamängden kommer från Helsingborgs stad och innehåller 524 bilder samt motsvarande mask på stadsområdena Eneborg, Fältbacken och Husensjö. För att få ut mer av datan görs en utökning av datamängden. För varje bild görs tre olika transformationer, en horisontell vändning, en vertikal vändning och en inzoomning på mitten av bilden. Generering av dessa tre nya bilder per orginalbild ökar datamängden till 2384 bilder, vilket är fyra gånger så stort som det ursprungliga. 80% av datamängden används till träning och 20% till validering. Sedan körs träningen genom U-Net-modellen. Träningen utförs i Google Colab, med deras NVIDIA Tesla K80 GPU [21]. Efter träning utvärderas modellen, parametrar ändras och ny träning startas. Sist sparas den bästa modellen som erhållits. Se appendix A.1 för detaljer om modellparametrar.

2.5 Bildbehandling

Den bäst tränade maskininlärningsmodellen används för att generera binärt segmenterade bilder över byggnaderna i stadsområdet Eneborg, och därefter assemblera dessa till en sammanslagen binär bild. Därefter genomgår bilden en serie bildbehandlingar med användning av flera algoritmer. Syftet med detta är att öka möjligheten att få ut korrekta hörnpunkter och en geometri som förenklar nästkommande processer.

Val av algoritmer för bildbehandling och deras ordningsföljd är ett resultat av en arbetsgång bestående av prövning och omprövning. Algoritmföljden som beskrivs nedan är den som ansågs ge bäst resultat. Bakomliggande programkod där algoritmerna implementerades tillsammans med parametrar finns i appendix B.1.

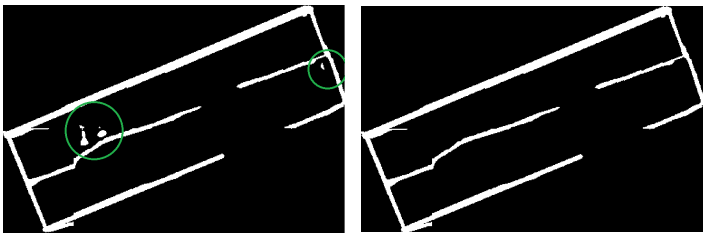
Inledningsvis appliceras *probabilistisk Hough-linje-transform* på den binära sammanslagna bilden i två iterationer med olika parametrar [22]. Algoritmen finner vita pixlar som kan representeras av en rät linje med start och slutpunkt. Beroende på ingående parametervärden kan detekterade linjer spänna mellan flera vita pixlar som är separerade av mindre glipor. Då den ursprungliga bilden innehåller områden med flera intilliggande vita pixlar med med glipor mellan, kan algoritmen användas för att fylla dessa glipor. På så sätt skapas en sammanhängande vit linje. Figur 5 illustrerar ett exempel på en byggnad där de inringade gliporna täcks av detekterade linjer. Efter två iterationer återstår inga glipor i geometrin för byggnaden.



(a) (b) (c) (d) (e)

Figur 5: Detekterade linjer i rött, glipor att fylla i markerade med grön cirkel, byggnadens geometri i vitt. (a) Obehandlad binär bild. (b) Första iteration *Hough-linje-transform*. (c) Resultat efter första iterationen. (d) Andra iteration *Hough-linje-transform*. (e) Slutgiltigt resultat.

I den sammanslagna binära bilden finns det oönskade komponenter som inte bidrar till generering av korrekt geometri för byggnaderna. I figur 6a presenteras ett exempel på detta. Kriteriet för en komponent är att en vit pixel centrerad i en 3×3-matris har en granne i alla riktningar, horisontellt, vertikalt och diagonalt. På de oönskade komponenterna kan hörn detekteras när hörnigenkänningsalgoritmen används längre fram. Därför tas dessa bort med användning av *Grana, Borghesani och Cucchiaras algoritm* [23]. Den listar alla komponenter i den sammanslagna bilden, tillsammans med antal pixlar i komponenten samt dess position. Därefter sätts ett gränsvärde för minsta tillåtna antal vita pixlar i en komponent, och om detta underskrids tas komponenten bort från listan. Slutligen genereras en ny helt svart bild med samma storlek och de kvarstående komponenterna i listan ritas in med vitt igen, se figur 6b.



(a) (b)

Figur 6: Mindre komponenter att ta bort i grön cirkel, byggnadens geometri i vitt. (a) Obehandlad binär bild. (b) Efter borttagning av mindre komponenter.

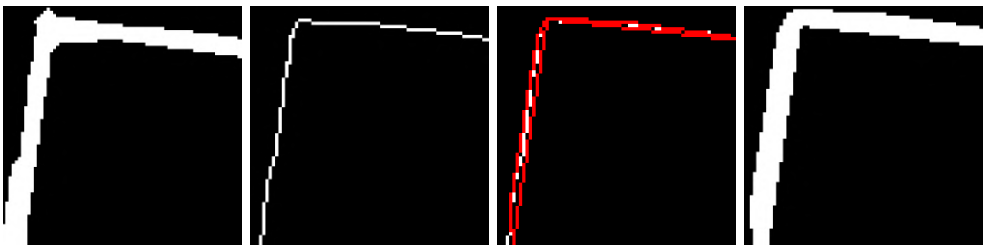
Nästa steg i bildbehandlingsprocessen är att utföra en förtunning av de vita komponenterna i bilden, vilket är ett nödvändigt steg inför nästkommande del. Förtunningen åstadkoms med *Zhang-Suens förtunningsalgoritm* [24]. Den producerar ett topologiskt skelett av komponenterna, vilket innebär att den förtunnade komponenten är ekvidistant till den ursprungliga komponentens kanter.

De förtunnade komponenterna är ojämna på grund av ojämnheter i ursprungsbilden. Det resulterar i att oönskade hörn detekteras när hörnigenkänningsalgoritmen körs, se bildexempel i figur 22 (appendix B.2). För att undvika denna problematik genomförs en utjämning av komponenternas kanter i en serie steg.

Inledningsvis finner man alla konturer, det vill säga ytterlinjer bestående av en samling linjesegment med olika riktning, kring de förtunnade komponenterna. Konturerna fås med hjälp av *Abe och Suzukis algoritm*, och resulterar i en lista med start- och slutpunkt för varje linjesegment i tillhörande ytterlinje [25]. Då komponenterna mestadels är en pixel tjocka innebär det också att ytterlinjerna på respektive sida av komponenten i många fall överlappar varandra. Det är en önskvärd effekt, och anledningen till att *Zhang-Suen förtunningsalgoritm* tillämpades tidigare. Om komponenten inte vore förtunnad hade man sett ett större avstånd mellan ytterlinjerna, vilket hade skapat två parallella linjer för varje komponent.

Ytterlinjernas form genererad med hjälp av *Abe och Suzukis algoritm* approximeras till ett färre antal linjesegment samtidigt som formen i viss utsträckning behålls, beroende på angivet parame-

tervärde. Detta minskar antalet kanter som ritar upp geometrin. Approximationen genomförs med hjälp av *Ramer-Douglas-Peuckers algoritm* [26]. Parametern som avgör hur avvikande ett linjesegment får vara från den nya formationen är baserad på varje ytterlinjes totala längd multiplicerat med en konstant, med antagandet att ytterlinjen kring komponenten är sluten. Slutligen utvidgas de approximerade ytterlinjerna så att avståndet mellan ytterlinjerna på respektive sida av komponenten mellan täcks upp, och ytterlinjerna sammanfogas till en vit linje. I figur 7 presenteras exempel på hur arbetsgången för en komponent som representerar ett hörn på en byggnad ser ut, där den slutligen får jämnare kanter. Notera också i figur 7c hur de approximerade linjerna på respektive sida av den förtunnade linjen har ett litet avstånd till varandra. Detta gör att de approximerade linjerna i 7d framstår som en enhetlig linje.



(a) (b) (c) (d)

Figur 7: Approximerad linje av *Ramer-Douglas-Peuckers algoritm* i rött, byggnadens geometri i vitt. (a) Obehandlad binär bild. (b) Förtunnad bild med *Zhang-Suens förtunningsalgoritm*. (c) Approximerade linjer. (d) Slutgiltigt resultat.

I den nya sammanslagna bilden har alla komponenter relativt jämna kanter. Återigen tillämpas en förtunningsalgoritm för att positionen på hörn detekterade i hörnigenkänningsdelen ej skall vara förskjuten på grund av komponentens tjocklek. I denna förtunning tillämpas *Guo-Halls förtunningsalgoritm* [27]. Anledningen till användandet av en annan förtunningsalgoritm är att den producerar ett bättre topologiskt skelett än *Zhang-Suens förtunningsalgoritm* vad avser antal grannar för varje vit pixel. Detta gör också att man får färre falska hörn detekterade på komponenter som är jämna, se figur 23 (appendix B.3). En jämförelse av antal vita pixlar efter förtunning, presenterat i tabell 5 (appendix B.3) bekräftar också iakttagelsen. Användning av *Guo-Halls förtunningsalgoritm* resulterar i cirka 10% färre vita pixlar och 38% färre hörn än *Zhang-Suens förtunningsalgoritm*.

Samtidigt är det inte heller lämpligt att använda *Guo-Halls förtunningsalgoritm* i tidigare förtunningssteg, då den påvisar oönskat utseende för komponenter med ojämna kanter. De ojämna kanterna resulterar i flera ”grenar” som bildas längs komponenten och senare detekteras som hörn. I figur 24 (appendix B.3) presenteras ett exempel på detta, där man också kan se att *Zhang-Suens förtunningsalgoritm* bildar färre grenar.

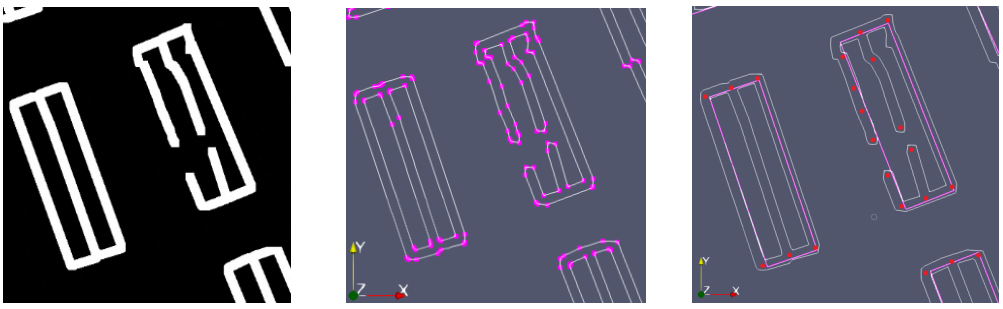
Avslutningsvis detekteras hörn på den förtunnade sammanslagna bilden. Detta genomförs med hjälp av *Shi-Tomasis hörnigenkänningsalgoritm* [28]. Parametervärdet som avgör minsta kvalitetsnivå på ett detekterat hörn kan sättas väldigt lågt i och med att komponenter har jämna kanter och risken för att detektera falska hörn är liten. Vidare möjliggör detta också att hörn som blivit rundade ändå uppnår en kvalitetsnivå som överstiger minimikravet. De registrerade hörnpunkterna sparas i en lista. Därefter förbereds bilddatan för nästa steg genom en förtjockning av komponenterna med användning av morfologisk utvidgning [29].

2.6 Takplan i 2D

I detta steg skapas polygoner som beskriver planen som utgör en byggnads tak sett uppifrån. Detta framställs genom att kombinera informationen från de segmenterade och behandlade bilderna, de hittade hörnen och byggnadspolygonerna. Först omvandlas bilddatan till geodetiskt refererade geometrier. Sedan kopplas geometrierna från bilddatan ihop med tillhörande byggnadspolygon, dessa processer finns mer utförligt beskrivna i appendix C. Slutligen filtreras och justeras geometrierna för att passa respektive byggnadspolygon bättre.

Bilddatan är uppbyggd som en $m \times n$-matris där varje element är en pixel i bilden. Eftersom

bilderna är binärt segmenterade är pixlarna antingen svarta eller vita. I detta steg av processen omvandlas alla områden med vita pixlar till en polygon som omsluter området. Med hjälp av den tillhörande geodetiska referensen till varje bild kan bildmatrisen omvandlas till koordinater. Det betyder att istället för att hantera datan som en matris där varje pixel har ett värde samt en separat georeferens hanteras nu datan som polygoner bestående av ihopkopplade punkter i ett geodetiskt refererat koordinatsystem, se figur 8a och 8b.



(a) Svartvit bild, lagrad som en matris

(b) Polygonerna med punkter i rosa och linjer i vitt

(c) Byggnadspolygon i rosa, hörnpunkter i rött och polygoner i vitt

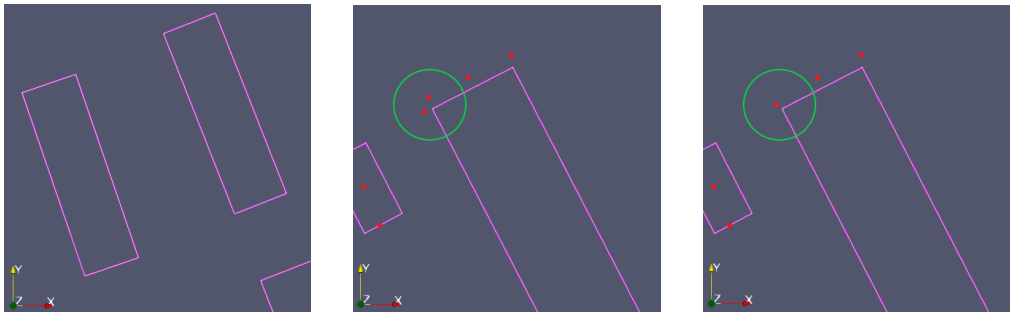
Figur 8: Omvandlingen från bilddata till polygoner bestående av sammankopplade punkter i ett kartesiskt koordinatsystem. (c) Visar även byggnadspolygonerna för respektive byggnad.

För att enkelt kunna hantera all data kopplas alla geometrier extraherade från bilden till respektive byggnadspolygon. I figur 8c visas byggnadspolygoner samt polygoner och hörnpunkter från bilderna. Dessa paras ihop genom att en omgivande rektangel ritas runt varje byggnadspolygon. De hörnpunkter och polygoner från bilddatan (härefter bildpolygoner) som ligger innanför rektangeln kopplas till byggnadspolygonen. Mer specifikt så är första steget i denna process att omvandla geometrierna från bilddatans koordinater till det lokala koordinatsystemet som används för punktmolnet. Geometrierna lagras i en speciell träd-datastruktur anpassad för spatiala data. Genom att lagra datan i ett träd så minskar beräkningskomplexiteten vilket ger en algoritm som är praktiskt användbar i större skala [30]. Även byggnadspolygonerna lagras i en träd-datastruktur. Anledningen till detta är för att närliggande byggnader ska kunna kopplas till respektive byggnad. Denna information behövs för att säkerställa att två tak inte går igenom varandra.

2.6.1 Generering av polygon i 2D

I de nästkommande stegen används den information som extraherats för att skapa en verklighetstrogen takgeometri i 2D. För varje byggnadspolygon kombineras hörnpunkterna, bildpolygonerna och eventuella närliggande byggnadspolygoner för att skapa polygoner för varje plan på respektive tak. Algoritmen är uppbyggd så att varje byggnadspolygon itereras över en gång. För varje byggnadspolygon genomförs de steg som presenteras nedan. Algoritmen finns mer utförligt beskriven i appendix D.

Det första steget är att byggnadspolygonen läses in. I samband med inläsningen beräknas antalet par av parallella linjer i byggnadspolygonen. En rektangulär byggnadspolygon med räta vinklar resulterar alltså i två par av parallella linjer. Se figur 9a för exempel på rektangulära byggnadspolygoner.



(a) Två byggnadspolygoner (b) Hörnpunkter i rött och byggnadspolygoner i rosa (c) Hörnpunkter i rött och byggnadspolygoner i rosa

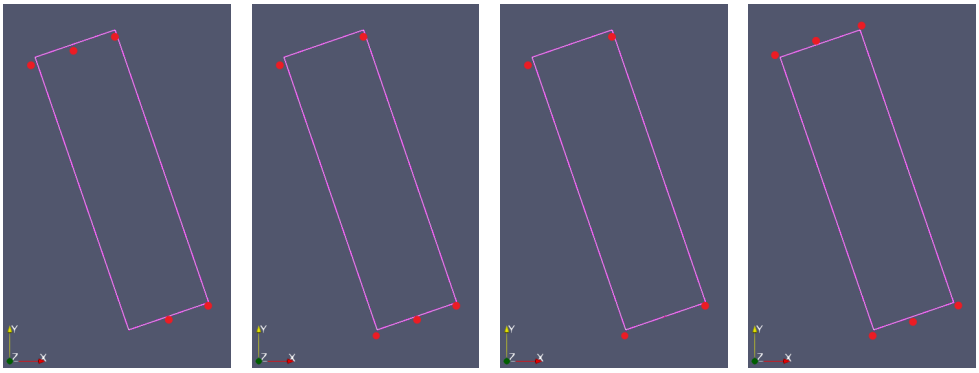
Figur 9: Hantering av hörnpunkter. (a) Vid inläsning hittas två par med parallella linjer för respektive byggnadspolygon. (b) Två hörnpunkter som representerar samma hörn på byggnaden. (c) Efter filtrering har punkterna från b slagits ihop till en punkt.

De hörnpunkter som extraherats under bildbehandlingen filtreras för att rensa eventuella fel. Det första felet som kontrolleras och åtgärdas är om två punkter ligger för nära varandra, alltså att det har genererats fler än en punkt inom en yta på taket som är för liten för att vara meningsfull. Det andra felet som kontrolleras är om en punkt ligger för långt utanför byggnadspolygonen. I första steget undersöks en hörnpunkt åt gången. För varje punkt hämtas alla punkter som ligger tillräckligt nära. Om det är punkter som ligger tillräckligt nära slås dessa ihop till en ny punkt genom att beräkna en medelpunkt mellan alla närliggande punkter. Om punkten nu ligger tillräckligt nära byggnadspolygonen så sparas den. I figur 9b och 9c visas ett exempel på två punkter som ligger för nära varandra och därför slås ihop till en punkt.

Med filtrerade punkter kan nu takets ytterkanter genereras. Denna process består i stort av tre delar: identifiering av ytterpunkter, generering av nya punkter och justering av punkternas positioner. Med takets ytterpunkter menas alla hörnpunkter som tangerar eller ligger utanför byggnadspolygonen och är nödvändiga för att spänna upp den polygon som omsluter takets ytterkanter. För att bestämma vilka punkter detta är så markeras alla punkter som ligger utanför eller tillräckligt nära en linje tillhörande byggnadspolygonen. Anledningen till att punkter som ligger innanför byggnadspolygonen måste tas med är på grund av fel i datan som extraherats från de segmenterade bilderna. Figur 10a exemplifierar en byggnad där den övre högra punkten är en ytterpunkt men ligger just innanför byggnadspolygonen. Sedan filtreras punkter som ej är nödvändiga för att spänna upp takets ytterpolygon genom att vinkeln mellan de två linjer som går ut från varje punkt beräknas. Mellan figur 10a och figur 10b har punkten i mitten på den övre kortsidan filtrerats bort.

Med generering av nya punkter menas att en ny, artificiellt bestämd punkt läggs till i listan av punkter för byggnaden. Detta steg behövs eftersom bildbehandlingsalgoritmerna kan missa hörn. I figur 10a saknas en punkt i det nedre vänstra hörnet. Punkter läggs till så länge byggnadspolygonen är större än den yta som bildas då takets ytterkanter kopplas ihop. Var den nya punkten ska läggas till bestäms genom att lägga till en av byggnadspolygonens punkter i taget. Den punkt som bidrar till störst ökning av area väljs ut och läggs till med ett bestämt avstånd utanför byggnadspolygonen. Avstånden beror på hur långt övriga ytterpunkter ligger utanför byggnadspolygonen. I figur 10b har en ny punkt lagts till nere till vänster.

Justering av takets ytterpunkter behövs av två anledningar. Den första är de fall då en byggnadskropp inrymmer fler än en byggnadspolygon. Det kan till exempel vara radhus eller flerbostadshus som är uppdelat i flera byggnadspolygoner. Sådana byggnader ligger under ett och samma tak men består av flera byggnadspolygoner, se figur 25 (appendix C). I dessa fall måste punkternas positioner vara exakt rätt för att skapa sammanhängande takytor. Den andra anledningen är för att skapa en mer verklighetstrogen geometri. Om punkterna som spänner upp ytterkanternas geometri nästan är vinkelräta genomförs justeringar för att göra vinklarna räta. Till sist anpassas punkter som inte används för ytterkanterna men låg nära genom att de förflyttades till närmsta ytterkantslinje. Figur 10 exemplifierar steg från ovanstående processer.



(a) (b) (c) (d)

Figur 10: Justering av polygoners punkter. (a) En byggnadspolygon och de 5 hörn som hittats under bildbehandlingen. (b) En punkt har lagts till i nedre vänstra hörnet och mittenpunkten på övre kortsidan har filtrerats bort. (c) Mittpunkten på nedre kortsidan har filtrerats bort. (d) Alla punkter för taket efter justeringar enligt beskrivna metoder.

Med korrekt yttergeometri för taket kan de polygoner som omsluter takplanen skapas. Det tilltänka tillvägagångssättet för att hitta alla plan på taket var att koppla samman punkterna som hör ihop och skapa en graf-datastruktur. Alla plan på taket kan då hittas genom att bestämma alla cykler i grafen. Punkterna i en cykel spänner upp ett polygonplan. Punkterna kopplas samman genom att en linje dras mellan alla kombinationer av punktpar. Om linjen inte korsar en annan punkt och ligger innanför en bildpolygon så sparas kopplingen. Se figur 8 för exempel på bildpolygoner. Implementeringen av denna metod misslyckades dock och en annan enklare *brute force*-algoritm användes istället.

2.7 Utvärdering av punktmoln

För att skapa takmodeller i 3D kombineras nu polygonerna som extraherats i föregående avsnitt med ett punktmoln. Ett icke nödvändigt men användbart steg i generering av 3D-geometri är att utvärdera punktmolnet. Detta kan göras för att få större kunskap om indatan och för att evaluera dess kvalitet. Utvärderingen gjordes för varje byggnad i en del av Hammarkullen. För en byggnad ritades dess punktmoln och fotavtryck upp och jämfördes med byggnadens satellitfoto. Utifrån detta kunde punktmolnet analyseras med avseende på var avvikande punkter ofta dök upp, vilka typer av byggnader som ofta gav problematik och andra faktorer som påverkade punktmolnets kvalitet negativt.

2.8 Para ihop datapunkt med polygon

De polygoner som extraheras i avsnitt 2.6 beskriver takets form. Vid 3D-modelleringen av tak ska ett plan anpassas till varje polygon och på så sätt beskriva en byggnads takgemoetri. För att genomföra denna anpassning krävs information om vilka datapunkter från punktmolnet som ingår i vilken polygon. Varje datapunkt behöver alltså paras ihop med en polygon och för detta används *quadrant angle-algoritmen*.

Algoritmen beräknar för varje punkt dess omloppstal med avseende på polygonen. Omloppstalet för en punkt p med avseende på den stängda polygonen P, är antal varv P omsluter p [31]. Om omloppstalet är lika med noll ligger punkten utanför polygonen, annars befinner den sig inuti eftersom den då måste omslutas av polygonens hörn. Det finns ett smidigt sätt att beräkna detta omloppstal utan att behöva använda sig utav tidskrävande trigonometriska funktioner:

- Fixera en punkt p. Låt $[q_0, ..., q_{n-1}]$ vara polygonens hörn i moturs ordning.
- Låt $v_i \in \{0, 1, 2, 3\}$ vara vilken kvadrant q_i befinner sig i med avseende på p.
- Låt delvinkeln $dv_i = v_{i+1} - v_i$, dvs. hur många kvadranter en polygonkant passerar.

- Om $dv_i = 0$ betyder det att motsvarande polygonkant befinner sig helt i en kvadrant, inga kvadrantgränser passeras, och inget händer.
- Om $dv_i = 1$ eller -3, sätts $dv_i = 1$, vilket betyder att polygonkanten passerar en av kvadrantgränserna i moturs riktning och en kvarts moturs rotation runt p sker.
- Om $dv_i = -1$ eller 3, sätts $dv_i = -1$, vilket betyder att polygonkanten passerar en av kvadrantgränserna i medurs riktning och en kvart medurs rotation runt p sker.
- Om $dv_i = 2$ eller -2, krävs vidare undersökning för att avgöra om en halv medurs eller en halv moturs rotation runt p sker. Om vinkeln mellan q_i, p och q_{i+1} är större än 180° byter dv_i tecken. Hur denna beräkning går till presenteras i detalj under appendix F.1.

- Omloppstalet kan nu beräknas som summan av delvinklarna dv_i.

I detta projekt har *quadrant angle-algoritmen* implementerats i Python enligt beskrivningen i appendix F. Med denna metod kan varje polygon, som beskriver en del av takgeometrin, få ett eget punktmoln vilket krävs för nästa steg.

2.9 Plangenerering med singulärvärdesuppdelning

När ett punktmoln har erhållits för varje polygon i taket behövs en uppskattning för vilken höjd polygonens olika hörn kommer ha. Ett plan kan anpassas till punktmolnet och hörnpunkternas höjd beräknas utifrån planets ekvation. För att anpassa ett plan till punktmolnet används en metod beskriven i [32]. Antag att vi har ett punktmoln bestående av 3D-punkter $\{p_1 \dots p_n\}$ och vill skapa ett plan beskrivet av en normalvektor $\boldsymbol{n}$ och en punkt p_c i planet. Det ortogonala avståndet mellan en punkt p och planet kan på matrisform skrivas $(p-p_c)^T\boldsymbol{n}$. Vi vill minimera summan av avståndet mellan alla punkter och planet, varför vårt problem kan skrivas

$$\min_{\|\boldsymbol{n}\|=1} \sum_{i=1}^{n} ((p_i - p_c)^T \boldsymbol{n})^2. \tag{1}$$

Att lösa för p_c ger $p_c = (\bar{x}, \bar{y}, \bar{z}) = \frac{1}{n}\sum_{i=1}^{n}(x_i, y_i, z_i)$ [32]. Låt nu A vara en $3 \times n$-matris bestående av elementen $[p_1 - p_c, p_2 - p_c, \dots p_n - p_c]$. Vi kan omformulera ekvation (1) till

$$\min_{\|\boldsymbol{n}\|=1} \|A^T\boldsymbol{n}\|_2^2. \tag{2}$$

Med singulärvärdesuppdelning på A får vi

$$\begin{aligned} ||A^T\boldsymbol{n}||_2^2 = ||(USV^T)^T\boldsymbol{n}||_2^2 = ||VS^TU^T\boldsymbol{n}||_2^2 = ||S^TU^T\boldsymbol{n}||_2^2 = ||S^T\boldsymbol{y}||_2^2 = \\ = ||(\sigma_1 y_1, \sigma_2 y_2, \sigma_3 y_3)||_2^2 = (\sigma_1 y_1)^2 + (\sigma_2 y_2)^2 + (\sigma_3 y_3)^2 \end{aligned} \tag{3}$$

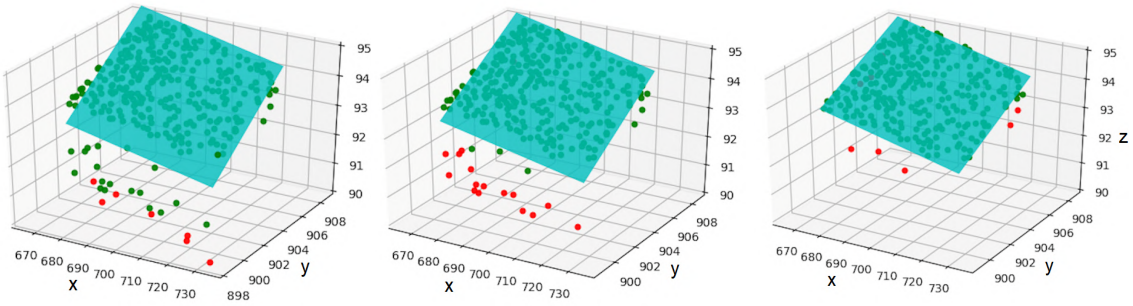
där vi har låtit $\boldsymbol{y} = U^T\boldsymbol{n}$. Eftersom $\sigma_1 \geq \sigma_2 \geq \sigma_3$ ser vi att (2) minimeras för $\boldsymbol{y} = (0, 0, 1)^T$ givet att $||\boldsymbol{n}|| = 1$. Eftersom U är en ortogonal matris, ty den uppkom från singulärvärdesuppdelningen, är detta ekvivalent med att $\boldsymbol{n} = (U_{1,3}, U_{2,3}, U_{3,3}) = U_{\boldsymbol{3}}$. Vi vet nu normalen för planet och en punkt det passerar igenom. Därmed är planet fullständigt bestämt.

2.10 Filtrering av avvikande datapunkter

Den laserdata som använts för att generera takplanen kan innehålla avvikande punkter, till exempel från närliggande träd eller byggnadens väggar. För att dessa inte ska påverka planens lutning måste punktmolnet filtreras. När vi nu har skapat ett ungefärligt plan som representerar taket i den aktuella polygonen, kan vi använda oss av det planet för att hitta avvikande punkter och exkludera dem. Givet en punkt $p = (p_x, p_y, p_z)$ och ett plan $P : ax + by + cz + d = 0$ får vi det ortogonala avståndet mellan dem genom

$$D = \frac{|ap_x + bp_y + cp_z + d|}{\sqrt{p_x^2 + p_y^2 + p_z^2}}. \tag{4}$$

Om $D > \xi$ för ett förutbestämt tröskelvärde ξ så exkluderas p, och ett nytt plan beräknas med det filtrerade punktmolnet. Detta kan ske iterativt, så att det nya planet och ett mindre ξ används för att gallra ytterligare punkter. Fördelen med att dela upp filtreringen i iterationer är att strängare villkor kan användas. Om tröskelvärdet för taket i figur 11 omedelbart hade satts till 0,5 m, så hade vissa punkter felaktigt exkluderats då det preliminära takplanet är för brant.

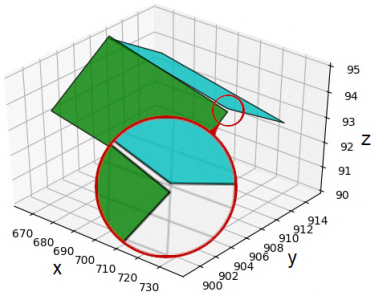


(a) $\xi = 2$ m (b) $\xi = 1$ m (c) $\xi = 0.5$ m

Figur 11: Tre iterationer av gallring. De röda punkterna har hamnat utanför det tillåtna området och tas därmed bort.

2.11 Sammanslagning av gemensamma hörnpunkter

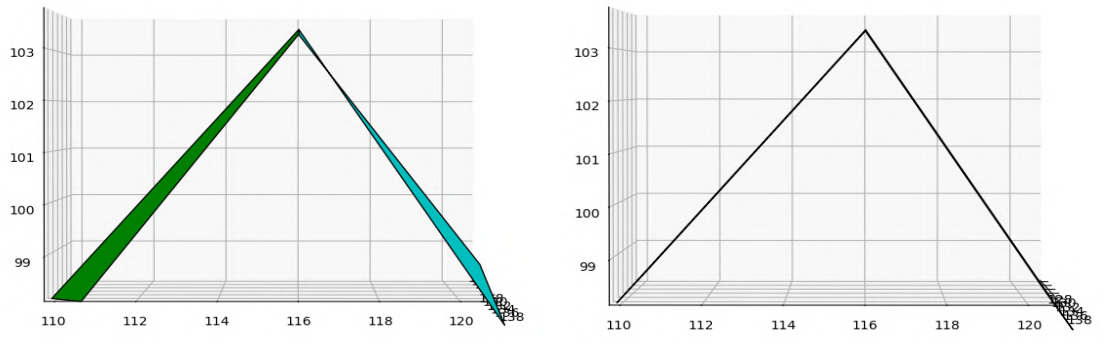
Två polygoner som angränsar till varandra kommer ofta dela vissa hörnpunkter, förutsatt att det inte finns ett avstånd i z-led mellan dem som utgörs av en vägg. När de preliminära takplanen beräknas från punktmolnen kommer z-värdet för en hörnpunkt sannolikt skilja sig från dess motsvarighet i en grannpolygon, på grund av laserdatans inexakthet. Detta orsakar glapp mellan polygonerna, se figur 12.



Figur 12: Två hörnpunkter som borde representeras av en gemensam punkt.

För att undvika dessa glipor hittar vi sådana dubblettpunkter och låter dem referera till ett gemensamt punktobjekt. För att identifiera och avgöra vilka punkter som ska slås samman undersöks avståndet mellan två punkter dels i xy-led, dels i z-led. Om avstånden understiger vissa tröskelvärden skapas ett nytt punktobjekt vars koordinater sätts till ett genomsnitt av de jämförda punkterna.

Skälet till att avstånden i xy-led och z-led undersöks separat är för att de kan bero på olika faktorer. Skillnader mellan dubblettpunkter i sidled kan komma från mindre fel i segmenteringen eller numeriska fel, så motsvarande tröskelvärde kan sättas till någon decimeter. Höjdskillnader mellan dubblettpunkter uppstår när takplanen får felaktiga lutningar, vilket kan orsakas av brister i laserdatan eller förskjutna segmenteringslinjer. Detta gör att toleransen för höjdskillnaden behöver vara större; om laserdatan är av låg kvalitet, takplanen branta och taknocken förskjuten i sidled kan z-värdet för två lika punkter i värsta fall närma sig en meter. Dock finns det även skäl att låta z-toleransen vara lägre än så, då två hörnpunkter som har samma x- och y-värden kan ha olika z-värden för att de har en vägg mellan sig, och då inte bör slås samman.



(a) (b)

Figur 13: (a) Modell av tak innan optimeringsproblemet appliceras. (b) Modell av tak efter optimeringsproblemet appliceras.

2.12 Optimering för planarisering

Efter att gemensamma hörnpunkter slagits samman kommer hörnpunkterna inte alltid ligga i samma plan, se figur 13a, vilket givetvis inte är önskvärt. S. Goebbels har formulerat ett linjärt optimeringsproblem som löser detta problem [33]. Grundtanken är att alla hörnpunkter flyttas i höjdled så att hörn tillhörande de olika segmenterade polygonerna hamnar i samma plan. Beslutsvariablerna representerar hur mycket hörnpunkterna förflyttas i höjdled. Målfunktionen är att minimera summan av alla punkters förflyttningar. Begränsningarna formuleras så att takplanen är platta vilket är synonymt med att planets ekvation uppfylls för de olika planen i taket.

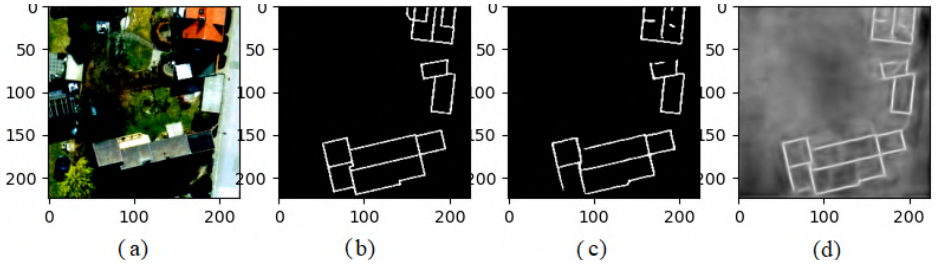
Av numeriska skäl är det omöjligt för planen att vara exakt platta. Vi formulerar därför först en definition av approximativ platthet som tillåter avvikelser givet en felmarginal. Vi använder definitionen för att formulera optimeringsproblemets begränsningar. Givet tre punkter kan ett matematiskt uttryck för approximativ platthet tas fram, se appendix G för mer detaljer. Optimeringsproblemet implementeras i Python med SciPy modulen *linprog*. Om ett plan var mycket platt innan optimeringen applicerades kan resultatet bli att taket blir mindre platt. Vi beräknar därför takets platthet enligt (6) innan och efter och väljer den modell av tak som ger bäst resultat. Eftersom optimeringsproblemet är linjärt och en byggnad generellt inte har många hörn går det snabbt att hitta en lösning.

3 Resultat

Resultaten av varje steg beskrivet i metoden presenteras nedan. Resultaten presenteras i två delar på grund av vilken data som använts vid respektive steg. Under ”Generering av 2D-geometri” presenteras resultaten av stegen från maskininlärningen till generering av takpolygoner i 2D. Dessa steg kördes på datan från Helsingborg. ”Generering av 3D-geometri” presenterar resultaten för övriga steg i metoden då takpolygonerna omvandlats till 3D med hjälp av punktmoln. I dessa steg användes data över Hammarkullen samt manuellt segmenterade takpolygoner.

3.1 Generering av 2D-geometri

Efter att ha tränat maskininlärningsmodellen utvärderades bästa modell. Resultatet mättes i form av andelen korrekt klassificerade pixlar, med bästa resultat: 98,61 procent. Träningsförlusten blev 0,1107 och valideringsförlusten blev 0,1982. I figur 14 visas resultatbilderna i jämförelse med en del av orginalbilden samt tillhörande segmenteringsmask.



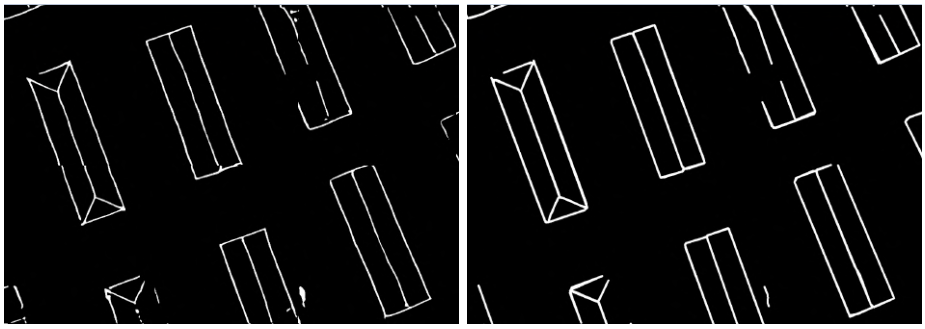
Figur 14: En jämförelse av indatabilder (a) och (b), och färdig segmentering (c) och (d). (a) Indatabild för träningen. (b) Manuellt segmenterad bild för träningen. (c) Binärt justerad förutsägelse efter tränad modell. (d) Icke-binärt justerad förutsägelse efter tränad modell.

Ur tabell 2 ser man att bildbehandlingen resulterade i färre komponenter och större genomsnittligt antal pixlar per komponent, något som indikerar en mer sammanhängande geometri. Vidare har också antal hörnpunkter minskat efter bildbehandling som följd av att oönskade och falska hörn är mindre frekventa. Jämförelsen av antal hörnpunkter förutsätter samma parametrar för *Shi-Tomasis hörnigenkänningsalgoritm* [28].

Tabell 2: Jämförelse av hörnpunkter och komponenter före och efter bildbehandling.

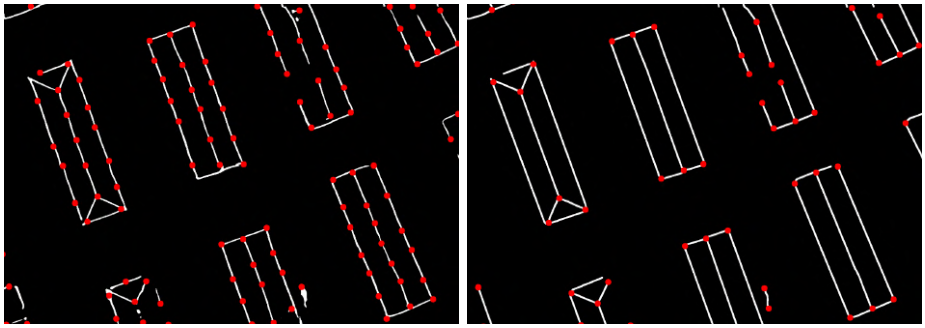
Bildbehandling	Antal hörnpunkter	Antal komponenter	Genomsnittligt antal pixlar per komponent
Före	2035	1309	891
Efter	1086	186	8879

Ur figur 15 noteras att resultatet av bildbehandlingen är en tydligare geometri där komponenter som skall representera linjer är jämnare. Många komponenter har sammanfogats och gliporna är färre. De minsta komponenterna har rensats bort. För några byggnader är både inner- och yttergeometri helt sluten som önskat.



Figur 15: Urklipp område från den sammanslagna bilden. (a) Ursprunglig bild. (b) Behandlad bild.

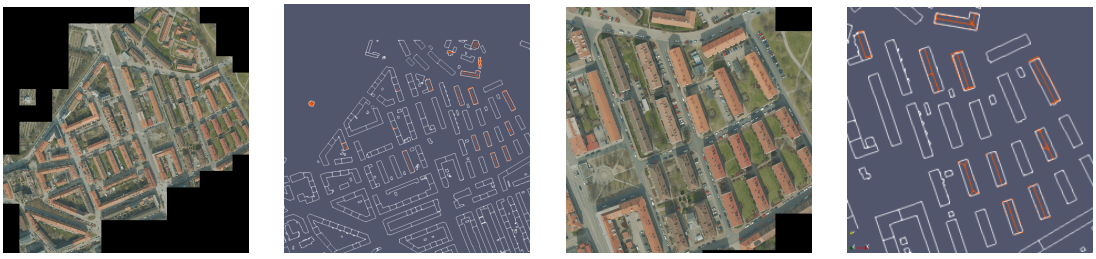
Antal detekterade hörnpunkter som presenteras i figur 16 minskar efter bildbehandling, där det förekommer färre oönskade och falska hörnpunkter. Precisionen för punkternas position har också ökat, som följd av att hörnigenkänningsalgoritmen kördes på den sammanslagna bilden med förtunnade komponenter. För några byggnader kvarstår endast de hörnpunkter som är relevanta för fullständig generering av både inner- och yttergeometri.



(a) (b)

Figur 16: Detekterade hörnpunkter ritade i rött för ett urklipp från den sammanslagna bilden. (a) Ursprunglig bild. (b) Behandlad bild.

Att på ett exakt sätt mäta hur bra algoritmerna som hanterar processen från bild till polygoner i 2D är svårt. Detta eftersom det inte finns någon korrekt data att jämföra med. För att kunna göra en korrekt utvärdering skulle det först krävas att manuellt rita upp polygoner med korrekta geografiska koordinater för byggnader och sedan jämföra hur långt den manuellt framtagna polygonen ligger ifrån polygonen skapad av algoritmerna. Därav skattas endast hur väl de genererade takpolygonerna i 2D ser ut att stämma med verkligheten. I området Eneborg med 66 segmenterade hustak ser 9 stycken ut att vara indelade i polygoner på ett korrekt eller nästan korrekt sätt. I figur 17 redovisas resultatet från området. I figur 18 och 19 visas två exempel på byggnader i Eneborg där takets plan hittats.



(a) Ortofotografi (b) Byggnadspolygoner i vitt och takpolygoner i orange (c) Förstoring av övre högra hörnet i a (d) Förstoring av övre högra hörnet i b

Figur 17: Redovisning av de takpolygoner som extraherats från ortofotografi över Eneborg.

(a) Ortofotografi (b) Takpolygoner ovanpå ortofotografi (c) Byggnadspolygon i vitt och takpolygoner i orange

Figur 18: Byggnad i Eneborg med nästan korrekta takpolygoner. (c) Visar att takpolygonerna ligger lite förskjutet i förhållande till byggnadspolygonen.

(a) Ortofotografi (b) Takpolygoner ovanpå ortofotografi (c) Byggnadspolygon i vitt och takpolygoner i orange

Figur 19: Byggnad i Eneborg med korrekta takpolygoner.

3.2 Generering av 3D-geometri

Resultatet av utvärderingen av punktmolnet var ett dokument där information som ansågs värdefull sammanställdes, både för fortsatt arbete inom projektet och för DTCC. Dokumentet, som finns i sin helhet under appendix E, kan vara ett bra redskap för framtida arbete med att utforma metoder som förbättrar datan, till exempel att minimera avvikande punkter. Som ett resultat av undersökningen implementerades ett paket för filtrering av avvikande punkter i DTCC:s befintliga kodbas, vilket förbättrade punktmolnens kvalitet. För projektet skapades också en egen metod för att hantera avvikande punkter under genereringen av geometrin. Metoden finns beskriven under avsnitt 2.10.

Resultaten av den framtagna algoritmen för att generera geometri mäts både kvalitativt och kvantitativt. För att bedöma huruvida ett tak anses korrekt modellerat eller ej togs två mått fram. Först studeras det totala avståndet mellan datapunkterna och det genererade taket. Eftersom takplanen inte är garanterat platta låter vi tre hörnpunkter spänna upp en approximation av varje takplan. Låt n beteckna antalet datapunkter och D_i avståndet i meter mellan en punkt p_i och takplanet enligt (4). Vi beräknar totala felet enligt

$$\frac{1}{n}\sum_{i=1}^{n} D_i =: \gamma. \tag{5}$$

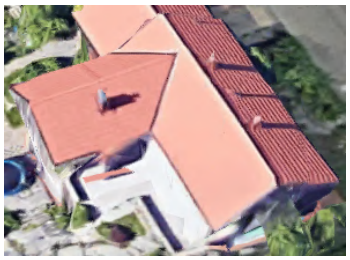
Det studeras också hur platta takplanen är, eftersom det i teorin är möjligt att ha ett takplan som passar väl till datapunkterna men där alla hörn inte ligger i samma plan. För att bedöma hur platt ett takplan är beräknar vi avståndet mellan planet och dess hörn. Låt samma tre punkter som ovan spänna upp planet, antag att vi har k st hörn och beräkna enligt

$$\frac{1}{k}\sum_{i=1}^{k} D_i =: \sigma. \tag{6}$$

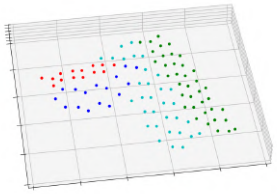
Detta kan tolkas som det genomsnittliga avståndet i meter varje hörn behöver flyttas för att planet ska vara helt platt.

För att få en bild av hur väl gruppens algoritm genererar geometri studeras tak inledningsvis kvalitativt. Figur 20 visar ett exempel. För att få en utförligare bild av hur de modellerade taken ser ut i jämförelse med de avbildade byggnaderna, se figurer i appendix H. Det kan konstateras att algoritmen klarar av att modellera många olika typer av tak, utan att några parametrar behöver justeras. De taktyper som ger bäst resultat är stora sadeltaktak, vilka generellt har fler datapunkter och ibland inte behöver optimeras för att bli helt platta.

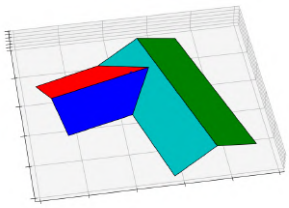
Kvantitativa resultat tas fram genom att låta algoritmen generera geometri åt 25 manuellt segmenterade byggnader på samma gång. Sedan beräknas σ och γ för alla byggnader och om $\sigma < 0.05$ och $\gamma < 0.4$ anser vi byggnaden korrekt modellerad. Se appendix H för en mer detaljerad analys av gränsvärden på σ och γ. Gruppen har gjort ett urval av byggnader för att så väl som möjligt representera en verklig stad. Byggnaderna består av 10 st sadeltak, 5st valmade tak, 5 st platta tak och 5 st komplexa tak. Med komplexa tak menas till exempel taken i figur 20 och 36. Algoritmen körs på en vanlig arbetsdator. Resultaten presenteras i tabell 3.



(a) Satellitfoto av byggnaden



(b) Punktmolnet med punkter uppdelade i olika segment



(c) Genererad geometri

Figur 20: Sammansatt sadeltak. $\sigma = 0.035$, $\gamma = 0.248$

Tabell 3: Resultat av generering av 3D-geometri

Antal korrekta byggnader	**Medelvärde σ**	**Medelvärde γ**	**Tid (s)**
21/25	0.027	0.176	3

4 Diskussion

I detta avsnitt analyseras hela processen från bild till fullständiga 3D-modeller av hustak. Vidare diskuteras möjliga förbättringsområden för framtida arbete inom fältet och en analys av samhälleliga och etiska aspekter presenteras. Slutligen sammanfattas rapporten och slutsatser dras från erfarenheterna av projektet.

4.1 Utvärdering av bildsegmentering till polygoner i 2D

Resultatet av maskininlärningsmodellen är bra men det finns utrymme för förbättring. Det främsta problemet med många maskininlärningsmodeller är risken att överanpassa en modell till träningsdatamängden. Med överanpassning menas att modellen lär sig en träningsdatamängd bra och presterar dåligt på okänd data. Generellt sett är modellen överanpassad om valideringsförlusten är mycket högre än träningsförlusten. Modellen lär sig mönster som av misstag stämmer i träningsdata men inte har en grund i verkligheten, och är således inte sanna i valideringsdatan. Vårt resultat är inte överanpassat vilket gav oss ett bra resultat. Ett sätt att ytterligare förbättra modellen hade varit att ta in mer träningsdata. Detta visade sig vara behövligt då försök att träna på data från Helsingborg och förutspå på data från Hammarkullen i Göteborg misslyckades. Oavsett är det svårt att få en modell att lyckas till 100%, därför är det nödvändigt att göra vidare behandling av linjer och hörn.

Bildbehandligen uppnår ett resultat med fler korrekta hörnpunkter och en ytter- och innergeometri som förbättrar möjligheterna till polygonbildning, vilket också var det inledande syftet. Samtidigt är det långt ifrån tillräckligt bra för att konsekvent ge goda resultat för alla byggnader i Eneborg. Till viss del kan detta beskyllas på kvaliteten på de binärt segmenterade bilderna erhållna från maskininlärningsmodellen, men mycket kan också förbättras vad avser arbetsgången i bildbehandlingsdelen.

Trots användning av *probabilistisk Hough-linje-transform* [22] i två iterationer återstår flera glipor mellan komponenter som i bästa fall hade sammanslutits. Detta är en konsekvens av att algoritmens ingående parametrar vad avser minsta kvalitetskrav och största glipstorlek är mycket restriktiva för att förhindra oönskad sammanslutning mellan två komponenter. En möjlighet kan vara att implementera ännu fler iterationer av *probabilistisk Hough-linje-transform* med varierande parametrar, väl avvägt sådant att de täcker igen glipor av varierande storlek utan att skapa oönskade sammanslutningar mellan komponenter.

Implementeringen av två olika förtunningsalgoritmer förbättrade resultatet vad avser korrekta hörnpunkter. Samtidigt återstod en problematik med att det bildades grenar i områden där komponenten var ojämn. Det är möjligt att implementera en algoritm som skulle kunna ta bort dessa grenar baserat på deras längd, men detta gjordes aldrig, på grund av tidsbrist. Vidare förekom

flera exempel av approximerade linjer genererade via *Ramer-Douglas-Peuckers algoritm* [26] som tog hänsyn till grenarna och resulterade i ojämnheter vilka detekterades som hörn när *Shi-Tomasis algoritm* [28] kördes. Se figur 24 (appendix B.3) för exempel på detta.

Byggnader kan skilja sig åt mycket i geometriskt utseende, vilket gör att utvalda parametrar fungerar olika bra för varje byggnad. Ett bättre tillvägagångssätt skulle kunna vara att behandla varje byggnad för sig och sedan bestämma algoritmernas ingående parametrar baserat på typen av geometri som byggnaden har. Detta skulle vara möjligt att implementera med hjälp av byggndaspolygonerna. Kring byggnadspolygonen skulle man kunna sätta upp en avgränsning som endast tillåter bildbehandling inom det avgränsade området. Därefter genomförs en bildbehandling på varje enskild byggnad med optimerade parametrar.

De två största anledningarna till att resultatet från skapandet av polygoner för taken i 2D inte blev bra är uppdelningen av byggnadspolygonen och att de genererade bilderna har en viss förskjutning i förhållande till byggnadspolygonen. Problemen med uppdelningen av byggnadspolygonen är att under ett tak kan det rymmas flera byggnadspolygoner. Se figur 25 (appendix C) för exempel på ett hus med flera byggnadspolygoner under ett tak. Som algoritmerna är skrivna nu ska ett tak genereras för varje byggnadspolygon. Det fungerar inte när hörnpunkterna i taket ligger för långt ifrån byggnadspolygonen. Det hade därför varit bättre att som ett första steg slå ihop alla byggnadspolygoner som ligger intill varandra och sedan skapa ett stort och sammanhängande tak för hela den huskroppen. Den andra försvårande omständigheten var att segmenteringarna som skapats med maskininlärningsmodellen har en förskjutning i förhållande till byggnadspolygonens position. Förskjutningen kan vara åt alla riktningar och längden på förskjutningen är också oregelbunden, se figur 25 (appendix C). Ansatsen som gjordes för att åtgärda detta var för dålig och fungerar inte för en godtycklig byggnadspolygon. Trots dessa misstag så har även denna delen av projektet uppfyllt rapportens syfte. Projektet gick ut på att undersöka, utveckla och utvärdera metoder och skulle de brister som diskuteras här och i appendix C och D åtgärdas hade resultaten troligen blivit bättre.

4.2 Resultat av generering av geometri i 3D

Vi inleder med att återigen konstatera att alla byggnader som genereras för den kvalitativa analysen i appendix H använder exakt samma algoritm och parametrar. Detta är den främsta styrkan i vår databaserade algoritm. Den klarar av att hantera många olika typer av byggnader och behöver inte göra approximationer eller förenklingar för att sortera in varje byggnad i en fördefinierad katalog likt en modellbaserad. Dessutom kommer vår algoritm kunna utnyttja en förväntad ökning av tätheten i punktmoln i framtiden. Priset som betalas för denna generalitet är att beräkningstiden kan öka och att takplanen inte alltid är helt plana.

Enligt de mått som tagits fram genererar vår algoritm korrekta modeller för 21 av 25 fall. De tak som inte blir korrekta är en blandning av komplexa tak, se exempelvis figur 36, och stora valmade tak. För stora tak ger icke-planaritet ett större utslag för vårt mått σ. Nästan alla tak uppfyller gränsen $\gamma < 0.5$, så vill vi förbättra resultaten bör fokus ligga vid att förbättra planariseringen hos takplanen. Vår algoritm är inte optimerad för prestanda. Utvecklingen har skett i Python och koden innehåller många for-satser och andra aspekter som ökar beräkningstiden. Om DTCC väljer att implementera den kommer koden skrivas om till C++ och förbättras för att minska antalet beräkningar vilket bör öka prestandan avsevärt.

Rimligheten hos våra mått σ och γ kan diskuteras. Givetvis ger hårdare gränsvärden för dessa färre korrekta byggnader och tvärtom. Måtten och gränsvärdena togs fram genom analys av de kvalitativa resultaten (se appendix H). Det finns många sätt att kvantifiera resultat av virtuella modeller och i slutändan valde gruppen det som tycktes mest relevant.

I teorin bör vår metod beskriven i avsnitt 2.12 säkerställa att alla takplan är platta inom en liten felmarginal. Resultaten visar däremot att så inte alltid är fallet. För de flesta enklare byggnader fungerar det väl, men för mer komplexa byggnader är planen inte alltid platta trots att en lösning till optimeringsproblemet har erhållits. För några få byggnader blir planen mindre platta efter att optimeringsproblemet har applicerats på dem. Vi har ännu inte funnit en definitiv orsak till detta. En teori är att vår formulering av optimeringsproblemet är fel, exempelvis genom att en begränsning för optimeringsproblemet skrivits in fel (detta ser vi dock som osannolikt då formuleringen är baserad på en vetenskaplig artikel av en professor vid ett universitet). En mer

sannolik anledning är numeriska fel och valet av optimeringslösare. Vi använder SciPy-modulen *linprog*. Den är enkel att använda och passar för projekt av undersökande karaktär, men är inte på samma nivå som en mer dedikerad optimeringslösare, exempelvis GLPK [34]. Ett framtida förbättringsområde är att implementera optimeringsproblemet i en bättre optimeringslösare vilket förhoppningsvis ger bättre resultat.

I detta projekt används en databaserad strategi, den främsta anledningen till det är att möjligheterna anses större med denna strategi. Målet med forskningsprojektet DTCC är att skapa digitala tvillingar av städer och om detta ska gå att uppnå måste alla möjliga typer av tak kunna generas, även de mest komplexa. Att fördefiniera alla möjliga takformer i en katalog skulle vara en nästintill omöjlig uppgift. Med en modellbaserad strategi skulle förenklingar av många takformer krävas vilket skulle leda till sämre modellering av byggnader som i sin tur skulle leda till mindre korrekta digitala tvillingar. Som tidigare nämnt så är de databaserade metoderna ofta mer känsliga för kvaliteten på punktmolnet. Det punktmoln som används i detta projekt är glest med omkring $1-2$ punkter/m^2 och DTCC har förhoppningar om att densiteten på punktmolnet ska ökas, för vissa platser så mycket som 10 gånger. Det är något som talar för att projektets framtagna metod har goda förutsättningar att generera bättre resultat i framtiden.

4.3 Framtida arbete

I huvudsak är det två framtida problemställningar som vi tror kan vara intressant att utreda. Den första är att man bör utreda hur byggnadspolygoner och punktmoln kan utnyttjas redan under bildsegmenteringsfasen med maskininlärningsmodellen. Den andra är att man bör utreda hur en data- och modellbaserad metod kan kombineras. Detta har redan testats [11] med goda resultat, och är därför av stort intresse.

Bakgrunden till det första förslaget är att metoden vi utvecklade gjorde sig beroende av att bildsegmenteringen erhållen från maskininlärningsmodellen var tillförlitlig. Men trots modellens goda resultat presenterat i avsnitt 3.1 skapades många misslyckade segmenteringar. Samtidigt fanns tillgång till data i form av byggnadspolygoner och punktmoln över området, som skulle kunna användas tidigare i vår metod. Byggnadspolygoner ger information om var byggnader finns, och punktmolnen skulle kunna utnyttjas för att avgöra var det bör vara ett tak. Således kan bildsegmenteringen bli mer tillförlitlig då den har mer data i olika former att utgå ifrån.

Det andra förslaget grundar sig i att vår databaserade metod påverkas negativt av glesa punktmoln. Den problematik som beskrivits i avsnitt 4.2 rörande optimeringsproblemet hade blivit mindre omfattande om ett tätare punktmoln erhållits. Samtidigt är det också viktigt att metoden fungerar tillräckligt bra på glesa punktmoln, då täta punktmoln över andra områden inte kan garanteras. Förslagsvis skulle takgenerering kunna inledas med en modellbaserad teknik för byggnader med lägre komplexitet som finns fördefinierade i en katalog, för att sedan gå vidare med att hantera mer komplexa takformer med den databaserade metoden som utvecklats i projektet. Ett annat tillvägagångssätt kan vara att applicera båda metoder på alla tak och sedan välja det bästa genererade taket efteråt. Detta förväntas kräva mer kompileringstid då det krävs att alla tak generas två gånger, men samtidigt försäkras också bästa möjliga utfall.

4.4 Samhälleliga och etiska aspekter

Som tidigare nämnts kommer detta arbete eventuellt bli en del av forskningsprojektet DTCC, vars syfte är att utveckla digitala tvillingar till Sveriges städer [1]. Även fast takmodeller som i större utsträckning efterliknar verklighetens tak utgör en liten del av en komplett digital tvilling, ökar nyttoaspekterna. Det tydligaste exemplet är att byggnader och stadsbilden bättre kan visualiseras, vilket underlättar vid planering och beslutsfattning kopplat till stadens fysiska utveckling.

Sett ur ett större perspektiv har sådana digitala tvillingar som DTCC utvecklar även många andra samhällsnyttiga aspekter. En övergripande stadsmodell möjliggör simuleringar av olika slag, vilka kan användas för att optimera lösningar och undgå kostsamma misstag. Ett sådant exempel är utformningen av trafiknätverk; genom att simulera trafikflöden för olika vägalternativ kan den lösning väljas ut som minimerar köer, buller eller risken för olyckor. Andra positiva aspekter möjliggörs av sensordata som uppdateras i realtid, så att modellen alltid återspeglar verkligheten

i största möjliga utsträckning. På så sätt kan exempelvis utryckningsfordon bli uppdaterade om vilka rutter som borde väljas för att undvika trafikstockningar.

Den plattform som DTCC:s digitala tvilling utgår ifrån kommer baseras på öppen källkod och data. Trots att känslig data inte används kan en sådan omfattande insamling och behandling av data medföra vissa risker. Flera olika former av rådata som separat inte utgör känslig information kan kombineras och därmed ge upphov till ny information, som kanske inte borde vara allmänt tillgänglig. Information som i allmänhetens händer gör stor nytta kan dock också användas i skadliga syften. Om en stad faller offer för en väpnad konflikt skulle en 3D-modell av staden vara mycket användbar, då den skulle kunna underlätta att planera angrepp eller identifiera strategiskt viktiga platser och utsiktspunkter.

Eftersom forskningsprojektet kommer baseras på öppen källkod finns det också en risk att andra implementerar egna digitala tvillingar med syften som inte nödvändigtvis är samhällsnyttiga. Detta skulle till exempel kunna ske om en statlig aktör har tillgång till känslig data som kan användas för att identifiera och övervaka medborgare. Därmed är det viktigt att för varje ny typ av data som inkorporeras, analysera eventuella risker som kan uppstå från sådana kombinationer av data. Gruppen kan dock inte identifiera några omedelbara risker som uppkommer från inkluderandet av de framtagna takmodellerna. På grund av detta, och eftersom de risker som nämnts ovan inte ännu är aktuella och ligger utanför arbetets omfattning, har vi inte tagit några beslut kopplat dessa samhälleliga och etiska aspekter.

4.5 Slutsats

I denna rapport har vi undersökt och arbetat med en mängd metoder för att automatiskt generera digitala takmodeller med detaljnivå LOD2. Alla metoder som använts har uppvisat lyckade resultat i åtminstone ett antal fall, dock har vissa komplikationer hindrat processen från att bli helt generell och automatisk.

Maskininlärningsmodellen som används för att skapa binärt segmenterade bilder på hustaken hittar inte alla linjer som utgör takens utmärkande drag. Vidare inkluderas vissa segment som egentligen inte är en del av taklinjerna. Denna problematik hade förmildrats av en större mängd och mer varierad träningsdata, och åtgärdas i nuläget till viss del i efterbehandlingen av bilderna: små överflödiga komponenter rensas bort och andra sammanfogas där glipor uppstått. Linjerna görs även rakare och tunnare vilket underlättar framtagningen av korrekt placerade hörnpunkter. Dock finns det en oregelbunden förskjutning mellan byggnadspolygoner från fastighetskartan och geometrier från maskininlärningsmodellen. Denna förskjutning har vi inte lyckats hantera för alla typer av byggnader och tak.

Efter att segmenteringarna slutförts är de tänkta att användas tillsammans med punktmolnet för att generera 3D-geometrierna. Eftersom ett konsekvent bra resultat inte hann uppnås användes istället manuellt segmenterade tak i utformningen av 3D-algoritmerna. Punktmolnet delas upp så alla datapunkter som ligger i en viss polygon kan isoleras. Vissa avvikande punkter från andra objekt än själva taket kan komma med, men rensas enkelt bort om punktdatan till största del består av relevanta datapunkter. Punkmolnet tillhörande vissa byggnader är för glest för att vara användbart, och skulle då behöva kompletteras eller ersättas med ett alternativt tillvägagångssätt som inte förlitar sig lika tungt på laserdatan. Optimeringen som sker för att göra takplanen helt platta lyckas med detta i de flesta fall, men korrigerar ibland inte taken helt och hållet.

Projektets omfattning tyder på att en helt automatiserad framtagning av korrekt utformade takmodeller inte är en enkel uppgift. Det finns fortfarande många problem att lösa och hinder att passera innan metodiken kan integreras i en digital tvilling och bidra med bättre takmodeller. Dock har vi kommit en bra bit på vägen, och både hoppas och tror att DTCC kommer kunna dra nytta av de framsteg och resultat som vi bidragit med.

Referenser

[1] Digital Twin Cities Centre, "Digital twin cities centre", 2021. URL: https://dtcc.chalmers.se/, (hämtad: 2021-05-11).

[2] IATE European Union Terminology, "Artificiellt neuronnät", 2021. URL: https://iate.europa.eu/search/standard/result/1620743015543/1, (hämtad: 2021-05-05).

[3] M. Valueva, N. Nagornov, P. Lyakhov, N. Chervyakov, M. Liu och Y. Zhan, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation", *Mathematics and Computers in Simulation*, årg. 177, 2020, ISSN: 0378-4754. DOI: 10.1016/j.matcom.2020.04.031.

[4] Lantmäteriet, "Vad är geodesi?", 2021. URL: https://www.lantmateriet.se/contentassets/bebb1b3574b446728d2b531f0e8d84f6/vad_ar_geodesi.pdf, (hämtad: 2021-05-05).

[5] F. Biljecki, H. Ledoux och J. Stoter, "An improved lod specification for 3d building models", *Computers, Environment and Urban Systems*, årg. 59, 2016, ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2016.04.005, License: Creative Commons BY-NC-SA.

[6] C. Kennedy, J. Cuddihy och J. Engel-Yan, "The changing metabolism of cities", *Journal of Industrial Ecology*, årg. 11, 2 2007, ISSN: 10881980. DOI: 10.1162/jie.2007.1107.

[7] V. Maliene, V. Grigonis, V. Palevičius och S. Griffiths, "Geographic information system: Old principles with new capabilities", *Urban Design International*, årg. 16, 1 2011, ISSN: 13575317. DOI: 10.1057/udi.2010.25.

[8] B. Ketzler, V. Naserentin, F. Latino, C. Zangelidis, L. Thuvander och A. Logg, "Digital twins for cities: A state of the art review", *Built Environment*, årg. 46, 4 2020, ISSN: 0263-7960. DOI: 10.2148/benv.46.4.547.

[9] N. Haala och M. Kada, "An update on automatic 3d building reconstruction", *ISPRS Journal of Photogrammetry and Remote Sensing*, årg. 65, s. 570–580, nov. 2010. DOI: 10.1016/j.isprsjprs.2010.09.006.

[10] F. Tarsha-Kurdi, T. Landes, P. Grussenmeyer och M. Koehl, "Model-driven and data-driven approaches using lidar data: Analysis and comparison", *PIA 2007 - Photogrammetric Image Analysis, Proceedings*, jan. 2007.

[11] Y. Zheng och Y. Zheng, "A hybrid approach for three-dimensional building reconstruction in indianapolis from lidar data", *Remote Sensing*, årg. 9(4), mars 2017. DOI: 10.3390/rs9040310.

[12] Lantmäteriet, "Ortofoto - produktbeskrivning", aug. 2020. URL: https://www.lantmateriet.se/globalassets/kartor-och-geografisk-information/flyg--och-satellitbilder/ortofoto.pdf, (hämtad: 2021-05-05).

[13] ESRI, "What is raster data?", 2016. URL: https://desktop.arcgis.com/en/arcmap/10.3/manage-data/r, (hämtad: 2021-05-05).

[14] Lantmäteriet, "Laserdata nh - produktbeskrivning", sept. 2020. URL: https://www.lantmateriet.se/globalassets/kartor-och-geografisk-information/hojddata/laserdata_nh_v2.10.pdf, (hämtad: 2021-05-05).

[15] S. K. Sim och A. H. Duffy, "A foundation for machine learning in design", *AI EDAM-Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, årg. 12, nr 2, s. 193–209, 1998.

[16] B. Mehlig, *Machine learning with neural networks*, 2021. arXiv: 1901.05639 [cs.LG].

[17] Labelbox, "Introduction to image segmentation for machine learning and ai", 2021. URL: https://labelbox.com/image-segmentation-overview, (hämtad: 2021-04-29).

[18] D. Guo, Y. Pei, K. Zheng, H. Yu, Y. Lu och S. Wang, "Degraded image semantic segmentation with dense-gram networks",
IEEE Transactions on Image Processing, årg. 29, s. 782–795, 2020.
DOI: 10.1109/TIP.2019.2936111.

[19] A. Krizhevsky, I. Sutskever och G. E. Hinton, "Imagenet classification with deep convolutional neural networks",
Advances in neural information processing systems, årg. 25, s. 1097–1105, 2012.

[20] O. Ronneberger, P. Fischer och T. Brox, "U-net: Convolutional networks for biomedical image segmentation", *CoRR*, årg. abs/1505.04597, 2015.
arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[21] Google, "Colaboratory - Frequently Asked Questions",
URL: https://research.google.com/colaboratory/faq.html, (hämtad: 2021-05-14).

[22] C. G. Jiri Matas och J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform",
Computer Vision and Image Understanding, årg. 78, nr 1, s. 119–137, 2000.

[23] D. B. Costantino Grana och R. Cucchiara, "Optimized block-based connected components labeling with decision trees",
IEEE Transactions on Image Processing, årg. 19, nr 6, s. 1596–1609, 2010.

[24] T. Y. Zhang och C. Y. Suen, "A fast parallel algorithm for thinning digital patterns",
Image Processing and Computer Vision, årg. 27, nr 3, s. 236–239, 1984.
URL: http://agcggs680.pbworks.com/f/Zhan-Suen_algorithm.pdf.

[25] S. Suzuki och K. Abe, "Topological structural analysis of digitized binary images by border following", *Computer Vision, Graphics, and Image Processing*, årg. 30, nr 1, s. 30–46, 1985.
URL: http://pdf.xuebalib.com:1262/xuebalib.com.17233.pdf.

[26] D. Douglas och T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature",
The Canadian Cartographer, årg. 10, nr 2, s. 112–122, 1973.
DOI: https://utpjournals.press/doi/10.3138/FM57-6770-U75U-7727.

[27] "Parallel thinning with twosubiteration algorithms",
Communications of the ACM, årg. 32, nr 3, s. 359–373, 1989.

[28] S. Birchfield och C. Tomasi, "A pixel dissimilarity measure that is insensitive to image sampling", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, årg. 20, nr 4, s. 401–406, 1998. URL:
http://robotics.stanford.edu/~birch/publications/dissimilarity_pami1998.pdf.

[29] R. C. Gonzales och R. E. Woods, "Digital image processing",
Addison-Wesley Publishing Company, s. 655–657, 1992.

[30] N. Wiegand, "Review of spatial databases with application to gis by philippe rigaux, michel scholl, and agnes voisard. morgan kaufmann 2002.",
ACM SIGMOD Record, årg. 32, nr 4, s. 111–112, 2003.

[31] K. Hormann och A. Agathos, "The point in polygon problem for arbitrary polygons",
Computational Geometry, årg. 20, nr 3, s. 131–144, 2001, ISSN: 0925-7721.
DOI: https://doi.org/10.1016/S0925-7721(01)00012-8. URL:
https://www.sciencedirect.com/science/article/pii/S0925772101000128.

[32] I. Söderkvist, "Using svd for some fitting problems", 2009.
URL: https://www.ltu.se/cms_fs/1.51590!/svd-fitting.pdf, (hämtad: 2021-05-05).

[33] S. Goebbels, R. Pohle-Fröhlich och J. Rethmann,
"Planarization of CityGML Models Using a Linear Program",
i *Operations Research Proceedings 2016*, ser. Operations Research Proceedings,
A. Fink, A. Fügenschuh och M. J. Geiger, utg., Springer, dec. 2018, s. 591–597.
DOI: 10.1007/978-3-319-55702-1. URL:
https://ideas.repec.org/h/spr/oprchp/978-3-319-55702-1_78.html.

[34] A. Makhorin, “Gnu linear programming kit”, juni 2012.
URL: https://www.gnu.org/software/glpk/, (hämtad: 2021-05-14).

[35] Bigironsphere, “Loss Function Library - Keras PyTorch”, *Kaggle*, jan. 2021. URL:
https://www.kaggle.com/bigironsphere/loss-function-library-keras-pytorch,
(hämtad: 2021-05-13).

[36] Torch Contributors, “Torch.optim”, 2019.
URL: https://pytorch.org/docs/stable/optim.html, (hämtad: 2021-05-13).

[37] G. Bradski, “The OpenCV Library”, *Dr. Dobb's Journal of Software Tools*, 2000.
URL: https://github.com/opencv/opencv, (hämtad: 2020-05-05).

[38] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen och D. C. et. al,
“Array programming with NumPy”, *Nature*, årg. 585, nr 7825, s. 357–362, sept. 2020. DOI:
10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

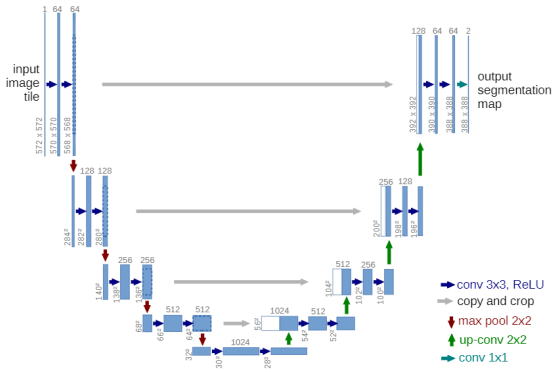
[39] S. Gillies et. al, “Rasterio”, 2016.
URL: https://github.com/mapbox/rasterio/blob/master/docs/intro.rst,
(hämtad: 2021-05-05).

[40] S. Gillies, “Shapely: Manipulation and analysis of geometric objects”, 2007.
URL: https://github.com/Toblerity/Shapely, (hämtad: 2021-05-05).

[41] K. Jordahl, “Geopandas: Python tools for geographic data”, 2014.
URL: https://github.com/geopandas/geopandas, (hämtad: 2021-05-05).

A Maskininlärningsmodell

Här är en schematisk figur på maskininlärningsmodellen som används i metodavsnitt 2.4 vid namn U-Net. Beskrivning av nätverket finns i avsnitt 2.3.2.



Figur 21: Input bilden (*input image tile*) inmatas till vänster i bilden. Segmenterade masken (*output segmentation map*) utmatas från höger i bilden. Horisontella pilarna (*conv*) syftar på CNN-lager och har som uppgift att hitta huvuddragen. Uppåtpilar (*max pool*) samt nedåtpilar (*up-conv*) resulterar i en förminskad samt förstorning av bild. [20]

A.1 Parametrar för maskininlärningsmodell

Här presenteras en kort sammanfattning av parametrar för maskininlärningsmodell, samt vad som användes vid träning av bästa modell i avsnitt 2.4.

- ***Batch*-storlek** - Stora datamängder är ofta uppdelade i så kallad *batches*. Istället för att varje enskilt element av datamängden matas in till träningsmodell kan man mata in en *batch*. *Batch*-storlek anger antalet element i en batch. I vår modell användes *batch*-storlek 32.
- **Epok** - En epok är en mått som beskriver när en maskininlärningsalgoritm har tränat på hela datamängden en gång. Det optimala antalet epoker som är nödvändigt för att träna en bra modell är beroende på antalet parametrar som är relaterade till datamängden själv men också målet med modellen. I vår modell användes 200 epoker för att träna modellen.
- **Förlustfunktion** - I neurala nätverksmodeller är förlustfunktion ett mått på totala felet för varje träningsbatch. Detta påverkar justeringen hur vikterna ska ändras i träning. Därav har valet av förlustfunktion en direkt påverkan på modellens prestation. Standardvalet för förlustfunktionen för segemntering och klassificeringsproblem är *Binary Cross-Entropy* (BCE) [35], vilket även användes i vår modell.
- **Inlärningshastighet** - En inställningsparameter i en optimeringsalgoritm som bestämmer stegstorleken vid varje iteration medan den går mot ett minimum av en förlustfunktion. Inlärningshastigheten styr hur snabbt modellen anpassas till målet. Mindre inlärningshastigheter kräver fler träningsepoker då vikterna ändras långsammare, medan högre inlärningshastigheter resulterar i snabba förändringar och kräver färre träningsepoker. I den valda optimeringsalgorithmen för vår modell valdes en inlärningshastighet på 0,0001.
- **Optimeringsalgoritm** - Används i neurala nätverk för att ändra attribut så som vikter och inlärningshastighet för att minska förlustfunktion. Den enklaste algoritmen är *Stochastic gradient descent* (SGD) [36]. I vår modell används: *Adaptive Moment Estimation* (ADAM) [36].

B Bildbehandling, utförlig beskrivning

I detta appendix presenteras en mer utförlig beskrivning över hur de algoritmer som redogjorts i avsnitt 2.5 använts. Tillhörande python script med kommentarer som beskriver kod finns tillgänglig på *GitLab*, se appendix I. Externa bibliotek som använts är OpenCV [37] och Numpy [38].

B.1 Funktioner och parametrar

Efter assemblering av binärt segmenterade bilder genererade från maskininlärningsmodellen till en sammanslagen bild genomfördes bildbehandlingen. I klassen *MosaicTilesToImg* körs funktionen *ApplyImageProcessing* med alla algoritmer i ordning, vilka läses via *utils*.

Inledningsvis appliceras *probabilistisk Hough-linje-transform* [22] i två iterationer med hjälp av funktionerna HoughLinesColor och HoughLinesGray i *utils* med varierande parametrar enligt det presenterat i tabell 4. ”rho” och ”theta” bestämmer avstånds- och vinkelupplösning i pixlar. ”threshold” avgör minsta krav för att spara linjesegment. ”minLineLength” respektive ”maxLineGap” avgör minsta tillåtna antal pixlar för linjesegment och högsta tillåtna glipstorlek i pixlar ett linjesegment får spänna över.

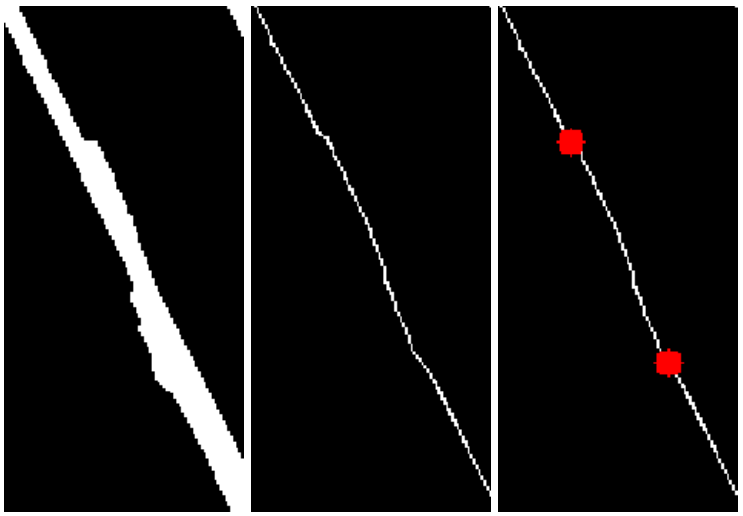
Tabell 4: Ingående parametrar för respektive iteration av *probabilistisk Hough-linje-transform*

Funktion i *utils*	rho	theta	threshold	minLineLength	maxLineGap
HoughLinesColor	8	$\pi/360$	600	60	20
HoughLinesGray	8	$\pi/360$	5000	200	50

Borttagning av mindre komponenter med användning av *Grana, Borghesani och Cucchiaras algoritm* [23] görs med hjälp av funktionen RemoveBlobs, där parametervärdet ”min_size” för minsta tillåtna antal pixlar per komponent sattes till 500. Förtunningsalgoritmerna *Zhang Suen* [24] och *Guo-Hall* [27] genomförs med hjälp av funktionen ZhangSuenLineThinning respektive GuoHallLineThinning i *utils*. Linjeutjämningen med hjälp av *Ramer-Douglas-Peuckers algoritm* [26] och *Abe och Suzukis algoritm* [25] genomfördes via funktionen DouglasPeucker i *utils*. Den konstant som bestämmer hur avvikande ett linjesegment får vara från den nya approximerade formationen sattes till 0,002.

Avslutningsvis appliceras *Shi-Tomasis hörnigenkänningsalgoritm* [28] med hjälp av funktionen FindCornersShiTomasi som har tre ingående parametrar. ”maxCorners” avgör högsta antal tillåtna hörn, och sätts till -1 som innebär att ingen övre gräns finns. ”qualityLevel” bestämmer kvalitetsnivån på detekterat hörn i en skala mellan 0 och 1. Denna sätts till 0,09. ”minDistance” fastställer minsta tillåtna sträcka i pixlar mellan två detekterade hörn, och sätts till 63. Om flera hörn detekteras inom denna sträcka sparas det med högst kvalitetsnivå.

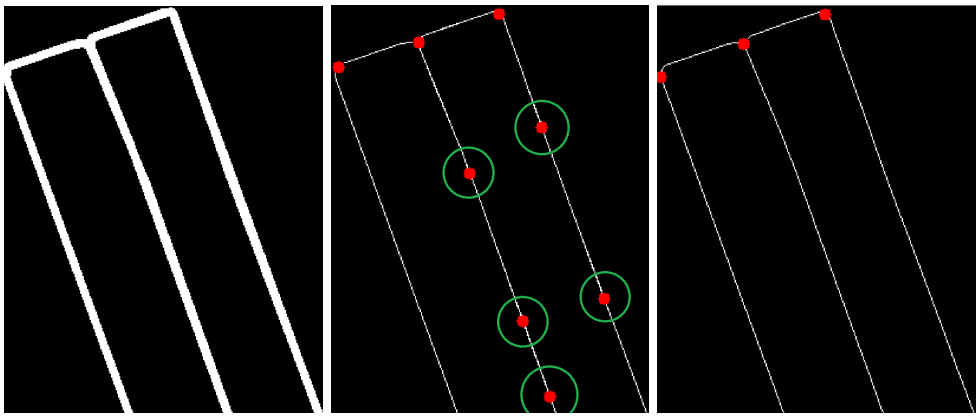
B.2 Hörn vid ojämnheter



(a) (b) (c)

Figur 22: (a) Bild på linje efter *probabilistisk Hough-linje-transform* och borttagning av mindre komponenter. (b) Förtunnad linje med *Zhang-Suens förtunningsalgoritm*. (c) Detekterade hörn i rött av *Shi-Tomasis hörnigenkänningsalgoritm*.

B.3 Jämförelse av förtunningsalgoritmer



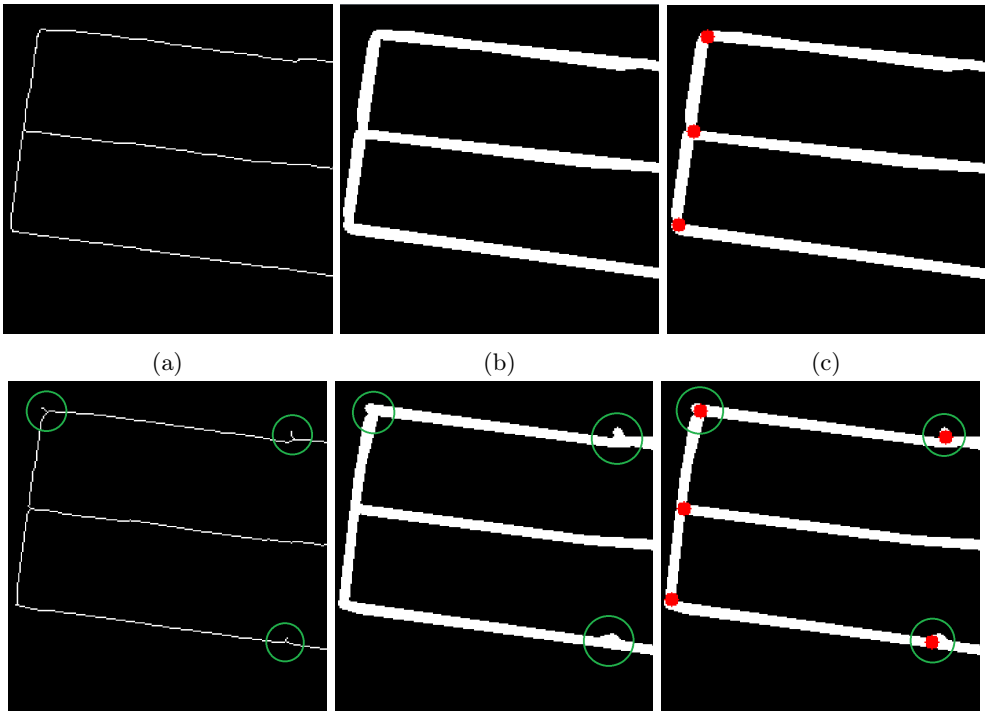
(a) (b) (c)

Figur 23: Detekterade hörn av *Shi-Tomasis hörnigenkänningsalgoritm* i rött, falska hörn inringade. (a) Bild efter linjeutjämning. (b) Förtunnad bild med *Zhang-Suens förtunningsalgoritm*. (c) Förtunnad bild med *Guo-Halls förtunningsalgoritm*.

Med applicering av *Guo-Halls förtunningsalgoritm* i sista steget minskar antal vita pixlar med cirka 10% och hörnpunkter med 38%. Minskningen av antal falska hörn så som presenterat i figur 23 är den största förklaringen till färre detekterade hörnpunkter för *Guo-Halls förtunningsalgoritm*.

Tabell 5: Jämförelse av antal vita pixlar och detekterade hörnpunkter beroende på förtunningsalgoritm.

Förtunningsalgoritm	Antal vita pixlar	Antal hörnpunkter
Zhang-Suen	213779	1746
Guo-Hall	191448	1086



Figur 24: Detekterade hörn av *Shi-Tomasis hörnigenkänningsalgoritm* i rött, position för grenar inringade. (a) Förtunnad bild med *Zhang-Suens förtunningsalgoritm*. (b) Linjeutjämning. (c) Detekterade hörn. (d) Förtunnad bild med *Guo-Halls förtunningsalgoritm*. (e) Linjeutjämning. (f) Detekterade hörn.

C Takplan i 2D, utförlig beskrivning

I detta appendix förklaras stegen som beskrivs i den första halvan av avsnitt 2.6 Takplan i 2D. De viktigaste algoritmerna redovisas och utvärderas baserat på erfarenheter som fåtts under projektets gång. All kod finns öppen på *GitLab* (se appendix I) och de kodfiler, klasser och metoder som refereras till finns där. För varje del redovisas metadata över körexemplet Eneborg. De externa bibliotek som används i algoritmerna presenterade i detta appendix är *rasterio* [39], *shapely* [40] och *geopandas* [41].

C.1 Bilddata till polygoner

När bildbehandlingen är genomförd omvandlas bilddatan till polygoner. Klassen för omvandlingen heter *ImgToPolygons*. Under omvandligen från bilddata till polygoner så bildas polygoner med många punkter. För att förenkla framtida beräkningar med dessa polygoner förenklas polygonernas geometri med Ramer-Douglas-Peuckers algoritm [26]. I tabell 6 redovisas resultat från körning av området Eneborg.

Tabell 6: Metadata för körning av *ImgToPolygons* där bilddata omvandlas till polygoner.

Antal extraherade polygoner	**Tid (s)**
412	3.8

C.2 Koppla bildgeometrier till byggnadspolygoner

I detta steg så paras alla geometrier som är tillräckligt nära en byggnadspolygoner ihop med byggnadspolygonen. Geometrierna som kopplas till byggnadspolygonerna är de polygoner som extraherats från bilddata i föregående steg samt de hittade hörnpunkterna. Klassen som hanterar detta steg heter *GeoMatcher*. Algortimen beskrivs i algoritm 1 och metadata från körning av området Eneborg presenteras i tabell 7.

Algorithm 1: GeoMatcher

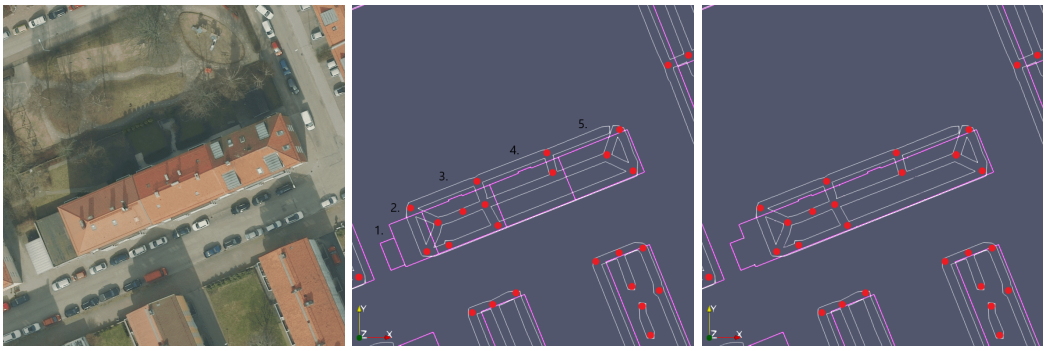
Function `BuildMatchedDataset(`*gdf, otherGDFs*`):`
- **Input:** gdf: GeoDataFrame med byggnadspolygoner. OtherGDFs: Lista med GeoDataFrames innehållande geometrier. En GeoDataFrame är ett tabellobjekt anpassat för geospatial data [41]
- **Output:** Ett dataset med byggnadspolygoner och alla geometrier som kopplats till respektive byggnadspolygon

initiera *buildingData* (output)
ladda geometrierna i *othersGDFs* till ett STR-träd [30], $\rightarrow$ *othTree*
ladda byggnadspolygonerna i *gdf* till ett STR-träd, $\rightarrow$ $footprintTree$
for *byggnadspolygon i gdf* **do**
  skapa omgivande rektangel runt *byggnadspolygon* $\rightarrow$ *boundingBox*
  geometrier från *othTree* innanför *boundingBox* $\rightarrow$ *geomsInside*
  byggnadspolygoner från $footprintTree$ innanför *boundingBox* $\rightarrow$ *adjacentFootprints*
  bifoga *geomsInside*, *adjacentFootprints* till *buildingData*
end
return *buildingData*

Tabell 7: Metadata för körning av *GeoMatcher* (algoritm 1) där geometrier kopplas till byggnadspolygoner.

Antal byggnadspolygoner	**Antal polygoner (RoofPolygons)**
409	884
Antal hörnpunkter (CornerPoints)	**Tid (s)**
1086	1.7

Som diskussionen nämnt är många byggnader uppdelade i flera byggnadpolygoner. En potentiellt stor förbättring av resultaten skulle uppnås om byggnadspolygonerna slogs ihop innan detta steg i processen. Byggnaden som visas i figur 25 skulle då se ut som figur 25c visar och hanteringen i nästkommande steg förenklas. Denna process skulle också förenkla nästkommande steg då polygonerna i 2D ska omvandlas till 3D. Detta eftersom många takplan sträcker sig över flera byggnadspolygoner. Om algoritmen analyserar varje byggnadspolygon för sig måste de extraherade 3D-geometrierna anpassas efter varandra vilket gör processen svårare. En till fördel med att slå ihop byggnadspolygonerna är att det inte ställs lika höga krav på punktmolnets densitet eftersom det räcker med några få punkter på varje takplan för att bestämma dess utbredning i 3D.



(a) Ortofotografi (b) Fem byggnadspolygoner under ett tak (c) Exempel då byggnadspolygonerna har slagits ihop

Figur 25: En byggnad bestående av fem byggnadspolygoner. Byggnaden exemplifierar svårigheter med: flera byggnadspolygoner under ett tak, förskjutningen mellan byggnadspolygonerna och geometrierna från den segmenterade bilden och hur bildsegmenteringen misslyckats att hitta de lägre platta taken som skjuter ut från den vänstra kortsidan.

D Generering av polygon i 2D, utförlig beskrivning

I detta appendix ska stegen som beskrivs i avsnitt 2.6.1 Generering av polygon i 2D förklaras mer djupgående. De viktigaste algoritmerna redovisas och utvärderas baserat på erfarenheter som fåtts under projektets gång. All kod finns öppen på *GitLab* (se appendix I) och de kodfiler, klasser och metoder som refereras till finns där. De externa bibliotek som används i algoritmerna presenterade i detta appendix är *shapely* [40] och *numpy* [38].

Denna del av processen att skapa tekpolygoner i 2D är baserad på datamängden som beskrivs i appendix C. En byggnadspolygon har bildpolygoner, hörnpunker och närliggande byggnadspolygoner kopplade till sig. Algoritmerna nedan appliceras på en byggnadspolygon i taget. Programmet är ligger i modulen *CleanRoof* och är uppbyggt kring klassen *CleanRoof* som sedan kallar på klasserna *Footprint*, *OuterGeometry* och *InnerGeometry*. Algoritm 2 beskriver stegen som genomförs för varje inläst byggnadspolygon. Denna algoritm har som syfte att slussa datan genom en mängd processer och redovisas endast för att visa ordningsföljden av hur de olika behandlingarna av geometrierna utförs.

Algorithm 2: CleanRoof

Function `CleanRoof(`*pythondictionary*`):`
- **Input:** *Python dictionary*-objekt med: footprint (byggnadspolygon), adjacentFootprints (byggnadspolygoner), cornerPoints (hörnpunkter), roofPolygons (bildpolygoner)
- **Output:** En lista med polygoner som beskriver byggnadspolygonens takplan i 2D

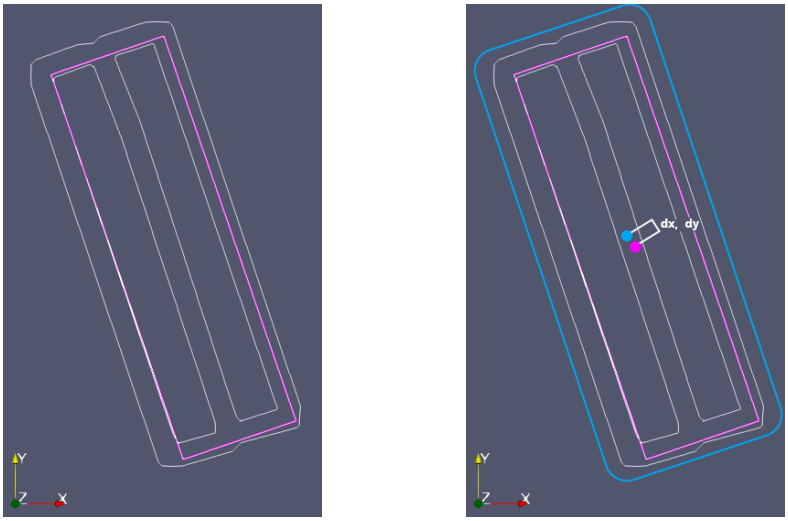
ladda $footprint$ i ett $Footprint$-objekt $\rightarrow fp$
ladda $adjacentFootprints$ i $Footprint$-objekt $\rightarrow adjFPs$
translatera $cornerPoints, roofPolygons$ med $TranslateRoofGeoms, \rightarrow points, linePolygons$
rensa $points$ med $CleanPoints, \rightarrow points$
ladda $fp, adjFPs, points$ i ett $OuterGeometry$-objekt, $\rightarrow outerPoints, points, outerPolygon$
koppla samman $points$ genom att extrahera linjer från $roofPolygons$ med $ExtractLines, \rightarrow rooflines, points$
ladda $points, outerPolygon, roofLines$ i ett $InnerGeometry$-objekt, $\rightarrow innerGeometry$
return $innerGeometry$

Syftet med algoritm 3 är att hantera problemet med att geometrierna extraherade från bilddatan är förskjuten i förhållande till byggnadspolygonen. Detta problemet har diskuterats tidigare och exemplifieras i figur 25 (appendix C). Förutsatt att närliggande byggnadspolygoner slagits samman som diskuterats i appendix C så bör *TranslateRoofGeoms* fungera väl i de flesta fall då segmenteringen är av bra kvalité. Dock så har inte algoritmen undersöks tillräckligt för att säkert säga att den alltid fungerar utan detta måste undersökas mer djupgående. Figur 26 visar hur algoritmen fungerar för en byggnad med enkel geometri.

Algorithm 3: TranslateRoofGeoms

Function `TranslateRoofGeoms(`$cornerPoints, roofPolygons$`):`
- **Input:** cornerPoints (hörnpunkter), roofPolygons (bildpolygoner), *CleanRoof*-objektet (self)
- **Output:** Translaterade hörnpunkter och bildpolygoner

hämta mittpunkten från $fp, \rightarrow fpCentroid$
hämta mittpunkten från unionen av $roofPolygons, \rightarrow geomsCentroid$
translatera $roofPolygons, cornerPoints$ med $(fpCentroid - geomsCentroid), \rightarrow trnasPolygons, transPoints$
return $transPolygons, transPoints$



(a) (b)

Figur 26: Algoritm 3 *TranslateRoofGeoms*. (a) Visar bildpolygoner i vitt och byggnadspolygon i rosa. (b) Det blå omgivande området visar unionen av alla bildpolygoner med en tillagd buffer, mittpunkten i samma färg. Den rosa mittpunkten visar byggnadspolygonens mittpunkt. Bildpolygonerna och hörnpunkterna translateras med dx, dy för att passa byggnadspolygonen bättre.

Funktionerna *CleanPoints* och *ExtractLines* redovisas ej här utan kan ses i kodbasen I. Funktionen *ExtractLines* kopplar samman hörnpunkterna med varandra genom att dra linjer mellan alla par-kombinationer av hörnpunkter. Om linjen inte korsar en punkt eller linje samt ligger innanför en bildpolygon så sparas den. Detta gör att segmenteringen måste vara väldigt bra och små fel kan leda till att punkter ej kopplas ihop vilket leder till att det är omöjligt att skapa takpolygoner. Denna funktion skulle alltså behöva anpassas för att bättre hantera de fallen med bristfällig segmentering.

D.1 Behandling av byggnadspolygoner, *Footprint*

Klassen *Footprint* har som främsta syfte att lagra byggnadspolygoner. Både den analyserade byggnadspolygonen men också de närliggande. Vid initieringen beräknas antalet par av parallella linjer i byggnadspolygonen. Denna information kan sedan användas för att dra slutsatser om hur hörnpunkternas positioner kan justeras. Vet vi att en byggnadspolygon är konvex och har två par av parallella linjer så är det ett rektangulärt hus. Då kan vi också anta att hustakets ytterpolygon är rektangulär. Tanken är att i en mer utbyggd version också kunna dra slutsatser för mer komplexa byggnadspolygoner. Till exempel har byggnader med en "vinkelform" (18) en konkav byggnadspolygon, sex räta vinklar och en långsida som är lika lång som de två andra parallella sidorna tillsammans. Tanken är inte att det ska gå att kategorisera alla typer av byggnader utan det bör gå att generera takpolygoner ändå. Fördelen i de fall det går är att det blir enklare att skapa en perfekt geometri.

D.2 Ytterpunkter, *OuterGeometry*

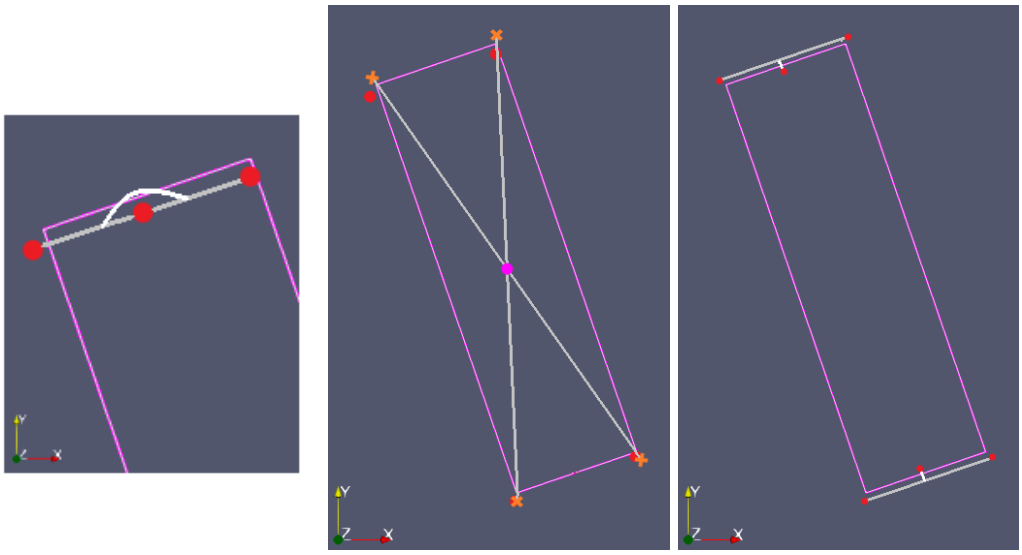
Klassen *OuterGeometry* har syftet att skapa en polygon som omger takets ytterkanter. Denna polygon behövs inte för att sedan skapa ett tak i 3D men tanken var att förenkla justeringen av punkternas positioner. Algoritm 4 visar de processer som genomförs för varje byggnadspolygon.

Funktionerna visas ej i detalj utan förklaras kort i algoritmen och exemplifieras i figur 27.

Algorithm 4: OuterGeometry

1 **Function** `OuterGeometry`$(fp, adjFPs, points)$**:**
  Input: footprint (byggnadspolygon), adjacentFootprints (närliggande byggnadspolygoner), points (hörnpunkter)
  Output: Punkter med justerade positioner
2 extrahera ytterpunkter från $points, ExtractOuterPoints \rightarrow outerPoints$
3 lägg till saknad punkt, $AddMissingPoints \rightarrow outerPoints$
4 justera punkternas positioner, $AdjustPoints \rightarrow outerPoints$
5 anpassa övriga punkter till ytterpunkternas nya positioner, $SnapToLine \rightarrow points$
6 **return** $outerPoints + points$

Av dessa funktioner är *AddMissingPoints* den mest kritiska. Som den är implemeterad nu läggs punkter till oavsett byggnadspolygon-typ (till skillnad från *AdjustPoints* som endast implementeras för rektangulära byggnadspolygoner). Algoritmen *AddMissingPoints* fungerar dock egentligen inte för alla typer av byggnadspolygoner vilket leder till att punkter kan läggas till felaktigt för vissa byggnader. Det som skulle gjorts annorlunda är att antingen skapa en ny generellt applicerbar funktion som lägger till punkter eller definiera en specifik funktion för varje typ av byggnadspolygon.



(a) (b) (c)

Figur 27: (a) Visar hur en punkt filtrera bort i *ExtractOuterPoints* eftersom vinkeln mellan linjerna som går ut från punkten är tillräckligt stor. (b) Visar hur ytterpunkterna justeras i *AdjustPoints* för en byggnadspolygon med två par av parallella linjer. Linjer dras från byggnadspolygonens mitt, genom en punkt på byggnadspolygonen och slutar på ytterpunkternas medelavstånd utanför byggnadspolygonen. Där placeras nu ytterpunkten istället. (c) Punkter som ej är ytterpunkter kopplas till linjen som utgörs av två ytterpunkter i *SnapToLine*. Denna koppling sker när punkterna ligger tillräckligt nära en linje.

D.3 Takpolygoner, *InnerGeometry*

Klassen *InnerGeometry* har syftet att skapa takpolygonen som beskriver takets plan i 2D. Indatan till denna klass är hörnpunkterna som processerats i tidigare steg. Funktionen *ExtractLines* i *CleanRoof* kopplar samman alla punkter i en enkel oriktad graf med avståndet mellan punkterna som vikter. Tanken var då att alla takpolygoner skulle kunna hittas genom att söka efter cykler i grafen med en sökalgoritm. Sökalgrotimen skulle alltså hitta kortaste vägen från en given punkt tillbaka till sig själv. Denna ansats misslyckades dock främst på grund av att tiden för projektet tog

slut. Nu implementeras istället en simplare och inte lika effektiv metod för att hitta takpolygonerna (se kodbasen).

E Utvärdering av punktmoln

Följande är en sammanfattning av det dokument som togs fram under studier av punktmolnet.

Metod

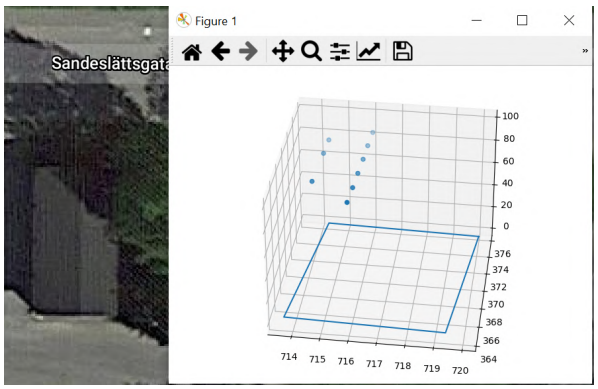
Ett python-script för att förenkla arbetet skapades. Scriptet tog en byggnads punktmoln som indata och plottade alla punkter. Det slog sedan upp byggnadens koordinater i Google Maps för att användaren enkelt skall kunna jämföra byggnadens verkliga utseende med punktmolnet. En stor mängd byggnader analyserades, totalt ca 100st. Om det fanns några problem med punkmolnet dokumenterades det. Avslutningsvis analyserades resultaten för att ge en uppfattning av vilka problem som fanns och vilka faktorer som påverkades laserskanningen och därmed punktmolnen.

Resultat

Tre övergripande slutsatser kunde dras, dessa redovisas nedan.

Små fotavtryck ger få datapunkter

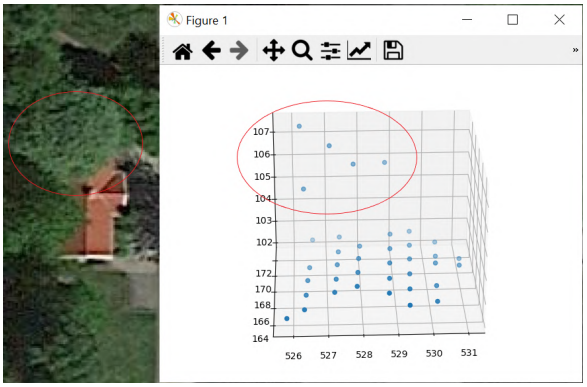
I en del fall resulterade små byggnader i att punktmolnet innehöll väldigt få datapunkter. Anledningen är att tätheten på laserskanningen inte är tillräckligt hög. Det konstaterades att detta är ett problem som inte kan göras något åt då den underliggande orsaken är datan från Lantmäteriet. En bra lösning är istället att anta att små byggnader har platt tak, vilket ofta är fallet i verkligheten.



Figur 28: Exempel på hur små byggnader kan ge få datapunkter i punktmoln

Höjdskillnader på grund av takfönster eller träd

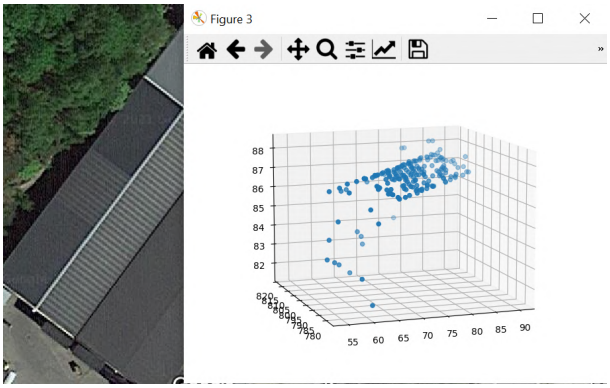
Det är relativt vanligt att det finns objekt, exempelvis träd, som blockerar syn på taket rakt ovanifrån. I dessa fall kommer laserstrålen träffa trädet istället för taket vilket ger avvikande datapunkter. I vissa fall finns fönster vilket då orsakar att laserstrålen träffar innanför fönstret. En lösning på detta kan vara att implementera någon typ av filtrering i punktmolnen. Det kan göras både i Core och under genereringen av taken.



Figur 29: Exempel på hur träd kan ge avvikelser i punktmoln

Avvikelser vid kanter på stora tak

En relativt vanlig företeelse var att hitta avvikande datapunkter vid kanterna på tak, särskilt om taken tillhör stora byggnader. Detta kan lösas genom att bland annat filtrera bort datapunkter nära kanterna.



Figur 30: Exempel på hur det kan finnas avvikande värden vid kanter på byggnader

F *Quadrant angle-algoritmen*

Algorithm 5: Quadrant Angle Algorithm

Function `QuadrantAngle_point_poly(`*p, polygon*`):`
- **Input:** Polygon bestående av hörn, punkt p att undersöka
- **Output:** Om punkten ligger inuti polygonen eller inte
- ta ut första hörnet av ingående polygon $\rightarrow q_0$
- ta reda på vilken kvadrant polygonhörnet befinner sig i med avseende på punkten p, med hjälp av $QuadrantAngle\_point\_point(q_0, p) \rightarrow v_1$
- Initera $totalAngle = 0$
- **for** *polygonhörn* **do**
  - ta ut nästa polygonhörn $\rightarrow q_1$
  - ta reda på vilken kvadrant polygonhörnet befinner sig i med avseende på punkten p, med hjälp av $QuadrantAngle\_point\_point(q_1, p) \rightarrow v_1$
  - beräkna vilka kvadranter polygonkanten rör sig mellan, delvinkel $\rightarrow dv = v_1 - v_0$
  - **if** *polygonkant gått från kvadrant 0 till 3* **then**
    - sätt delvinkel till -1 $\rightarrow dv = -1$
  - **else if** *polygonkant gått från kvadrant 3 till 0* **then**
    - sätt delvinkel till 1 $\rightarrow dv = 1$
  - **else if** *polygonkant rört sig över 2 kvadrantgränser* **then**
    - ta reda på om vinkeln mellan polygonhörnen är mer än 180 grader
    - **if** *vinkeln mellan polygonhörn* $> 180°$ **then**
      - ändra tecken på delvinkeln $\rightarrow dv = -dv$
    - **end**
  - **end**
- **end**
- uppdatera totala vinkeln genom att lägga till delvinkeln $\rightarrow$totalAngle $+= dv$
- uppdatera vilket polygonhörn vi kollar på och vilken kvadrant vi befinner oss i $\rightarrow q_0 = q_1, v_0 = v_1$
- **return** True om totala vinkel är icke-noll

Function `QuadrantAngle_point_point(`*p, q*`):`
- **Input:** Två punkter p och q
- **Output:** Vilken kvadrant p befinner sig i map q
- **if** *x-värdet av punkt p* $>$ *x-värdet av punkt q* **then**
  - **if** *y-värdet av punkt p* $>$ *y-värdet av punkt q* **then**
    - **return** 0
  - **else**
    - **return** 3
  - **end**
- **else**
  - **if** *y-värdet av punkt p* $>$ *y-värdet av punkt q* **then**
    - **return** 1
  - **else**
    - **return** 2
  - **end**
- **end**

F.1 Ska delvinkeln byta tecken?

För att bestämma huruvida vinklen mellan q_0, p och q_1 är större eller mindre än 180° används följande metodik utifrån figur 31:

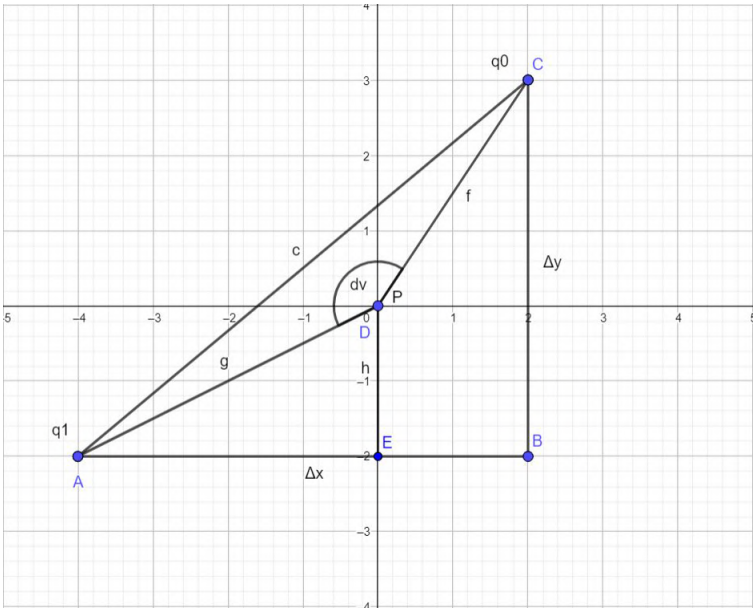
Om olikheten

$$q_{1,x} + p_x > (q_{1,y} - p_y) \cdot \frac{q_{0,x} - q_{1,x}}{q_{0,y} - q_{1,y}} \tag{7}$$

gäller är vinkeln dv i figur 31 större än 180°. Om punkten p sätts i origo och $\frac{q_{0,x}-q_{1,x}}{q_{0,y}-q_{1,y}}$ skrivs $\frac{\Delta x}{\Delta y}$ kan olikheten (7) förenklas till

$$q_{1,x} > q_{1,y} \cdot \frac{\Delta x}{\Delta y} \Leftrightarrow \frac{q_{1,x}}{q_{1,y}} > \frac{\Delta x}{\Delta y}$$

vilket ger en jämförelse mellan de två trianglarna ABC och AED.



Figur 31: Två av polygonens hörn q_0 och q_1 samt punkten p att undersöka.

För att öka tydligheten ges två exempel:

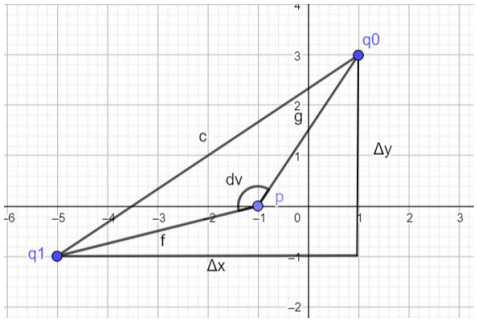
- Om vinkeln dv är större än 180°, som i figur 32, ska följande ifrån (7) gälla:

$$\frac{p_y - q_{1,y}}{p_x - q_{1,x}} < \frac{\Delta y}{\Delta x}$$

Sätts siffror från figuren in:

$$\frac{0 - (-1)}{(-1) - (-5)} < \frac{4}{6} \Leftrightarrow \frac{1}{4} < \frac{4}{6}$$

Och vi kan se att olikheten stämmer.



Figur 32: Två av polygonens hörn q_0 och q_1 samt punkten p att undersöka.

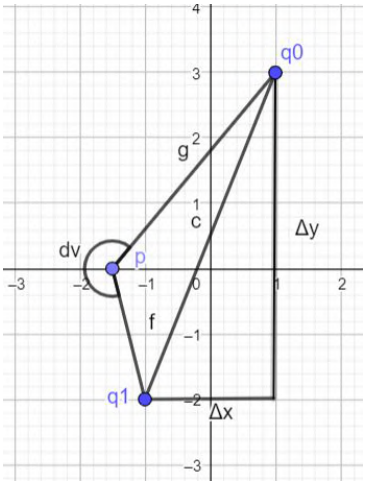
- Om vinkeln dv är mindre än 180°, som i figur 33, ska istället det omvända gälla:

$$\frac{p_y - q_{1,y}}{p_x - q_{1,x}} > \frac{\Delta y}{\Delta x}$$

Sätts siffror från figur in:

$$\frac{0-(-2)}{(-1.5)-(-1)} > \frac{5}{2} \Leftrightarrow \frac{2}{0.5} > \frac{5}{2}$$

Och vi kan se att även denna olikhet stämmer.



Figur 33: Två av polygonens hörn q_0 och q_1 samt punkten p att undersöka.

G Optimeringsproblem för planarisering

Vi ser ett tak som en mängd av n st takplan i form av polygoner $P_k, k \in \{1, 2 \dots n\}$. Varje polygon har en mängd av tre eller flera hörn $p_{k,1}, \dots, p_{k,m_k} \in V$ där $V \subset \mathbb{R}$ är mängden av alla hörn på taket. Varje hörn är en vektor $p_{k,j} = (p_{k,j}.x, p_{k,j}.y, p_{k,j}.z)$. Vårt mål är att ändra hörnens koordinater så att alla hörnen i en polygon hamnar i samma plan. Hörnens x- och y-koordinater härstammar från segmenteringen och byggnadens fotavtryck, det är viktigt att dessa förblir oförändrade eftersom vi vill att vår virtuella modell av staden ska matcha fastighetskartor. Vi begränsar oss därför till att endast modifiera hörnens z-koordinater.

Målet är därmed att för varje hörn hitta en funktion $h(p)$ som ger en ny höjdkoordinat. För att behålla formen på taken är det även önskvärt att förflytta hörnen så lite som möjligt. Vi sätter $h(p) = p.z + h_1(p) - h_2(p)$ där $h_1(p), h_2(p) \geq 0$ vilket innebär att vi för varje hörn adderar eller subtraherar dess höjd med ett tal. Vi formulerar sedan målfunktionen i optimeringsproblemet att minimera $\sum_{p \in V} h_1(p) + h_2(p)$ för att minimera totala förflyttningen av hörnpunkter. För att undvika att hörn flyttas stora avstånd sätter vi dessutom en övre begränsning på variablerna $h_1(p), h_2(p) \leq \epsilon$ med $\epsilon > 0$.

För att formulera begränsningarna i optimeringsproblemet behöver vi först definiera vad en platt polygon innebär. Att enbart säga att alla hörn måste ligga i samma plan fungerar inte då avrundningsfel och numerisk alltid kommer resultera i små fel. Istället ger Goebbels en definition av approximativ platthet. Vi kallar en polygon P_k approximativt platt om och endast om det för alla hörn $p_{k,j}$ $j \in \{1 \dots m_k\}$ i polygonen gäller att höjden $p_{k,j}.z$ skiljer sig från z-koordinaten för planet vid $(p_{k,j}.x, p_{k,j}.y)$ med mindre än:

$$\delta_k := \mu + \frac{\sqrt{1 - v_k.z^2}}{v_k.z} \cdot \sqrt{2}\mu \tag{8}$$

Där μ är en parameter vi väljer själva, lämpligt kan vara att tillåta fel upp till $\mu = 0.001$m. $v_k.z$ är z-koordinaten för polygonens normalvektor.

Vi låter nu tre punkter i polygonen $(p_{k,u}, p_{k,v}, p_{k,w})$ definiera ett plan. Val av punkter kan göras som i [33] för ökad stabilitet. Då de flesta av polygonerna vi arbetar med endast har fyra hörn väljer vi enkelt de tre första elementen i en array av polygonens hörn. Vi tittar nu endast på x- och y-koordinater och antar att $\mathbf{a_k} = (p_{k,u}.x - p_{k,v}.x, p_{k,u}.y - p_{k,v}.y)$ och $\mathbf{b_k} = (p_{k,w}.x - p_{k,v}.x, p_{k,w}.y - p_{k,v}.y)$ är linjärt oberoende. Skulle så inte vara fallet ändrar vi enkelt vilka punkter som definierar vårt plan. Då kan alla övriga punkter $p_{k,j}$ i polygonen skrivas som en linjärkombination av vektorerna enligt $p_{k,j} = p_{k,v} + r_{k,j}\mathbf{a_k} + s_{k,j}\mathbf{b_k}$. Konstanterna $r_{k,j}, s_{k,j}$ kan beräknas genom att med exempelvis Cramers regel lösa ekvationssystemet:

$$r_{k,j}(p_{k,u}.x - p_{k,v}.x) + s_{k,j}(p_{k,w}.x - p_{k,v}.x) = p_{k,j}.x - p_{k,v}.x \tag{9}$$

$$r_{k,j}(p_{k,u}.y - p_{k,v}.y) + s_{k,j}(p_{k,w}.y - p_{k,v}.y) = p_{k,j}.y - p_{k,v}.y \tag{10}$$

Vi är nu redo att formulera begränsningar för platthet i vårt optimeringsproblem. Det gäller att en punkt i vårt optimeringsproblem $h(p_{k,j})$ ligger i planet som spänns upp av punkterna $(h(p_{k,u}), h(p_{k,v}), h(p_{k,w})$ om och endast om

$$h(p_{k,j}) = h(p_{k,v}) + r_{k,j}(h(p_{k,u}) - h(p_{k,v}) + s_{k,j}(h(p_{k,w} - h(p_{k,v})) \tag{11}$$

Vi flyttar över $h(p_{k,j})$, skriver ut $h(p) = p.z + h_1(p) - h_2(p)$, gör några omskrivningar och introducerar variablerna $\alpha_{k,j}$ för att erhålla:

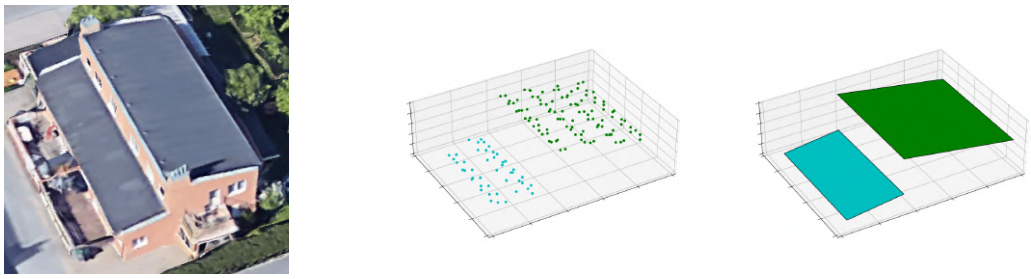
$$\begin{aligned} \alpha_{k,j} := -h_1(p_{k,j}) + h_2(p_{k,j}) + (1 - r_{k,j} - s_{k,j})(h_1(p_{k,v}) - h_2(p_{k,v})) + \\ + r_{k,j}(h_1(p_{k,u}) - h_2(p_{k,u})) + s_{k,j}(h_1(p_{k,w}) - h_2(p_{k,w}) + c_{k,j} \end{aligned} \tag{12}$$

Där $c_{k,j} := p_{k,j} + (1 - r_{k,j} - s_{k,j})p_{k,v}.z + r_{k,j}p_{k,u}.z + s_{k,j}p_{k,w}.z$ är en konstant. Eftersom vi endast kräver approximativ platthet låter vi $\delta_k \leq \alpha_{k,j} \leq \delta_k$. Vi vill att denna begränsningen ska gälla för alla punkter $p_{k,j} \in P_k$ förutom $(p_{k,u}, p_{k,v}, p_{k,w})$ och definierar $M_k := \{p_{k,1} \dots p_{k,m_k}\} \backslash \{u, v, w\}$.

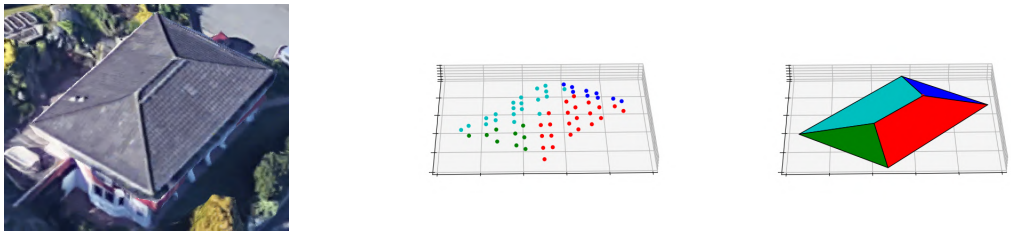
Sammanlagt kan optimeringsproblemet formuleras:

$$\begin{aligned} \text{Min} \sum_{p \in V} & h_1(p) + h_2(p) \\ \text{s.t: } & 0 \leq h_1(p), h_2(p) \leq \epsilon \\ & -\delta_k \leq \alpha_{k,j} \leq \delta_k \text{ för alla } k \in \{1 \dots n\}, j \in M_k \end{aligned} \tag{13}$$

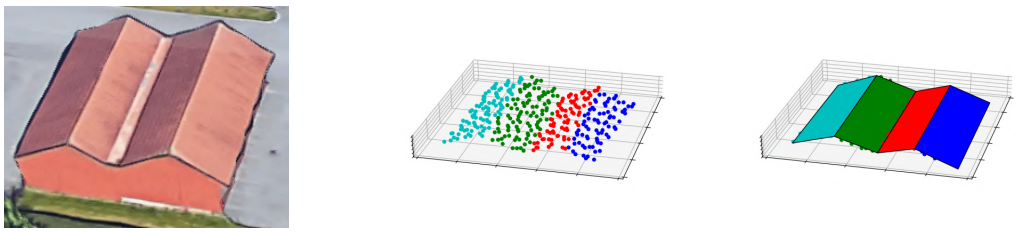
H Ett urval av genererade tak



Figur 34: Två separata platta tak. $\sigma < 0.001$, $\gamma = 0.043$



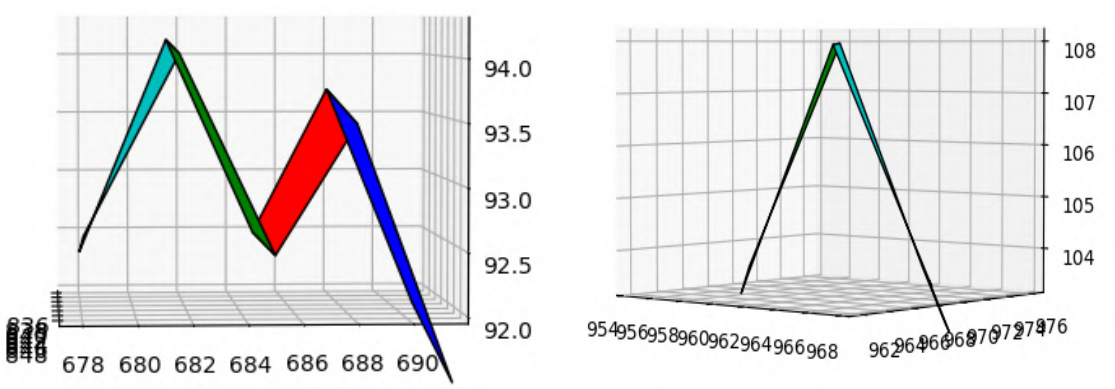
Figur 35: Valmat tak. $\sigma = 0.029$, $\gamma = 0.374$



Figur 36: Dubbla sadeltak. $\sigma = 0.111$, $\gamma = 0.518$

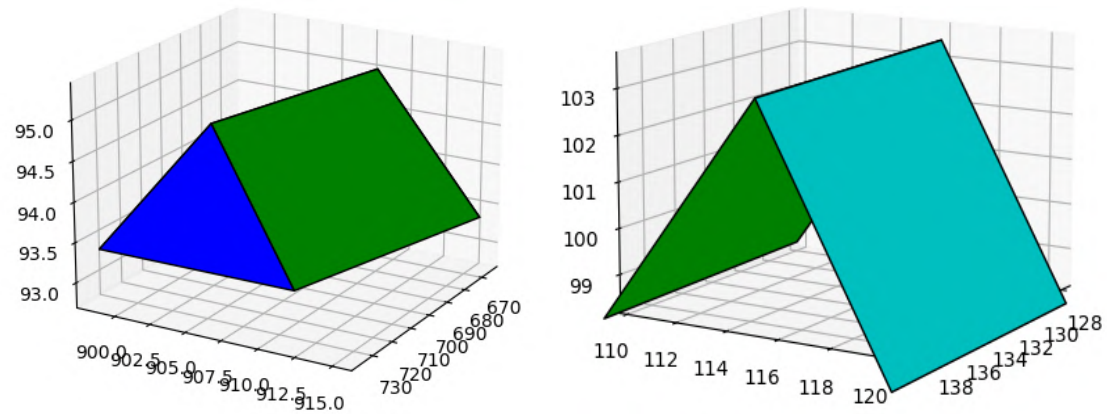
Framtagande av måttgränser

För att ta fram rimliga måttgränser studerades tak kvalitativt. Ett antal olika tak genererades och det analyserades hur väl dessa bedömdes passa punktmolnet samt hur platta de var. Efter det noterades värdena för σ och γ vartefter rimliga gränsvärden bestämdes. För exempel på ett tak som ej bedömdes ha modellerats korrekt se figur 37, notera hur takplanen inte är helt platta. För exempel på tak som ansågs korrekta se figur 38.



(a) $\sigma = 0.08$, $\gamma = 0.518$ (b) $\sigma = 0.069$, $\gamma = 0.178$

Figur 37: Byggnader som ej anses korrekt modellerade



(a) $\sigma = 0.007$, $\gamma = 0.216$ (b) $\sigma < 0.001$, $\gamma = 0.322$

Figur 38: Byggnader som anses korrekt modellerade

I Kod

All kod till projektet finns öppen på tre olika *git repositories*:

- Koden för maskininlärningen som är skriven av DTCC [1] finns på *GitLab*: `https://gitlab.com/dtcc3d/core/-/tree/pub/Rooftops/roof-edge-segmentation`
- Koden för bildbehandlingen som är skriven av Ivan Flensburg finns öppet i filen *utils.py* på *GitLab*: `https://gitlab.com/dtcc3d/core/-/tree/bsc/MLgroup/roof-edge-reconstructor/RoofEdgeReconstructor`
- Koden för generering av polygoner i 2D som är skriven av Holger Johansson finns öppet på *GitLab*: `https://gitlab.com/dtcc3d/core/-/tree/bsc/MLgroup/roof-edge-reconstructor/RoofEdgeReconstructor`
- Koden för generering av takmodeller i 3D som är skriven av Alex Matsson, Mats Richardson och Alexander Samuelsson finns öppet på *GitLab*: `https://gitlab.com/matssonalex/lod2-geo`