Računanje visina najnižeg dijela zgrade iz podataka zračnog laserskog skeniranja učinkovitim traženjem relevantnih točaka terena

Jurešić, Ana

Sveučilište
Sjever

Završni rad br. 008/GIG/2024

Računanje visina najnižeg dijela zgrade iz podataka zračnog laserskog skeniranja učinkovitim traženjem relevantnih točaka terena

Ana Jurešić, 0336054639

Varaždin, rujan 2024. godine

Sveučilište Sjever

Odjel za Geodeziju i geomatiku

Završni rad br. 008/GIG/2024

Računanje visina najnižeg dijela zgrade iz podataka zračnog laserskog skeniranja učinkovitim traženjem relevantnih točaka terena

Student

Ana Jurešić, 0336054639

Mentor

Hrvoje Matijević, doc. dr. sc.

Varaždin, rujan 2024. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Geodezija i geomatika

STUDIJ Sveučilišni prijediplomski studij geodezije i geomatike

PRISTUPNIK Ana Jurešić | MATIČNI BROJ 0336054639

DATUM 24.09.2024. | KOLEGIJ Geoinformacijski sustavi

NASLOV RADA Računanje visina najnižeg dijela zgrade iz podataka zračnog laserskog skeniranja učinkovitim traženjem relevantnih točaka terena

NASLOV RADA NA ENGL. JEZIKU Calculating the height of lowest part of a building using the efficiently determined relevant terrain points from aerially laser scanned data

MENTOR Hrvoje Matijević | ZVANJE Doc. dr. sc., znanstveni suradnik

ČLANOVI POVJERENSTVA

1. Doc. dr. sc. Nikola Kranjčić, predsjednik povjerenstva
2. Doc. dr. sc. Hrvoje Matijević, mentor
3. Prof. dr. sc. Vlado Cetl, član
4. Izv. prof. dr. sc. Danko Markovinović, zamjenski član
5.

Zadatak završnog rada

BROJ 008/GIG/2024

OPIS

Cilj završnog rada je iz oblaka točaka dobivenog zračnim laserskim skeniranjem izračunati visine najnižeg dijela zgrade. Za provedbu zadatka će se koristiti Python programski jezik, a tražene visine će se izračunati kao aritmetička sredina točaka terena u neposrednoj okolini zgrade. Koristiti će se postojeći tlocrti zgrada od kojih će se učinkovito pretražiti oblak točaka u svrhu pronalaženja točaka terena relevantnih za izračun visine najnižeg dijela zgrade. Za učinkovito pretraživanje relevantnih točaka terena koristiti će se neka Python implementacija algoritma Fast Library for Approximate Nearest Neighbor - FLANN. Vizualno će se provjeriti ispravnost rada implementacije.

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER

ZADATAK URUČEN 15.04.2024. | POTPIS MENTORA

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU

Završni/diplomski/specijalistički rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Ana Jurešić pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog rada pod naslovom Računanje visina najnižeg dijela zgrade iz podataka zračnog laserskog skeniranja učinkovitim traženjem relevantnih točaka terena te da unavedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Ana Jurešić

(vlastoručni potpis)

Sukladno članku 58., 59. i 61. Zakona o visokom obrazovanju i znanstvenoj djelatnosti završne/diplomske/specijalističke radove sveučilišta su dužna objaviti u roku od 30 dana od dana obrane na nacionalnom repozitoriju odnosno repozitoriju visokog učilišta.

Sukladno članku 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje umjetnička djelatnost i visoko obrazovanje.

Sažetak

U ovom završnom radu detaljno je istražena i razrađena metoda računanja visine najnižeg dijela zgrade programskim jezikom Python uz korištenje oblaka točaka i tlocrta zgrada. U radu su objašnjeni pojmovi KD stablo i FLANN koji omogućuju brzo pronalaženje najbližih susjeda, što značajno ubrzava cijeli proces računanja visine te su temelj programskog rješenja. Također su objašnjeni pojmovi vezani uz oblake točaka, njihovo prikupljanje, obradu i formate pohranjivanja kao i programski jezik Python te biblioteke; Geopandas, Open3D i Numpy koje su korištene u praktičnom dijelu. U zaključku ovaj rad ističe praktičnost određivanja visine najnižeg dijela zgrada ovom metodom te ukazuje na mogućnost usavršavanja metode kao i njezinu primjenu u raznim područjima.

Ključni pojmovi: LiDAR, oblaci točaka, tlocrti zgrada, Python, GeoJSON, PLY

Abstract

This thesis thoroughly investigates and develops a method for calculating the height of the lowest part of buildings using the Python programming language, along with point clouds and building floor plans. The thesis explains the concepts of KD-tree and FLANN, which enable fast nearest neighbor searches, significantly speeding up the entire height calculation process and forming the foundation of the software solution. Additionally, the thesis covers concepts related to point clouds, their collection, processing, and storage formats, as well as the Python programming language and libraries used in the practical part: geopandas, open3D, and numpy. In conclusion, this thesis highlights the practicality of determining the height of the lowest part of buildings using this method and points out the potential for further improvement and its application in various fields.

Keywords: LiDAR, point clouds, building floor plans, Python, GeoJSON, PLY

Popis korištenih kratica

LiDAR Light Detection and Ranging

AHN The national height model of the Netherlands

BAG The Building and Addres register of The Nethaerlands

GPS Global Positioning System

IMU Inertial Measurement Unit

LAS LiDAR Aerial Survey

LAZ LASzip

PLY Polygon File Format

API Application Programming Interface

FLANN Fast Library for Approximate Nearest Neighbors

K-D tree K-Dimensional Tree

Sadržaj

Uvod

U suvremenom urbanom planiranju, građevinskoj industriji i kartografiji, određivanje visina i terenskih značajki predstavlja ključan izazov. Visinski podaci o zgradama i terenu igraju ključnu ulogu u različitim aplikacijama, uključujući analize sigurnosti, planiranje infrastrukturnih projekata i procjene utjecaja na okoliš. Lasersko skeniranje predstavlja naprednu tehnologiju koja omogućava prikupljanje visoko preciznih prostornih podataka u kratkom vremenu, ali učinkovita analiza i interpretacija tih podataka zahtijeva sofisticirane metode obrade.

Ovaj rad istražuje metodu za izračunavanje visine najnižeg dijela zgrade koristeći podatke dobivene laserskim skeniranjem i podatke o tlocrtima zgrada. Kroz primjenu algoritama za traženje najbližih susjeda i analizu susjednih točaka terena, rad se fokusira na identifikaciju i precizno određivanje visinskih karakteristika zgrada. Ova metoda oslanja se na efikasno pretraživanje i obradu velikih skupova podataka, čime se omogućava generiranje modela terena i visinskih mjerenja.

Napredak u računalnim tehnologijama i algoritmima za obradu podataka omogućuje obradu velikih količina informacija u relativno kratkom vremenu. U kontekstu laserskog skeniranja, ovo istraživanje koristi tehnike za pretraživanje susjednih točaka kako bi se identificirali ključni dijelovi terena relevantni za određivanje visine zgrade. Ova metodologija ne samo da poboljšava točnost visinskih podataka, već i doprinosi učinkovitosti analize.

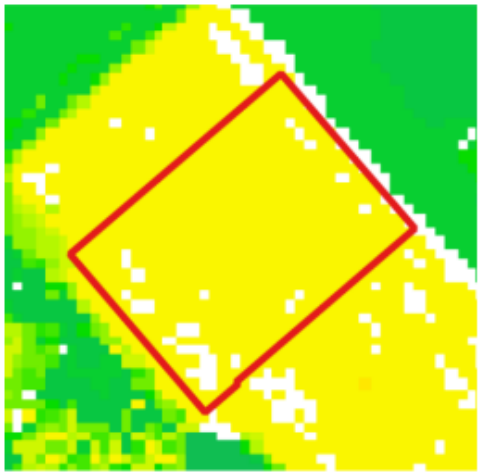
Razumijevanje kako učinkovito obraditi i analizirati visinske podatke iz laserskog skeniranja omogućuje bolju interpretaciju terenskih karakteristika i može imati značajne primjene u različitim disciplinama. Cilj ovog rada je istražiti tehniku za precizno izračunavanje visina zgrada koristeći moderne metode obrade podataka, procijeniti njihovu učinkovitost i točnost te raspraviti mogućnosti za buduća unapređenja u ovoj oblasti.

Ovaj rad bi mogao doprinijeti boljem razumijevanju kako napredne metode obrade podataka mogu poboljšati analizu visinskih informacija i pružiti korisne smjernice za primjenu ovih tehnika u stvarnim scenarijima.

1. Izračun visina zgrada

Izrada 3D modela gradova postaje sve popularnija. Popularnost modeliranja gradova može se pripisati rastućoj dostupnosti i lakoći pristupa otvorenim geoprostornim podacima. Svoju primjenu nalaze u analizi urbanog prostora, geoprostornim analizama, urbanističkom planiranju te raznim analizama rizika i sigurnosti. Osim vizualnog efekta, vrlo bitan dio 3D modela jest podatak o njegovoj visini koji omogućuje bolje razumijevanje urbanog prostora. Razvijeno je nekoliko metodologija određivanja visina zgrada koje se baziraju na korištenju LiDAR podataka te na podacima o tlocrtima, odnosno obrisima zgrada.

Jednu od metoda razvili su González-Aguilera et al. koristeći uglavnom LiDAR podatke. Metoda se zasnivala na segmentiranju krovova na nekoliko ravnina. Krovovi su segmentirani metodom (Galvanin i Dal Poz; Sampath) kako bi se generirali obrisi zgrada iz DMR-a, premda je Orford dokazao kako takav pristup daje rezultate manje točnosti [1]. Metodu određivanja visina zgrada temeljenu na kombinaciji AHN i BAG bazama podataka razvili su Dukai, B., Ledoux, H., i Stoter, J. E. [2]. Za svaku zgradu računaju se percentilne visine koristeći 3dfier softver, temeljem visinskih točaka unutar poligona zgrade. Za definiranje minimalne visine tla koristi se zona od 0,5 metara oko zgrade uzimajući u obzir filtrirane točke. Na slici (Slika 1) prikazan je model zgrade (crveno) prekriven rasteriziran AHN3 (veličina ćelije 0,5 m, gradijent boja prema visini, gdje je zelena niska, žuta je visoka elevacija, a nedostajući podaci su bijeli). U slučaju ovog određenog modela, žute ćelije su klasificirane kao "zgrada", a zelene ćelije kao "tlo" u AHN3. Referentne visine zgrada izračunavaju se koristeći podatke iz laserskog skeniranja terena i geometrijske podatke zgrada. Proces započinje identificiranjem točaka visine iz oblaka točaka koje odgovaraju određenoj zgradi, koje su ključne za određivanje stvarne visine zgrade. Zatim se izračunavaju različiti percentili visina kako bi se dobile različite referentne vrijednosti za visinu prizemlja i krova. Za prizemne visine izračunava se šest različitih percentila (0, 10, 20, 30, 40, 50). Percentili omogućuju različite načine definiranja visine tla, ovisno o specifičnim potrebama, kao što je najniža ili medijalna visina prizemlja. Slično tome, za visinu krova izračunava se šest percentila (25, 50, 75, 90, 95, 99), pružajući fleksibilnost u izboru referentne visine krova koja najbolje odgovara namjeni, bilo da je riječ o prosječnoj visini ili gotovo najvišoj točki krova. Zatim se izračunate visine dodaju kao atributi 2D BAG geometrijama omogućujući korisnicima ekstruziju 2D zgrada u 3D modele. Proces se provodi za područje cijele Nizozemske na način da se podaci dijele na 1377 pločica te se svaki mjesec provodi ažuriranje kako bi se uključile nove zgrade i ažurirali podaci.



Slika 1 Model zgrade i podaci AHN3

Metodu razvijenu od strane Jean-Michel Guldmann-a također koristi LiDAR oblake točaka te tlocrte zgrada za računanje njihovih visina. Metoda obuhvaća korake: preklapanje GIS sloja tlocrta zgrada s oblakom točaka radi filtriranja točaka koje nisu dio zgrada, osiguravanje da se tlocrti ne preklapaju kako bi svaka LiDAR točka pripala samo jednom tlocrtu, priprema uzoraka za testiranje s podacima prema stvarnom stanju te grupiranje tlocrta u različite klase kako bi se testiralo doprinosi li zaseban klasifikator poboljšanju ukupne točnosti. Razmatraju se četiri klase LiDAR točaka koje se klasificiraju kao: točke na krovu, zidovi, tlo i visoki objekti. Za karakterizaciju svake od točaka računa se jedanaest značajki podijeljenih u tri skupine: značajke specifične za točku, relativne za tlocrt zgrade te značajke susjedstva točke. Na temelju jedanaest odabranih značajki za klasifikaciju se koristi random forest metoda za koju Jean-Michel Guldmann tvrdi da pruža bolje performanse u usporedbi s drugim klasifikatorima zahvaljujući svojoj sposobnosti rangiranja važnosti značajki i otpornosti na prenaučenost, uz procjenu točnosti klasifikacije kroz četiri mjere [3].

2. LiDAR oblaci točaka

LiDAR oblaci točaka su skupovi podataka prikupljeni pomoću LiDAR (Light Detection and Ranging) tehnologije. Princip rada se temelji na emitiranju laserske zrake i mjerenju vremena njenog puta nakon refleksije od površine objekta. Svaka prikupljena točka predstavlja reflektiranu lasersku zraku od određenog objekta s koordinatama (X,Y i Z) i dodatnim informacijama poput intenziteta. Jedna od ključnih prednosti LiDAR oblaka točaka je njihova sposobnost da prodru kroz vegetaciju. To znači da se tlo ispod šuma i guste vegetacije može precizno mapirati, što je od velikog značaja za šumarstvo, poljoprivredu i zaštitu okoliša.

Postoje terestički, zračni te mobilni laserski skeneri. Laserski terestički skener postavlja se na tronožac i najveće je točnosti. Zračni laserski skener je uređaj koji se instalira na bespilotnoj letjelici i zrakoplovu te je relativno niske točnosti zbog njegove pokretljivosti dok mobilni odnosno ručni skeneri su manjih dimenzija i težine te se postavljaju na različite platforme zbog lakog transporta. Laserski skeneri se mogu podijeliti i prema načinu snimanja, načinu mjerenja udaljenosti i načinu prikupljanja oblaka točaka. Prema načinu snimanja, postoje skeneri kamere koji sadrže ograničeni prozor snimanja, panoramski skeneri koji su limitirani stajalištem instrumenta te hibridni skeneri koji imaju prozor snimanja po horizontalnoj osi 360° i vertikalnoj 60°. Prema načinu mjerenja udaljenosti, razlikujemo pulsne koji rade na principu mjerenja vremena između odaslanog i primljenog signala te se najviše koriste u terestičkoj izmjeri, fazni koji radi na principu razlike u fazi između odaslanog i primljenog signala, triangulacijski koji radi na principu optičke triangulacije međutim zbog svog malog dometa nije primjenjiv u geodeziji. Prema načinu prikupljanja oblaka točaka odnosno, podataka dijele se na one apsolutne tj. georeferencirane i relativne odnosno lokalne. Za potrebe prikupljanja podataka zgrada zbog velikog područja i složenosti terena upotrebljavaju se zračni laserski skeneri instalirani na bespilotne letjelice i zrakoplove.

LiDAR objedinjuje laserski skener, prijemnik signala te inercijalnu mjernu jedinicu. Laser uključuje diodu koja emitira svjetlost na specifičnoj frekvenciji. Sustav je sposoban zabilježiti vremensku razliku između emisije laserskog impulsa i prijema povratnog signala (vrijeme leta). Poznavanje brzine prijenosa i vremena leta omogućuje izračunavanje udaljenosti između laserskog odašiljača i reflektirajuće površine (terena). GPS i IMU uređaji se koriste za određivanje 3D pozicije i orijentacije laserskog skenera u svakom mjernom trenutku [4].

2.1. Alati za obradu

Alati za obradu 3D oblaka točaka omogućuju korisnicima upravljanje i analiziranje prikupljenih podataka. Oni uključuju funkcionalnosti poput filtriranja i uklanjanja šuma, postavljanja u prostor (registracije) različitih oblaka točaka, stvaranja trokutastih mreža (meshing), segmentacije i klasifikacije točaka, kao i izračunavanja raznih geometrijskih i statističkih parametara. Osim toga, alati često omogućuju vizualizaciju 3D podataka, što je ključno za njihovu interpretaciju i prezentaciju. Zahvaljujući napretku u računalnoj tehnologiji, alati postaju sve pristupačniji i moćniji, omogućujući obradu velikih količina podataka na standardnim računalima. Time je omogućen širok spektar primjena, od preciznog mjerenja i analize do kreativnih i vizualnih projekata.

2.1.1. CloudCompare

CloudCompare je softver za obradu 3D oblaka točaka i trokutastih mreža. Izvorno je dizajniran za usporedbu između dva gusta 3D oblaka točaka ili između oblaka točaka i trokutaste mreže, oslanjajući se na specifičnu oktalnu strukturu namijenjenu ovoj zadaći. Kasnije je proširen kako bi postao općenitiji softver za obradu oblaka točaka, uključujući mnoge napredne algoritme, kao što su registracija, presnimavanje, upravljanje bojama, normalama, poljima skalarnih vrijednosti, izračun statistike, upravljanje senzorima, interaktivna ili automatska segmentacija i poboljšanje prikaza.

Korisničko sučelje CloudComparea temelji se na Qt i OpenGL tehnologijama, pružajući intuitivno okruženje za rad. Softver omogućuje izračunavanje udaljenosti između oblaka i mreže te podržava širok spektar formata datoteka, uključujući otvorene formate oblaka točaka (ASCII, LAS, E57), formate proizvođača (DP, Riegl, FARO) te formate trokutastih mreža (OBJ, PLY, STL, FBX). Također podržava nekoliko formata polilinija ili poligona (SHP, DXF) i određene SfM formate (Bundler, Photoscan PSZ).

CloudCompare podržava mehanizam proširenja putem dodataka. Postoje dvije vrste dodataka: standardni dodaci (algoritmi) i OpenGL filtri (napredni shaderi). Ovi dodaci omogućuju dodatne funkcionalnosti poput ambijentalne okluzije ili primjene real-time filtera na sirove oblake točaka. Sve ove karakteristike čine CloudCompare moćnim alatom za 3D obradu i analizu, osobito u kontekstu usporedbe oblaka točaka i mreža, što je korisno u raznim industrijama i istraživačkim područjima [5].

2.2. Formati zapisa podataka

Formati 3D datoteka igraju ključnu ulogu u upravljanju, obradi i razmjeni trodimenzionalnih podataka. Oni omogućuju učinkovito pohranjivanje informacija o geometriji objekata, njihovim svojstvima, kao i dodatnim atributima poput boje i tekstura. Različiti formati prilagođeni su specifičnim potrebama i industrijama, od geodezije, arhitekture do 3D modeliranja.

2.2.1. LAZ

LAZ format je format kompresije koji ne gubi podatke, razvijen za učinkovitu pohranu i distribuciju LiDAR podataka, koji su izvorno pohranjeni u LAS formatu. Dizajniran je s ciljem značajnog smanjenja veličine datoteka, zadržavajući pritom sve originalne informacije bez ikakvog gubitka preciznosti ili točnosti. U prosjeku, LAZ format smanjuje veličinu datoteka na samo 7 do 25 posto od njihove izvorne veličine, omogućujući tako jednostavniju pohranu i prijenos velikih količina LiDAR podataka.

Jedna od ključnih karakteristika LAZ formata je njegova sposobnost za strujanje podataka i slučajni pristup, što znači da korisnici mogu dekomprimirati i pristupati podacima u dijelovima, bez potrebe za dekompresijom cijele datoteke. To omogućuje brže pretraživanje i obradu podataka, čineći LAZ format idealnim za aplikacije koje zahtijevaju brzo rukovanje velikim količinama podataka.

LAZ također koristi učinkovite algoritme za kompresiju koji su specifični za strukturu LiDAR podataka. Ovi algoritmi uzimaju u obzir karakteristike poput uniformne distribucije točaka i varijabilne preciznosti podataka, čime se postiže bolja kompresija u odnosu na generičke kompresijske algoritme poput ZIP-a ili RAR-a. Kompresijski proces podržava sve verzije LAS formata do 1.3, uključujući različite tipove točaka i pripadajuće atribute poput koordinata, intenziteta, boje i GPS vremena. LAZ format je široko prihvaćen u industriji te se koristi od strane različitih softverskih organizacija i alata [6].

2.2.2. PLY

PLY (Polygon File Format), poznat i kao Stanford Triangle Format, koristi se za pohranu 3D objekata kao skup poligona, a ima dvije verzije: ASCII i binarnu. ASCII format omogućuje jednostavno razumijevanje i ručno uređivanje zbog svoje čitljivosti, dok je binarni format optimiziran za kompaktno pohranjivanje i brži pristup podacima.

Osnovna struktura PLY datoteke uključuje zaglavlje koje opisuje tipove elemenata, te njihove atribute. U binarnom formatu, podaci su pohranjeni u učinkovitijem binarnom obliku, što rezultira manjim veličinama datoteka i bržim operacijama čitanja i pisanja. PLY format omogućuje dodavanje novih elemenata i atributa, kao što su boje ili materijali, bez narušavanja kompatibilnosti s postojećim alatima. Ova fleksibilnost i jednostavnost čine PLY format korisnim za razmjenu 3D modela među različitim softverskim alatima i aplikacijama [7].

2.2.3. GeoJSON

GeoJSON je format za kodiranje različitih geografskih podataka koristeći JavaScript Object Notation (JSON). Ovaj format omogućuje predstavljanje različitih vrsta geografskih objekata, uključujući Geometrije, kao što su točke, linije i poligoni, kao i entitete s prostornim granicama, nazvane Feature, ili skupove tih entiteta, označene kao FeatureCollection. GeoJSON podržava sedam tipova geometrijskih objekata: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon i GeometryCollection.

Glavna svrha GeoJSON-a je da pruži jednostavan i standardiziran način za rad s geografskim podacima u web aplikacijama, koristeći poznati JSON format. Koncepti GeoJSON-a temelje se na prethodnim otvorenim standardima za geografske informacije, no sučelje je pojednostavljeno kako bi bolje odgovaralo web aplikacijama i JSON baziranim bazama podataka. GeoJSON format je stekao značajnu popularnost od svog prvog izdanja 2008. godine i danas se široko koristi u JavaScript web-mapping bibliotekama, bazama podataka temeljenim na JSON-u, te web API-jevima [8].

3. Python

Python je programski jezik poznat po svojoj jednostavnosti i čitljivosti. Razvio ga je Guido van Rossum, a prvi put je objavljen 1991. godine. Dizajniran je tako da bude jednostavan za učenje i korištenje, omogućujući programerima da izraze koncepte u manje redaka koda u odnosu na jezike poput C++ ili Jave. Portabilnost je omogućena jer se Python program može pokrenuti neovisno o operativnim sustavima. Kao kreator Pythona, Guido van Rossum istaknuo je svoje razloge za stvaranje ovog jezika i podijelio osobnu filozofiju. Ideja za Python nastala je tijekom božićnih praznika 1989. godine kada je tražio "hobi" projekt. Njegova namjera bila je stvoriti interpreter za novi skriptni jezik koji bi privukao Unix/C programere, inspiriran ABC-om, jezikom koji je također pomogao stvoriti. ABC je bio elegantan i moćan, ali nije uspio u Unix/C svijetu zbog svoje zatvorenosti i ograničenih mogućnosti proširenja. Python je trebao izbjeći te greške i nuditi fleksibilnost i proširivost. Python se brzo etablirao kao moćan alat u mnogim područjima, osobito u znanstvenim istraživanjima, analizi podataka i geografskim informacijskim sustavima (GIS). Njegova sposobnost za obradu velikih količina podataka dodatno je pojačana bibliotekama poput NumPy, Pandas i GeoPandas. NumPy omogućuje rad s velikim multidimenzionalnim nizovima i matricama, pandas olakšava manipulaciju tabelarnim podacima, dok GeoPandas omogućuje rad s prostornim podacima. Ove biblioteke čine Python izuzetno moćnim alatom za analizu podataka. Pythonov interpretirani način rada omogućuje brzo testiranje i iteraciju koda, što značajno ubrzava razvojni proces i smanjuje broj grešaka [9]. Programeri mogu testirati dijelove koda u naredbenoj liniji prije nego što ih integriraju u cjeloviti program, čime se povećava produktivnost i efikasnost. Pythonova virtualna mašina prevodi Python kod u poseban binarni kod, omogućujući brže programiranje i lako otkrivanje pogrešaka. Jedna od najkontroverznijih značajki Pythona je upotreba uvlaka za grupiranje naredbi, što doprinosi čitljivosti koda smanjujući vizualni nered i omogućujući ujednačen stil pisanja koda. Ovo naslijeđe iz ABC-a čini Python kod čitljivijim i lakšim za održavanje, što je ključna filozofija jezika. Python nije najbrži skriptni jezik, ali je vrlo konkurentan zahvaljujući stalnom povećanju brzine hardvera i učinkovitosti uštede vremena programera tijekom pisanja i debugiranja koda. Većina programera smatra Python pobjednikom u smislu vremena kodiranja, a mnogi uživaju u radu s njim, što je najbolja preporuka koju jezik može dobiti. Guido van Rossum priznaje da je uspjeh Pythona rezultat zajednice korisnika koji su od samog početka doprinosili jeziku kroz povratne informacije, prijedloge značajki, kodne doprinose i osobne uvide. Zajednica je ključna za razvoj i popularnost Pythona, čineći ga jednim od najvažnijih programskih jezika današnjice [10].

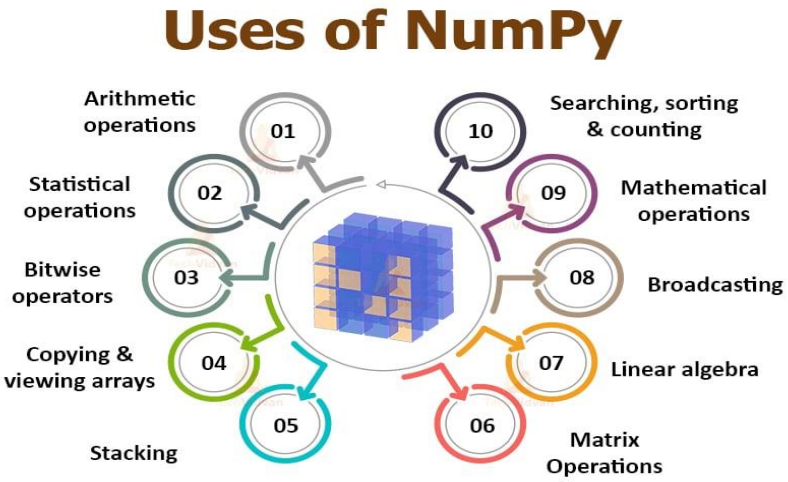
3.1. Python biblioteke

Python biblioteke su kolekcije unaprijed napisanog koda koje korisnici mogu uvesti u svoje projekte kako bi brzo i jednostavno dodali dodatne funkcionalnosti bez potrebe za pisanjem vlastitog koda od nule. Ove biblioteke pokrivaju širok spektar namjena, od osnovnih matematičkih operacija do složenih mrežnih protokola i grafičkih sučelja.

Jedna od ključnih prednosti Python jezika je njegova bogata standardna biblioteka, koja uključuje module za rad s tekstualnim formatima, bazama podataka, mrežnim protokolima, testiranjem i mnogo više. Osim standardne biblioteke, Python zajednica je razvila i veliki broj vanjskih (treće strane) biblioteka koje su dostupne putem Python Package Indexa (PyPI). Ove biblioteke omogućuju korisnicima da lako implementiraju specijalizirane funkcionalnosti, poput strojnog učenja, obrade podataka, vizualizacije i mnogih drugih područja [10].

3.1.1. Numpy biblioteka

NumPy je osnovna biblioteka za znanstveno računalstvo u Pythonu. Ona pruža višedimenzionalne nizove i različite izvedene objekte, poput maskiranih nizova i matrica, kao i širok spektar funkcija za brze operacije nad tim nizovima. Ove funkcije obuhvaćaju matematičke operacije, logičke operacije, manipulaciju oblikom, sortiranje, odabir, I/O operacije, diskretne Fourierove transformacije, osnovnu linearnu algebru, osnovne statističke operacije, slučajnu simulaciju i još mnogo tog (Slika 2).



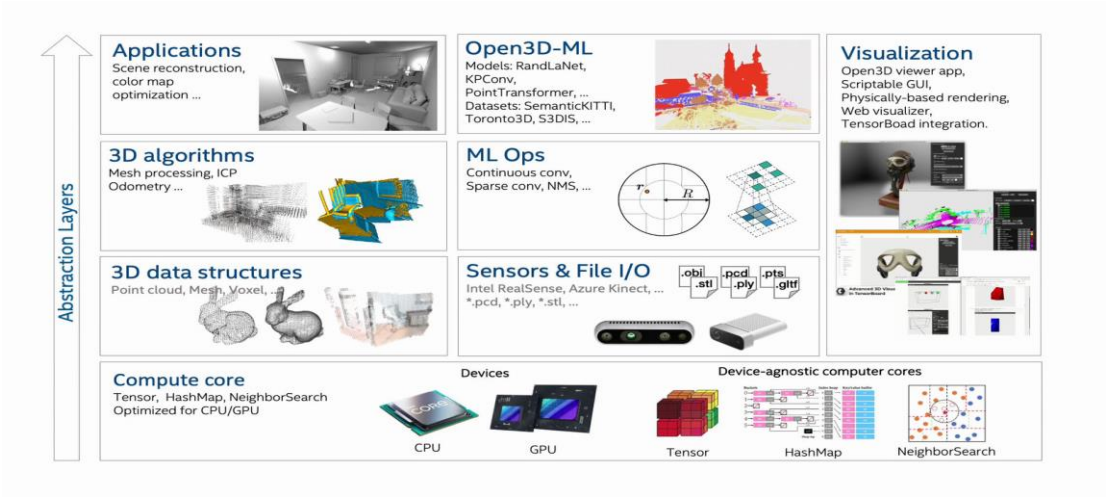
Slika 2 Spektar Numpy funkcija

Srž NumPy-a čini ndarray objekt, koji omogućuje rad s n-dimenzionalnim nizovima homogenih tipova podataka, s operacijama koje se izvode u kompajliranom kodu radi postizanja visokih performansi. NumPy nizovi imaju fiksnu veličinu pri stvaranju, za razliku od Python popisa koji mogu dinamički rasti. Svi elementi u NumPy nizu moraju biti iste vrste podataka, što omogućuje učinkovito korištenje memorije i poboljšava performanse. NumPy značajno olakšava napredne matematičke operacije nad velikim skupovima podataka. Ove operacije se često izvode učinkovitije i s manje koda nego korištenjem Python ugrađenih sekvenci. Sve veći broj znanstvenih i matematičkih paketa temeljenih na Pythonu koristi NumPy nizove. Oni obično podržavaju Python sekvencijski unos, no pretvaraju ga u NumPy nizove prije obrade, i često izlaze NumPy nizovi. To znači da za učinkovito korištenje većine modernog znanstvenog i matematičkog softvera temeljenog na Pythonu, poznavanje NumPy nizova postaje ključno. NumPy pruža značajne prednosti u performansama. Kao primjer, množenje svakog elementa u 1-D nizu s odgovarajućim elementom u drugom nizu iste duljine u Pythonu bi zahtijevalo korištenje petlji, što je sporo i neučinkovito za velike nizove. NumPy omogućava istu operaciju jednostavnom naredbom c = a * b, zahvaljujući vektorizaciji i kompajliranom C kodu u pozadini. Vektorizacija u NumPy-u znači odsutnost eksplicitnog ponavljanja i indeksiranja u kodu, jer se ti procesi odvijaju "iza kulisa" u optimiziranom, unaprijed kompajliranom C kodu. Vektorizirani kod je koncizniji, lakši za čitanje, sadrži manje grešaka i nalikuje standardnoj matematičkoj notaciji. Emitiranje, s druge strane, omogućuje implicitne operacije element po element između nizova različitih oblika, pod uvjetom da su kompatibilni, čime se povećava fleksibilnost i učinkovitost koda. NumPy također podržava objektno orijentirani pristup, gdje ndarray klasa posjeduje brojne metode i atribute. Ova fleksibilnost omogućuje programerima da kodiraju na način koji im najbolje odgovara, čineći NumPy de facto jezikom za višedimenzionalnu razmjenu podataka u Pythonu. Kao rezultat toga, mnogi znanstveni i matematički paketi poput SciPy i Pandas temelje se na NumPy nizovima, čineći poznavanje NumPy-a ključnim za učinkovito korištenje ovih alata [11].

3.1.2. Open3d biblioteka

Open3D je otvorena biblioteka koja omogućava brzi razvoj softvera za obradu 3D podataka. Kroz svoj frontend, Open3D nudi niz pažljivo odabranih struktura podataka i algoritama dostupnih u C++ i Pythonu. Backend je visoko optimiziran i podržava paralelnu obradu. Open3D je razvijen od nule s minimalnim i pažljivo odabranim setom ovisnosti, što omogućava jednostavnu instalaciju na različitim platformama i kompilaciju iz izvornog koda uz minimalan trud. Kod je

čist, konzistentno stiliziran i održavan kroz jasan mehanizam pregleda koda. Open3D je korišten u brojnim objavljenim istraživačkim projektima i aktivno se koristi u oblaku (Slika 3) [12].



Slika 3 Arhitektura rada open3D-a

3.1.3. Geopandas biblioteka

GeoPandas je ključna biblioteka u Pythonu koja značajno olakšava geoprostornu analizu zahvaljujući svojim moćnim funkcijama. Ova biblioteka omogućuje učinkovitu sintezu geoprostornih podataka i mapiranje, čineći kompleksne geoprostorne operacije pristupačnijima i intuitivnijima za korisnike.

GeoPandas omogućuje učitavanje geoprostornih podataka iz različitih formata, kao što su ShapeFile datoteke, što je prvi korak u analizi podataka o različitim geografskim područjima. Nakon učitavanja podataka, moguće je primijeniti različite vizualizacije kako bi se istaknuli specifični podaci unutar GeoDataFramea. Primjerice, korištenjem naredbe gdf.plot(column="specific_column"), korisnik može vizualizirati varijacije u skupu podataka s različitim shemama boja. Jedna od ključnih mogućnosti GeoPandas je izračunavanje gustoće stanovništva ili drugih demografskih podataka dijeljenjem ukupne populacije s površinom određenih područja. Ove informacije mogu se pohraniti u novi stupac, poput demographic_density, te se zatim koristiti za iscrtavanje koropletnih karata. Ovakve karte omogućavaju vizualizaciju podataka na intuitivan i razumljiv način, pružajući dublji uvid u demografske obrasce unutar analiziranih područja. Poboljšanje koropletnih karata moguće je uključivanjem osnovnih karti pomoću Contextilyja, što omogućava bolje razumijevanje geografskog konteksta. Pretvaranjem GeoDataFramea u Web Mercator projekciju (EPSG:3857), osigurava se točno poravnanje slojeva na karti. Contextily pruža različite osnovne karte koje se

mogu prilagoditi potrebama korisnika, omogućavajući detaljnu i preciznu vizualizaciju. GeoPandas je korišten u brojnim objavljenim istraživačkim projektima i aktivno se koristi u različitim domenama, uključujući urbanističko planiranje, ekologiju i druge znanstvene discipline. Kao što je istaknuto, "GeoPandas igra ključnu ulogu u olakšavanju geoprostorne analize putem svojih moćnih funkcija," što ovu biblioteku čini nezamjenjivim alatom za sve koji se bave geoprostornom analizom u Pythonu. Kroz korištenje GeoPandas, korisnici mogu upravljati kompleksnim geoprostornim podacima i izraditi vizualizacije koje pružaju bogatiji i detaljniji uvid u podatke omogućavajući bolje donošenje odluka temeljenih na analizi podataka [13].

4. KD stablo

KD-stablo (K-dimensionalno stablo) je struktura podataka koja se koristi za organizaciju K-dimenzionalnih podataka radi brzog pretraživanja. Glavna svrha KD-stabla je olakšati učinkovito pretraživanje podataka u K-dimenzionalnom prostoru.

Funkcije KD-stabla :

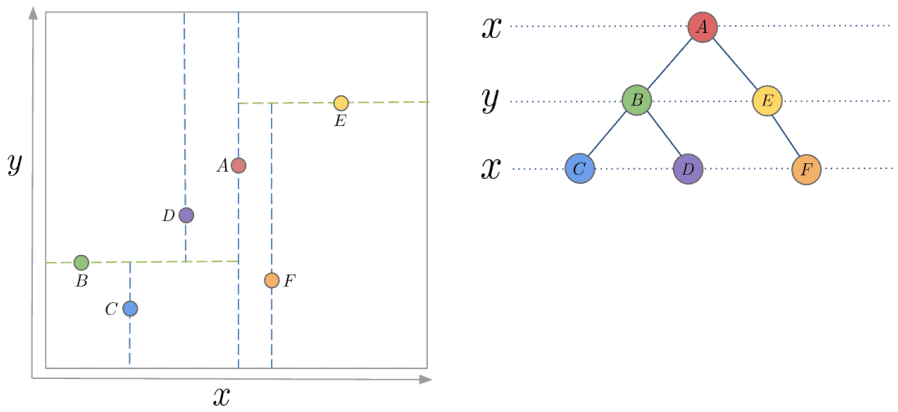
Organizacija prostora: KD-stablo dijeli prostor na poluprostorije kako bi organiziralo podatke. Svaki čvor stabla predstavlja jedan podatak ili skup podataka (Slika 4).

Rekurzivna podjela: Prostor se rekurzivno dijeli po dimenzijama (prvo po prvoj dimenziji, zatim po drugoj i tako dalje). Na primjer, ako imamo podatke u 2D prostoru (x, y), stablo će naizmjenično dijeliti prostor po x i y dimenzijama.

Brzo pretraživanje: Zahvaljujući organizaciji po dimenzijama, pretraživanje u KD-stablu može biti brzo. Možemo brzo pronaći najbliže susjede ili druge informacije u prostoru.

Primjena: KD-stablo se često koristi u algoritmima poput k-najbližih susjeda (KNN) gdje je potrebno brzo pronalaženje najbližih susjeda u K-dimenzionalnom prostoru.

Operacije: Osim pretraživanja, KD-stablo može podržavati operacije umetanja (insert), brisanja (delete) i ažuriranja podataka [14].



Slika 4 KD-stablo

5. FLANN

"FLANN [15] je biblioteka za brzo približno pretraživanje najbližih susjeda u visokodimenzionalnim prostorima. Sadrži kolekciju algoritama za koje se pokazalo da najbolje funkcioniraju za pretraživanje najbližih susjeda te sustav za automatski odabir najboljeg algoritma i optimalnih parametara ovisno o skupu podataka. FLANN je napisan u C++-u i sadrži poveznice za sljedeće jezike: C, MATLAB, Python i Ruby."

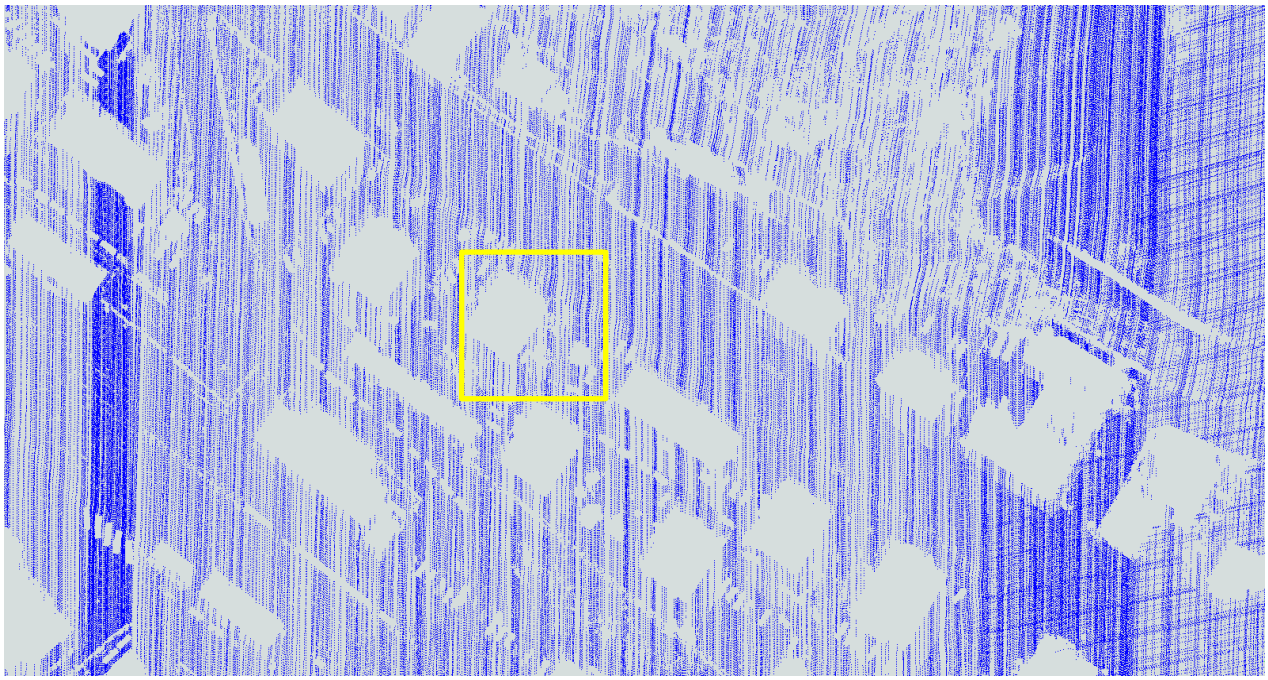
Pretraga najbližih susjeda od velike je važnosti u širokom spektru aplikacija poput prepoznavanja slika, kompresije podataka, prepoznavanja obrazaca i klasifikacije, strojnog učenja, sustava za dohvat dokumenata, statistike i analize podataka. Međutim, u visokodimenzionalnim prostorima rješavanje ovog problema predstavlja izazov jer standardne metode poput brute-force pretrage postaju ne efikasne.

FLANN pruža alternativu u obliku aproksimativne pretrage najbližih susjeda. Ova vrsta algoritama traga za približnim najbližim susjedima koji često daju dovoljno točne rezultate u praktičnim primjenama, uz znatno smanjeno vrijeme izvršavanja u usporedbi s točnim metodama.

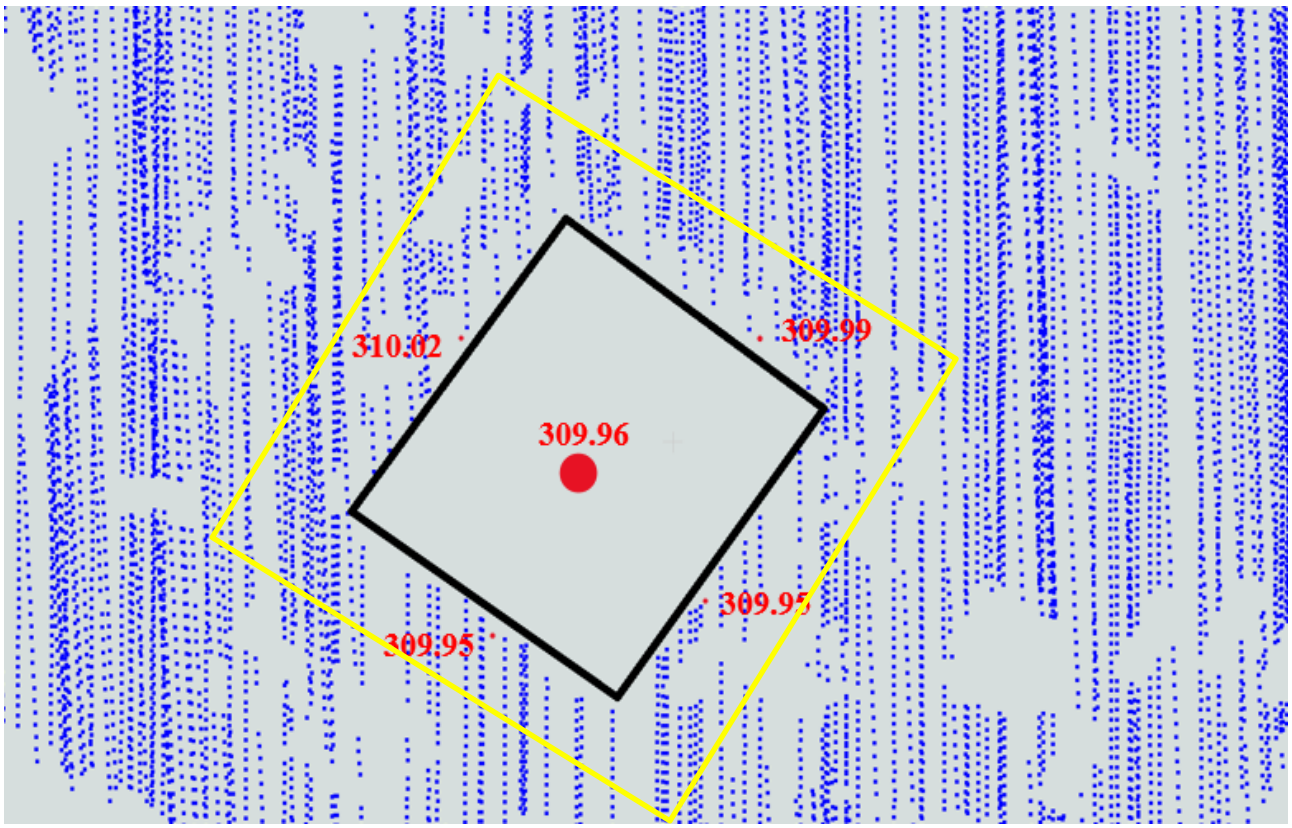
U konačnici, FLANN predstavlja moćan alat za sve koji se bave problemima pretrage najbližih susjeda u visokodimenzionalnim prostorima, omogućujući im da postignu značajno poboljšanje performansi u usporedbi s tradicionalnim metodama bez značajnog gubitka točnosti u većini praktičnih slučajeva.

6. Praktični dio rada

U praktičnom djelu završnog rada potrebno je odrediti visine najnižih dijelova zgrada kao jednog od koraka u određivanju ukupne visine zgrade. Koristeći se otvorenim podatcima tlocrtima zgrada i LiDAR oblacima točaka upotrebom programskog jezika Python. Visina najnižeg djela zgrade će se odrediti koriteći „KDTreeFlann“ iz biblioteke Open3D. To će biti provedeno korištenjem centroida poligona zgrade dobivenog iz tlocrta zgrade iz shapefile podataka na osnovu kojeg će se iz podataka oblaka točaka terena (Slika 5) pronaći 100 najbližih točaka. Srednja vrijednost Z koordinata najbližih točaka je tražena visina najnižeg dijela zgrade (Slika 6). Ta visina će biti dodijeljena centroidu, tako obrađeni podaci će biti izvezeni u GeoJSON formatu te će predstavljati najnižu visinu zgrade.



Slika 5 Prikaz oblaka točaka terena



Slika 6 Skicirani prikaz metode određivanja visina

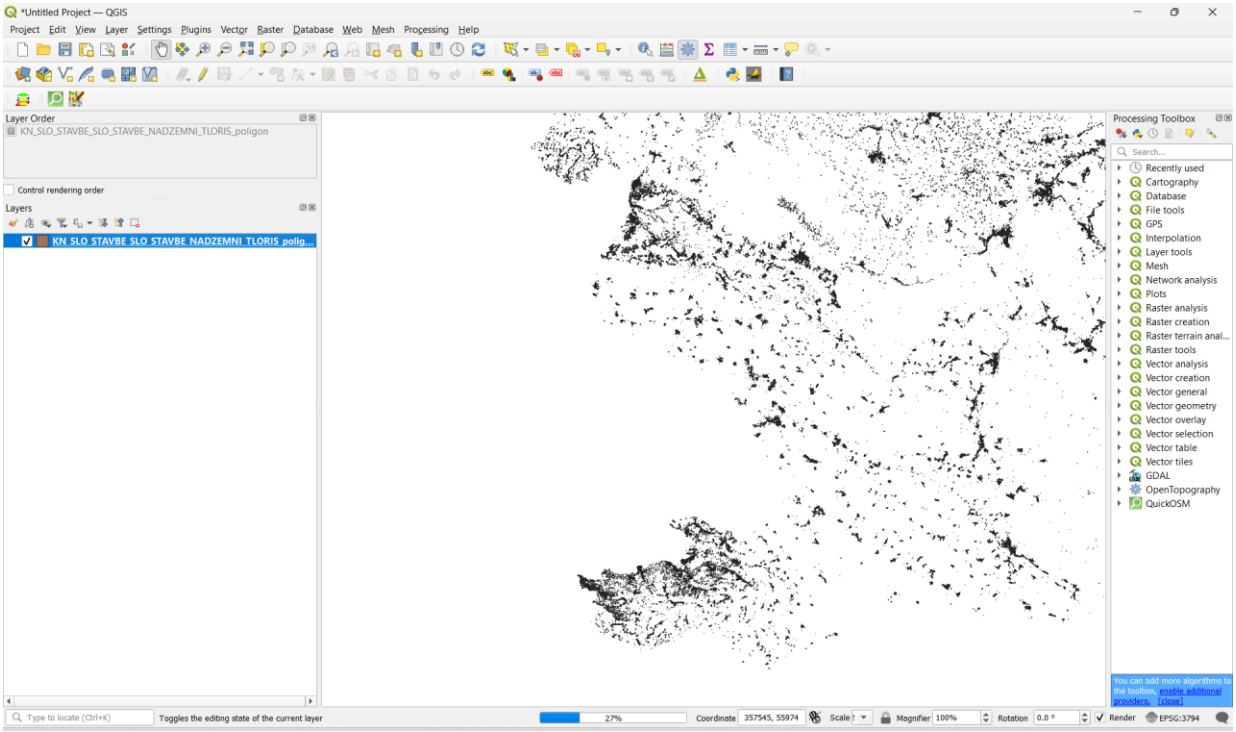
Glavni koraci praktičnog dijela rada su:

- Pripremanje LiDAR podataka pomoću CloudCompare-a i jednostavnih Python skripta
- Pripremanje podataka o tlocrtima zgrada pomoću QGIS-a i jednostavnih Python skripti
- Određivanje područja interesa koristeći složenije Python skripte
- Filtriranje zgrada unutar tog područja koristeći složenije Python skripte
- Pretvaranje oblaka točaka u Numpy koristeći složenije Python skripte
- Izgradnja k-d stabla koristeći složenije Python skripte
- Pronalaženje najbližih točaka koristeći složenije Python skripte
- Izračun srednje visine koristeći složenije Python skripte

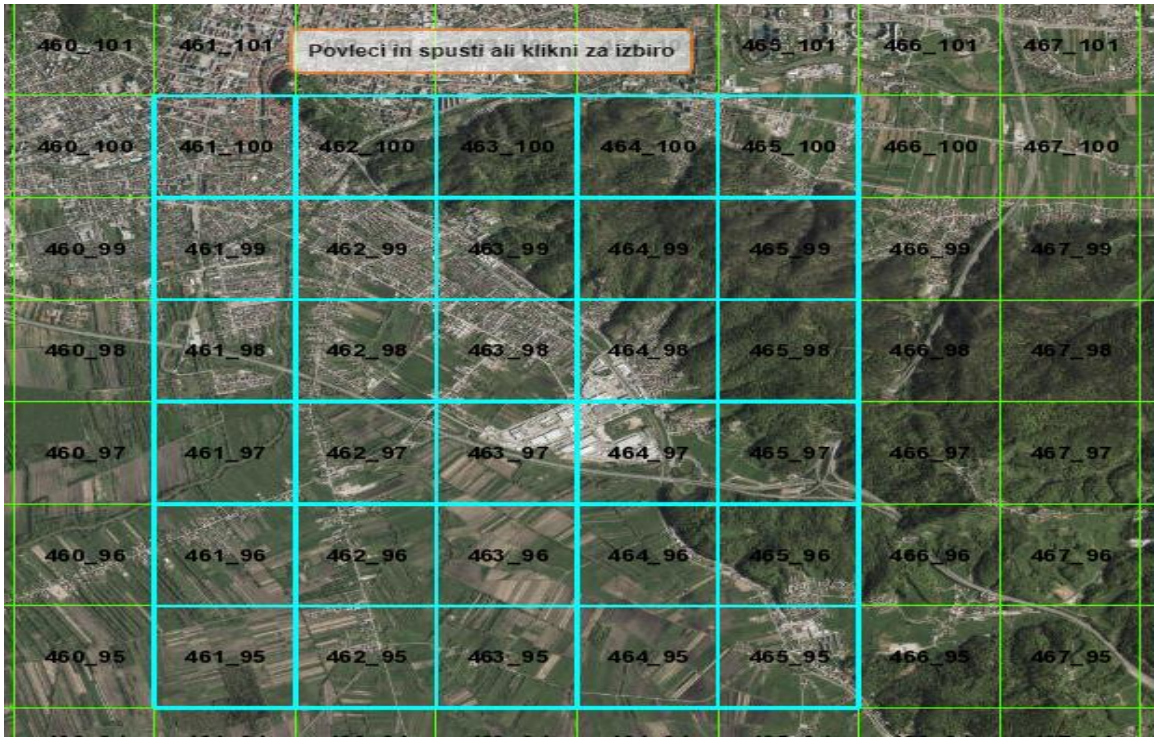
6.1. Preuzimanje podataka

Korišteni su otvoreni podaci Republike Slovenije. Podaci tlocrta zgrada preuzeti su u shapefile formatu te su dostupni putem linka https://ipi.eprostor.gov.si/jgp/data (Slika 7), dok su oblaci točaka preuzeti sa http://gis.arso.gov.si. Skup LiDAR oblaka točaka koji je korišten je GKOT (georeferencirani i klasificirani oblak točaka) koji sadrži točke klasificirane na tlo, zgrade i tri tipa

vegetacije (niska, srednja i visoka), pohranjeno u LAS formatu. Podaci su dostupni za područje u obliku pločica koje odgovaraju površini 1km² (Slika 8).



Slika 7 Sirovi podaci tlocrta zgrada



Slika 8 Prikaz područja za koji su preuzeti LiDAR podatci

6.2. Tehničke specifikacije ulaznih podataka

6.2.1. Lidar oblaci točaka

Navedene su tehničke specifikacije opreme i podataka korištenih tijekom prikupljanja Lidar oblaka točaka, uključujući karakteristike laserskog skenera, GNSS-a te preciznost i točnost podataka [16].

Upotrijebljena oprema tijekom snimanja	
Laserski skener	RIEGL LMS-Q780
INS (Inertial Navigation System)	IGI Aerocontrol Mark II.E 256Hz
GNSS (Global Navigation Satellite System)	Novatel OEMV-3
Maksimalni kut skeniranja	±30°
Pogreška mjerenja kuta skeniranja	≤ 0,25 mrad
Preciznost INS kutova	"roll/pitch" 0,004° i "heading" 0,01°, preciznost položaja 5 cm
Pogreška sinkronizacije	0,005 ms između INS, GNSS i laserskog sustava
Visina leta iznad tla	1200 do 1400 m nad tlom
Kontrola apsolutne visinske točnosti	Blok B35: Upotrijebljeno 89 kontrolnih točaka izmjerenih GNSS RTK metodom, RMS elipsoidnih visina 0,06 m
Prosječna gustoća prvih odboja	Blok B35: 97,6 % kvadrata veličine 10 m² odgovara gustoći od 5 točaka/m² (zahtijevano barem 90 %)

Tablica 1 tehinčke specifikacije LiDAR podataka

6.2.2. Tlocrti zgrada

Podaci za zgrade su prikupljeni pomoću fotogrametrijskog snimanja, terenskog prikupljanja podataka, te preuzimanjem podataka iz drugih evidencija prilikom uspostavljanja katastra zgrada. Položajna preciznost varira ovisno o metodi prikupljanja koordinata: terensko mjerenje: 0-12 cm,

fotogrametrijsko snimanje: 0-50 cm, iz temeljnih topografskih planova (TTP): 0-150 cm, iz grafičkih podataka zemljišnog katastra (ZK) preko 150 cm, terensko prikupljanje podataka bez poveznice na mrežu: preko 150 cm, mjerenje zgrada: 0-12 cm, prikupljanje podataka iz Registra prostornih jedinica (RPE): 0-150 cm.

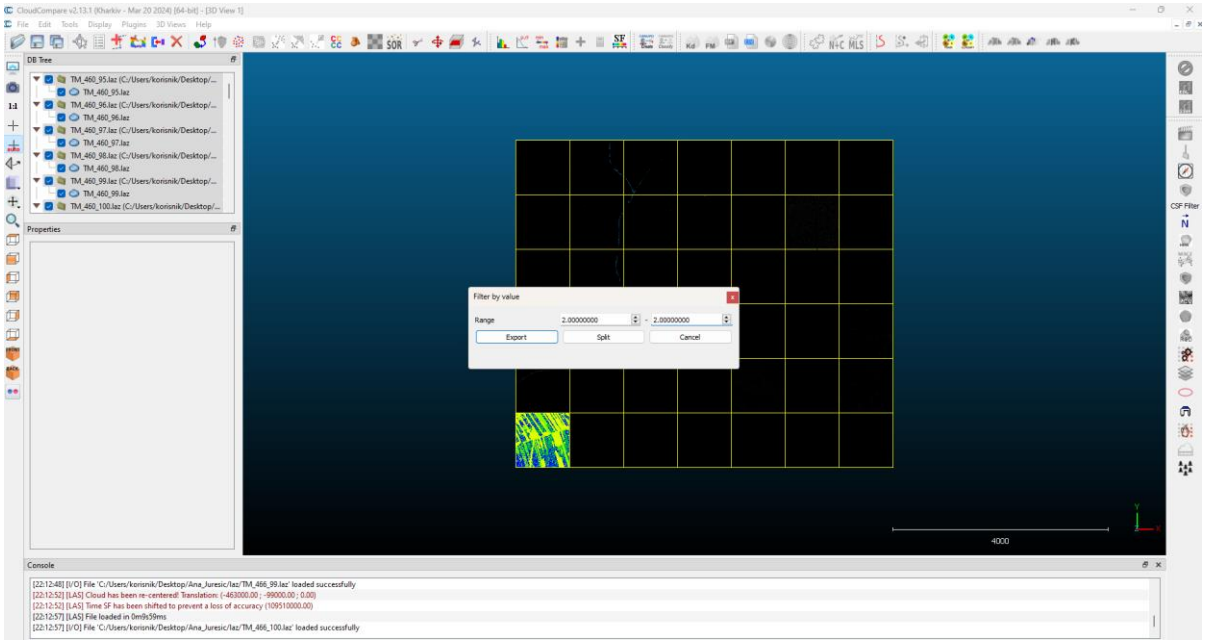
Održavanje katastra nekretnina provodi se putem različitih vrsta katastarskih postupaka, kojima se evidentiraju promjene podataka na temelju inicijative različitih subjekata (investitori, vlasnici, korisnici, upravitelji, država, općine, itd.), po službenoj dužnosti, na temelju pravomoćnih sudskih odluka i drugih akata, te na temelju promjena u povezanim bazama podataka (Registar prostornih jedinica, Registar adresa, Evidencija državne granice, Zemljišna knjiga, Centralni registar stanovništva, Poslovni registar, Evidencija stvarne uporabe zemljišta, PIS, itd.).

Katastar nekretnina je uspostavljen u skladu sa Zakonom o katastru nekretnina (ZKN). Podaci u digitalnom obliku su dostupni za cijelo područje Republike Slovenije, a baza podataka se vodi u državnom ETRS koordinatnom sustavu D96/TM [17].

6.3. Priprema podataka

6.3.1. Priprema LiDAR podataka

Podaci su preuzeti u LAZ formatu te se obrađuju u softveru CloudCompare. Oblaci točaka sadrže 3 klase tlo, zgrade i različite tipove vegetacije. Za određivanje najnižeg dijela zgrade potrebna je samo klasa tlo. Klasa tlo nalazi se pod vrijednosti 2 te su pomoću alata „Filter by value“ izdvojeni samo podaci o tlu (Slika 9).



Slika 9 Filtracija podataka tla

Koordinate svih točaka su pomaknute za 463 000 po X osi i za 99 000 po Y osi kako bi radili sa manjim brojevima i time povećali točnost krajnjeg rezultata. Potrebni podaci su nakon uređivanja spremljeni u PLY format koji je kompatibilan sa Open3d bibliotekom koja će se koristiti u daljnjim koracima u binarnom obliku kako bi podaci zauzimali manje prostora. S obzirom da softver ne nudi spremanje više datoteka u PLY format podaci su spremljeni zasebno u tom formatu te su programski spojeni kako bi daljnji rad bio jednostavniji. U nastavku su opisani pojedini koraci uz prikaz relevantnog dijela koda za svaki korak.

- **Spajanje oblaka točaka**

Prikazan je prvi korak ovog postupka koji uključuje definiranje putanje do direktorija koji sadrži ulazne .ply datoteke:

```
input_dir = r'C:\ZAVRSNI -SLOVENIJA\EXTRACT'
```

Korištenjem glob modula, pretražujemo navedeni direktorij i pronalazimo sve datoteke s ekstenzijom .ply, koje će se kasnije učitati.

```
import open3d as o3d
import glob
import os
import numpy as np
```

Sljedeći korak je inicijalizacija prazne liste „all_points“ koja će pohraniti sve točke iz svih učitanih .ply datoteka.

```
# Lista za pohranu svih točaka
all_points = []
```

Za svaku pronađenu .ply datoteku, koristi se open3d.io.read_point_cloud funkcia iz biblioteke Open3D kako bi se učitao oblak točaka. Nakon toga, točke se izdvajaju u obliku numpy array-a i dodaju u listu all_points što je prikazano.

```
# Učitava svaku .ply datoteku i dodajte točke u all_points
for ply_file in ply_files:
    ply_cloud = o3d.io.read_point_cloud(ply_file)
    points = np.asarray(ply_cloud.points)
    all_points.append(points)
```

Nakon što su sve točke učitane i pohranjene u listu, spajaju se u jedan numpy array koristeći numpy.vstack funkciju. To omogućava da sve točke budu objedinjene u jednoj strukturi podataka.

```
# Spojjanje u jednu numpy array
all_points = np.vstack(all_points)
```

Prikazano je kreiranje Open3D PointCloud objekta. Inicijalizira se PointCloud objekt i postavljaju se njegove točke koristeći o3d.utility.Vector3dVector, koji pretvara numpy array u odgovarajući format za Open3D.

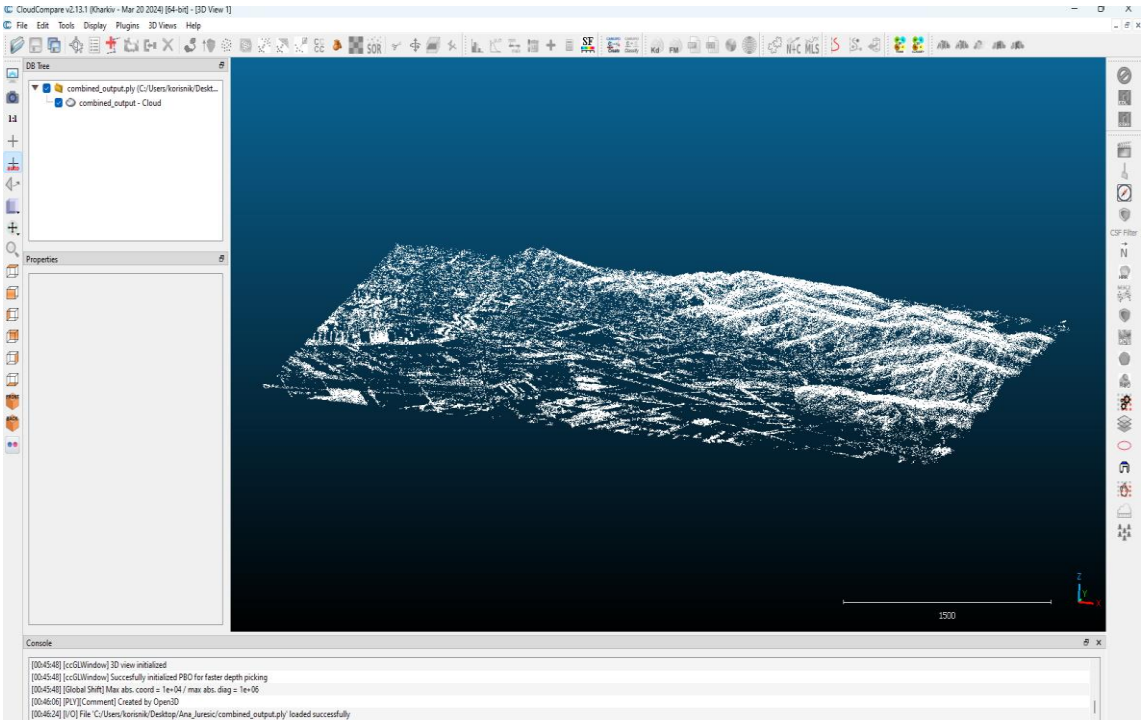
```
# Kreiranje Open3D PointCloud objekt
point_cloud = o3d.geometry.PointCloud()
point_cloud.points = o3d.utility.Vector3dVector(all_points)
```

U konačnici prikazan je objedinjeni oblak točaka koji se sprema u novu .ply datoteku koristeći open3d.io.write_point_cloud funkciju. Datoteka se sprema pod nazivom combined_output.ply u

definirani direktorij, čime se završava proces spajanja i pohrane oblaka točaka. Na slici (Slika 10) prikazane su spojene datoteke u programu CloudCompare.

```
# Spremanje u PLY
output_file_path = r'C:\ZAVRSNI -SLOVENIJA\EXTRACT\combined_output.ply'
o3d.io.write_point_cloud(output_file_path, point_cloud)

print(f'Successfully saved the combined point cloud to {output_file_path}')
```



Slika 10 Prikaz spojenih datotetka u CC

6.3.2. Priprema podataka tlocrta zgrada

Podaci o tlocrtima zgrada preuzeti su u shapefile formatu. Podaci su obrađeni softverom Qgis. Izvezeni su u GeoJSON format te je geometrija iz multipoligona pretvorena u poligone. Kada svaka zgrada čini jedan poligon, translatirana je cijela geometrija za isti iznos kao i LiDAR podatci.

- **Translacija tlocrta zgrada**

Prikazan je prvi korak koji je učitavanje GeoJSON datoteke koja sadrži geometrije zgrada koristeći biblioteku geopandas.

```
import geopandas as gpd
from shapely.affinity import translate
```

GeoJSON datoteka SLOVENIJA_SINGLEPARTS.geojson sadrži informacije o tlocrtima zgrada. Učitavanjem ove datoteke u GeoDataFrame objekt tlocrt, omogućeno je jednostavno manipuliranje i analiziranje prostornih podataka.

```
# Učitavanje GeoJSON datoteke
file_path = r'C:\ZAVRSNI -SLOVENIJA\ZGRADE\ispravno\SLOVENIJA_SINGLEPARTS.geojson'
tlocrt = gpd.read_file(file_path)
```

Nadalje su definirane translacije po x, y i z osima. Vrijednosti x_shift, y_shift i z_shift određuju koliko će se geometrija pomaknuti duž svake osi. U ovom slučaju pomak po x osi iznosi -463000 m, po y osi -99000 m, dok po z osi nema pomaka što prikazuje ovaj dio koda.

```
#  translacije po x, y i z osima
x_shift = -463000
y_shift = -99000
z_shift = 0
```

Definirana je funkcija shift_geometry koja koristi translate funkciju iz biblioteke shapely za pomicanje geometrija. Ova funkcija prima geometriju i vrijednosti pomaka po svakoj osi, te vraća novu, pomaknutu geometriju.

```
#  funkcija za pomak geometrije
def shift_geometry(geometry, x_shift, y_shift, z_shift):

    shifted_geometry = translate(geometry, xoff=x_shift, yoff=y_shift, zoff=z_shift)
    return shifted_geometry
```

Korištena je pply metoda kako bismo primijenili funkciju shift_geometry na svaku geometriju unutar GeoDataFrame-a tlocrt. Na taj način, sve geometrije su pomaknute za definirane vrijednosti po x, y i z osima.

```
tlocrt['geometry'] = tlocrt['geometry'].apply(lambda geom: shift_geometry(geom, x_shift, y_shift, z_shift))
```

Spremljeni su translatirani podatci natrag u GeoJSON format. Spremanjem datoteke pod nazivom slovenija_translatirano.geojson, osigurano je da su sve promjene pohranjene i spremne za daljnju obradu.

```
# Spremanje
output_file_path = r'C:\ZAVRSNI -SLOVENIJA\ZGRADE\ispravno\slovenija_translatirano.geojson'
tlocrt.to_file(output_file_path, driver='GeoJSON')
```

Na slici (Slika 11) i slici (Slika 12) prikazana je geometrija jedne od zgrada prije i nakon translacije zapisane u GeoJSON formatu.


```
{
"type": "FeatureCollection",
"name": "SLOVENIJA_SINGLEPARTS",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:EPSG::3794" } },
"features": [
{ "type": "Feature", "properties": { "FEATUREID": "STAVBE_NADZEMNI_TLORIS.100200000120003740", "EID_STAVBA": "100200000120003740" },
"geometry": { "type": "Polygon", "coordinates": [ [ [ 548696.01, 158340.78 ], [ 548694.63, 158340.29 ], [ 548693.28000000002794,
158344.04 ], [ 548691.959999999962747, 158343.57 ], [ 548690.459999999962747, 158347.60999999998603 ], [ 548691.810000000055879,
158348.1 ], [ 548690.810000000055879, 158350.87 ], [ 548693.900000000023283, 158351.980000000010477 ], [ 548693.449999999953434,
158353.230000000010477 ], [ 548693.650000000023283, 158353.299999999988358 ], [ 548692.51, 158356.5 ], [ 548701.75, 158359.72 ],
[ 548702.85999999998603, 158356.57 ], [ 548703.96999999997206, 158356.97 ], [ 548707.339999999967404, 158347.56 ],
[ 548706.800000000046566, 158347.37 ], [ 548707.12, 158346.43 ], [ 548705.839999999967404, 158345.96 ], [ 548706.35999999998603,
158344.57 ], [ 548700.199999999953434, 158342.29 ], [ 548699.310000000055879, 158340.76 ], [ 548697.74, 158340.200000000011642 ],
[ 548696.01, 158340.78 ] ] ] } },
```


Slika 11 Geometrija prije translacije u GeoJSON formatu


```
{
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:EPSG::3794" } },
"features": [
{ "type": "Feature", "properties": { "FEATUREID": "STAVBE_NADZEMNI_TLORIS.100200000120003740", "EID_STAVBA": "100200000120003740" },
"geometry": { "type": "Polygon", "coordinates": [ [ [ 85696.010000000009313, 59340.779999999998836 ], [ 85694.630000000004657,
59340.290000000008149 ], [ 85693.28000000002794, 59344.040000000008149 ], [ 85691.959999999962747, 59343.570000000006985 ],
[ 85690.459999999962747, 59347.60999999998603 ], [ 85691.810000000055879, 59348.100000000005821 ], [ 85690.810000000055879,
59350.869999999995343 ], [ 85693.900000000023283, 59351.980000000010477 ], [ 85693.449999999953434, 59353.230000000010477 ],
[ 85693.650000000023283, 59353.299999999988358 ], [ 85692.510000000009313, 59356.5 ], [ 85701.75, 59359.720000000001164 ],
[ 85702.85999999998603, 59356.570000000006985 ], [ 85703.96999999997206, 59356.970000000001164 ], [ 85707.339999999967404,
59347.559999999997672 ], [ 85706.800000000046566, 59347.369999999995343 ], [ 85707.119999999995343, 59346.429999999993015 ],
[ 85705.839999999967404, 59345.959999999991851 ], [ 85706.35999999998603, 59344.570000000006985 ], [ 85700.199999999953434,
59342.290000000008149 ], [ 85699.310000000055879, 59340.760000000009313 ], [ 85697.739999999990687, 59340.200000000011642 ],
[ 85696.010000000009313, 59340.779999999998836 ] ] ] } },
```


Slika 12 Geometrija nakon translacije u GeoJSON formatu

6.4. Određivanje visine najnižeg dijela zgrade

6.4.1. Učitavanje biblioteka i modula

Prvi korak ovog prikazanog postupka uključuje učitavanje nekoliko ključnih biblioteka koje će se koristiti u analizi i obradi 3D geometrijskih podataka.

Prvo, korišten je „import open3d as o3d“ kako bi bio omogućen rad s bibliotekom Open3D, koja je specijalizirana za obradu 3D podataka. Open3D biblioteka će omogućiti učitavanje i manipulaciju oblakom točaka koji se nalazi u .ply datoteci, što je ključno za daljnju analizu visine zgrada.

Nakon toga slijedi „import geopandas as gpd“, koji omogućuje korištenje GeoPandas biblioteke. Ova biblioteka se koristi za rad s geoprostornim podacima, poput poligona koji predstavljaju tlocrt zgrada. Korištenjem ove biblioteke, može se jednostavno upravljati i analizirati te geoprostorne podatke.

Treći korak je „import numpy as np“, gdje se učitava NumPy biblioteka, koja je osnovna za rad s numeričkim podacima i nizovima. NumPy će biti korišten za matematičke operacije nad podacima, kao što je izračun srednje visine točaka unutar oblaka.

Nadalje, korišten je „import json“ kako bi se omogućio rad s JSON formatom podataka. Ova biblioteka se koristiti za generiranje i spremanje rezultata analize u GeoJSON formatu, što je standardni format za razmjenu geografskih podataka na webu.

Na kraju, korišten je“ from shapely.geometry import box“ kako bi se omogućila funkcionalnost kreiranja pravokutnika (bounding box-a) iz zadanih koordinata. Ova funkcija omogućuje definiranje specifičnog područja interesa unutar kojeg će se ograničiti analiza zgrada i oblaka točaka.

```
import open3d as o3d
import geopandas as gpd
import numpy as np
import json
from shapely.geometry import box
```

6.4.2. Učitavanje oblaka točaka i kreiranje bounding box-a

Prikazano je učitavanje oblaka točaka terena iz specificirane .ply datoteke koristeći open3d biblioteku. Ova operacija omogućava rad s 3D podacima o terenu.

```
# Učitavanje oblaka točaka terena
pcd_teren = o3d.io.read_point_cloud(ply_teren_path)
```

Nakon učitavanja oblaka točaka, korištena je metoda“ get_axis_aligned_bounding_box()“ kako bi se dobio pravokutnik koji koincidira sa koordinatama oblaka točaka. Ovaj pravokutnik je definiran svojim minimalnim i maksimalnim koordinatama u prostoru, odnosno predstavlja najnižu i najvišu točku oblaka točaka duž X, Y i Z osi.

```
# kreiranje bounding box-a oblaka točaka
bbox = pcd_teren.get_axis_aligned_bounding_box()
```

Izdvojene su minimalne (min_bound) i maksimalne (max_bound) koordinate bounding box-a. Te koordinate su važne jer definiraju granice područja koje pokriva oblak točaka. Te granice će biti korištene kasnije za filtriranje podataka nad kojim će se provoditi analiza.

```
# Krajnje koordinate bounding box-a
min_bound = bbox.min_bound
max_bound = bbox.max_bound
```

Na kraju, se ispisuju minimalne i maksimalne koordinate bounding box-a kako bi se uvjerili da su točno određene granice oblaka točaka.

```
print(f"Bounding Box Min: {min_bound}, Max: {max_bound}")
```

6.4.3. Učitavanje geometrije tlocrta zgrada te filtriranje zgrada unutar područja interesa

Prikazano je učitavanje geometrijskih podataka zgrada iz GeoJSON datoteke koristeći GeoPandas. Datoteka sadrži informacije o položaju i obliku zgrada u formatu vektorskih podataka (poligoni). Učitani podatci su pohranjeni u GeoDataFrame, što omogućava jednostavnu analizu i manipulaciju geografskih informacija.

```
# Učitavanje geometrije tlocrta zgrada
tlocrt_zgrada = gpd.read_file(geojson_tlocrt_path)
```

Ovaj dio koda prikazuje korištenje krajnjih koordinata bounding box-a oblaka točaka (koje su dobivene iz .ply datoteke) kako bi se definiralo područje interesa „area_of_interest“. Funkcija „box“ iz biblioteke Shapely kreira pravokutnik koji obuhvaća područje između minimalnih i maksimalnih koordinata po X i Y osi. Ovo područje interesa služi za filtriranje zgrada koje se nalaze unutar tih granica.

```
# Definiranje područja interesa (bounding box) koristeći rubne koordinate iz .ply datoteke
area_of_interest = box(min_bound[0], min_bound[1], max_bound[0], max_bound[1])
```

U prikazanom koraku filtriraju se sve zgrade koje se nalaze unutar definiranog područja interesa „area_of_interest“. Funkcija „intersects“ provjerava koji poligoni (zgrade) iz GeoDataFrame-a

„tlocrt_zgrada“ presijecaju ili se nalaze unutar definiranog bounding box-a. Rezultat je novi GeoDataFrame „zgrade_u_podrucju“, koji sadrži samo zgrade unutar područja interesa.

```
# Filtriranje zgrada unutar bounding box-a .ply datoteke
zgrade_u_podrucju = tlocrt_zgrada[tlocrt_zgrada.intersects(area_of_interest)]
```

6.4.4. Pretvaranje oblaka točaka u Numpy , izgradnja k-d stabla te inicijalizacija liste za pohranu

Prikazano je pretvaranje točaka terena iz oblaka točaka (pcd_teren) u numpy array. Što je ključno za daljnju analizu jer numpy array omogućava brže i efikasnije matematičke operacije nad velikim skupovima podataka. Ove točke koristit će se kasnije za izračunavanje visine najnižeg dijela zgrada.

```
# Pretvaranje točaka terena u numpy array
points_array = np.asarray(pcd_teren.points)
```

Prikazano je koristenje k-d stabla (k-dimensional tree), što je struktura podataka koja omogućuje brzu pretragu najbližih susjeda u višedimenzionalnim prostorima, često korištena u analizama oblaka točaka. „KDTreeFlann“ iz biblioteke Open3D izgrađuje ovo stablo koristeći oblak točaka terena (pcd_teren). Ova struktura se koristiti za brzo pronalaženje točaka terena koje su najbliže centroidu svake zgrade.

```
# Izgradnja k-d stabla na oblaku točaka terena
pcd_terr_tree = o3d.geometry.KDTreeFlann(pcd_teren)
```

Prikazana je inicijalizacija praznog GeoJSON objekta, u kojem će se pohranjivati rezultati. GeoJSON je standardni format za kodiranje geoprostornih podataka u JSON obliku. Ovaj objekt će na kraju sadržavati sve zgrade sa svojim pripadajućim visinama, definiranim kao geoprostorne značajke (features).

```
# Lista za pohranu rezultata u GeoJSON formatu
rezultati_geojson = {"type": "FeatureCollection", "features": []}
```

6.4.5. Pronalaženje najbližih točaka te izračun srednje vrijednosti najnižeg dijela zgrade

Prikazan je korak prolaženja kroz svaki poligon (zgradu) unutar filtriranog skupa podataka „zgrade_u_podrucju“. Za svaku zgradu izračunava se centroid (središnja točka) njenog poligona, koja se koristi kao „search_point“ za pretragu najbližih točaka terena unutar k-d stabla. Metoda „search_knn_vector_3d“ traži 100 najbližih točaka u odnosu na centroid zgrade.

```
# Iteracija kroz svaki poligon u filtriranoj geometriji tlocrta zgrada
for index, poligon in zgrade_u_podrucju.iterrows():
    spt = poligon.geometry.centroid
    search_point = np.array([spt.x, spt.y, 0]).astype("float64")
    [k, idx, _] = pcd_terr_tree.search_knn_vector_3d(search_point, 100)
```

Prikazano je postavljanje uvjeta kojim će se izračunati visina jedino ako pretraga pronađe barem jednu točku ($k > 0$), tada se izračunava prosječna visina (Z vrijednost) tih najbližih točaka, što predstavlja visinu najnižeg dijela zgrade. Kreira se GeoJSON objekt koji predstavlja trenutnu zgradu. Objekt sadrži:

-geometry: Točka (centroid) koja predstavlja zgradu u 2D prostoru.

-properties: Informacije o zgradi, uključujući jedinstveni identifikator zgrade (uzet iz atributa FEATUREID ili EID_STAVBA) i izračunatu visinu. Prikazano je generiranje GeoJSON objekta koji se dodaje u zbirku rezultata, koja sadrži sve obrađene zgrade sa pripadajućim visinama. Ukoliko nije pronađena niti jedna točka terena u blizini zgrade, ispisuje se poruka koja upozorava kako nema dovoljno podataka za izračun visine za tu specifičnu zgradu.

```
if k > 0:  # Ako ima barem jedna točka
    # Izračunavanje aritmetičke sredine Z vrijednosti najbližih točaka
    srednja_visina = np.mean(points_array[idx, 2])
    # Generiranje GeoJSON formata
    feature = {
        "type": "Feature",
        "geometry": {
            "type": "Point",
            "coordinates": [spt.x, spt.y]
        },
        "properties": {
            "building_id": poligon['FEATUREID'] if 'FEATUREID' in poligon else poligon['EID_STAVBA'],
            "visina": srednja_visina
        }
    }
```

Prikazani dio koda zadužen je za spremanje svih prethodno obrađenih i prikupljenih podataka u GeoJSON datoteku.

```
        }
        rezultati_geojson["features"].append(feature)
    else:
        print(f"Nema dovoljno podataka za zgradu s ID: {poligon['FEATUREID'] if 'FEATUREID' in poligon else poligon['EID_STAVBA']}")
```

Prilikom toga podešene su dvije opcije. Opcija with open(output_path_geojson, 'w') as geojson_file koja otvara datoteku na putanji definiranoj u varijabli output_path_geojson u načinu rada za pisanje ('w'). Ako datoteka s tim imenom već postoji, bit će prepisana. Korištenjem with osigurava se da će datoteka biti ispravno zatvorena nakon što je završeno sa pisanjem, čak i ako se dogodi neka greška tijekom izvođenja koda. Opcija json.dump(rezultati_geojson, geojson_file, indent=2) sadržaj varijable rezultati_geojson (koja sadrži sve informacije o zgradama i njihovim visinama) pretvara se u JSON format i zapisuje u datoteku. Parametar indent=2 koristi se za lijepo formatiranje izlaza, gdje se JSON struktura uvlači s 2 razmaka, što čini datoteku čitljivijom (Slika 13).

```
# Spremanje rezultata u GeoJSON datoteku
with open(output_path_geojson, 'w') as geojson_file:
    json.dump(rezultati_geojson, geojson_file, indent=2)
```

U konačnici, ovaj redak ispisuje poruku u konzolu, informirajući korisnika da su rezultati uspješno spremljeni u GeoJSON datoteku na specificiranoj putanji. To služi kao potvrda da je proces dovršen bez grešaka.

```
print(f"Rezultati su spremljeni u {output_path_geojson}")
```

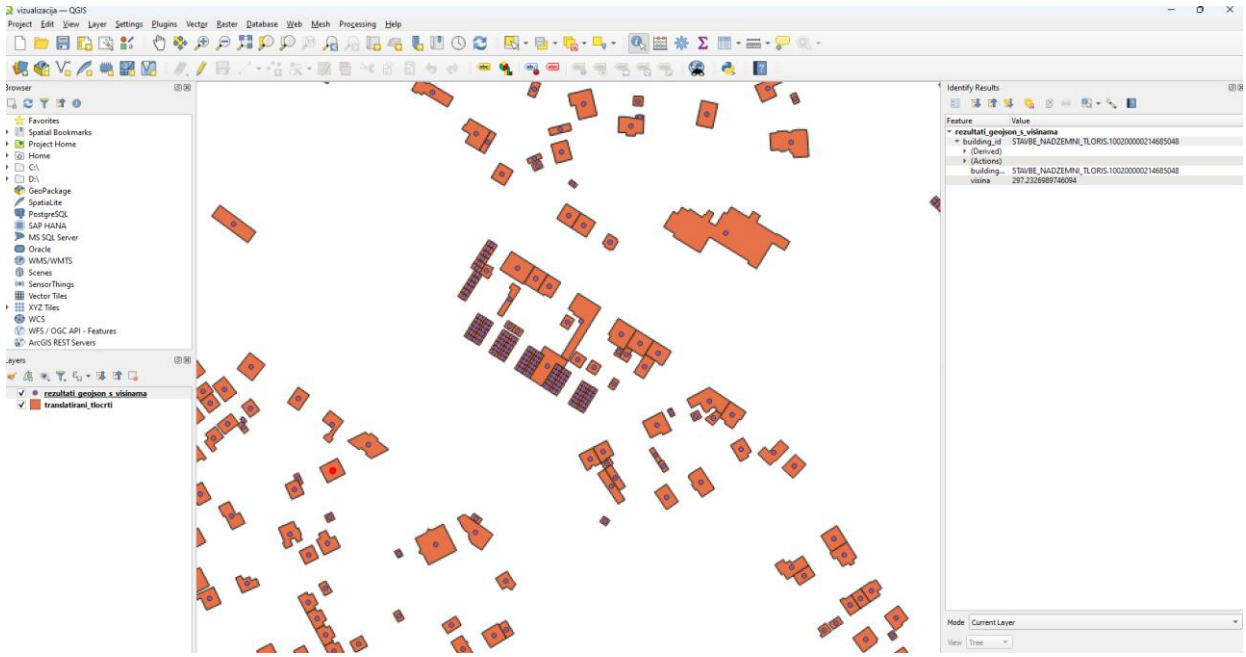
```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          296.49379503955197,
          797.8778623104928
        ]
      },
      "properties": {
        "building_id": "STAVBE_NADZEMNI_TLORIS.100201000000068259",
        "visina": 312.48820068359373
      }
    },
```


Slika 13 Primjer rezultata u GeoJSON formatu

7. Analiza rezultata

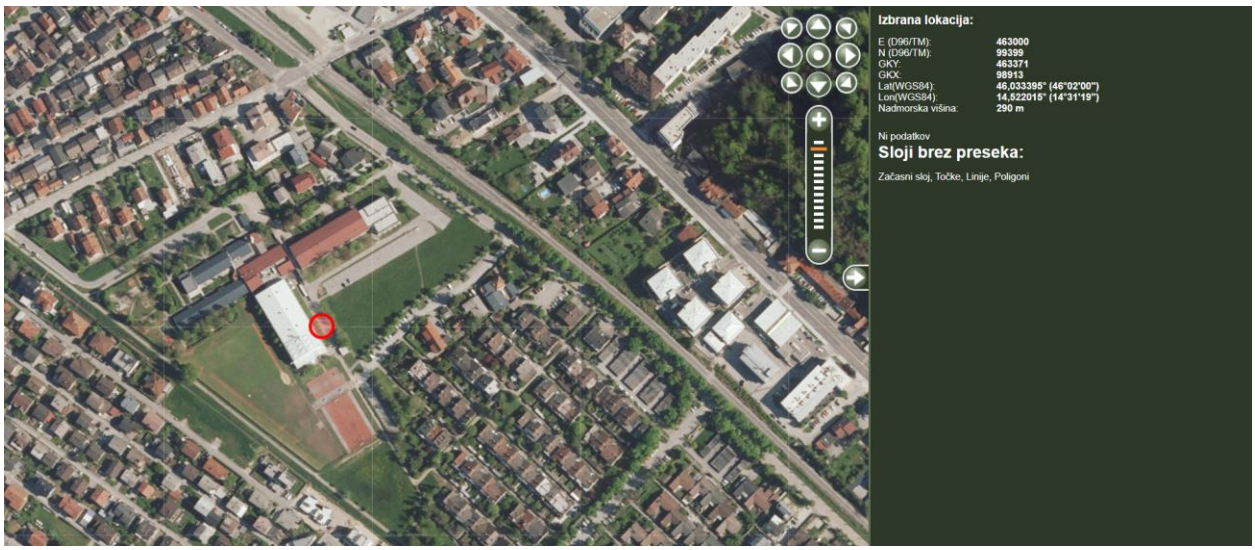
Provedena je analiza za područje 1km². Za područja dimenzija 1 km² koristeći oblak točaka terena i geometriju tlocrta zgrada te koristeći računalo s procesorom 12th Gen Intel(R) Core(TM) i7-12650H, 16 GB RAM-a i operativnim sustavom Windows 11 Pro. Cjelokupno trajanje obrade podataka za izračun visina najnižih dijelova svih zgrada unutar navedenog područja (653) iznosilo je 1 minutu i 50 sekundi.

Dobivene visine najnižih dijelova zgrada mogu se dalje obrađivati u GIS alatima (Slika 14) ili koristiti za daljnju analizu.

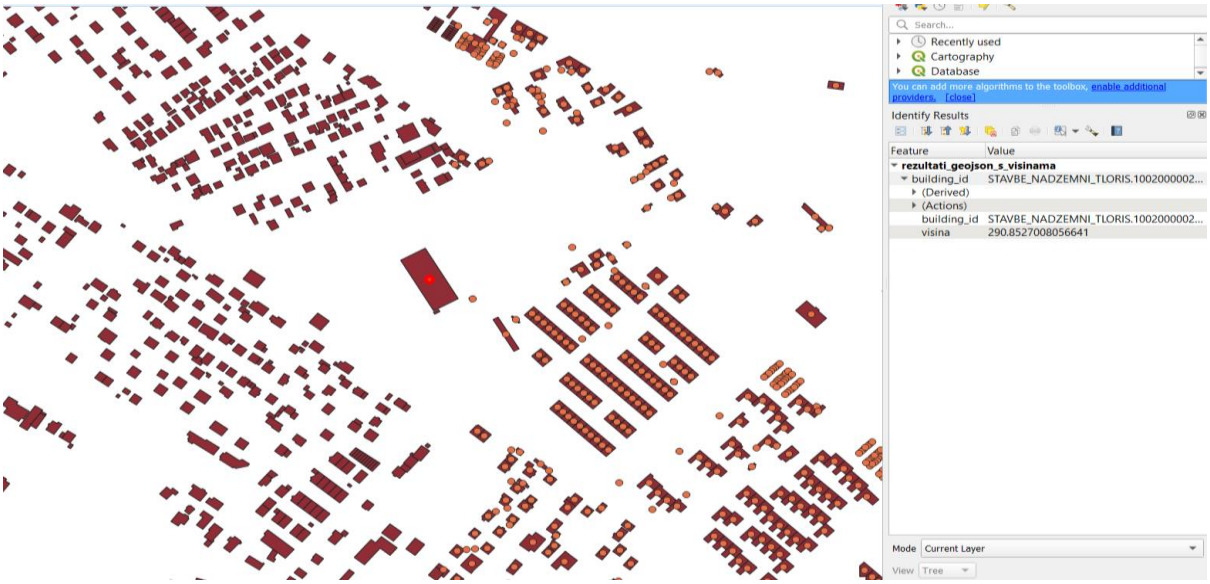


Slika 14 Vizualizacija rezultat u QGIS-u

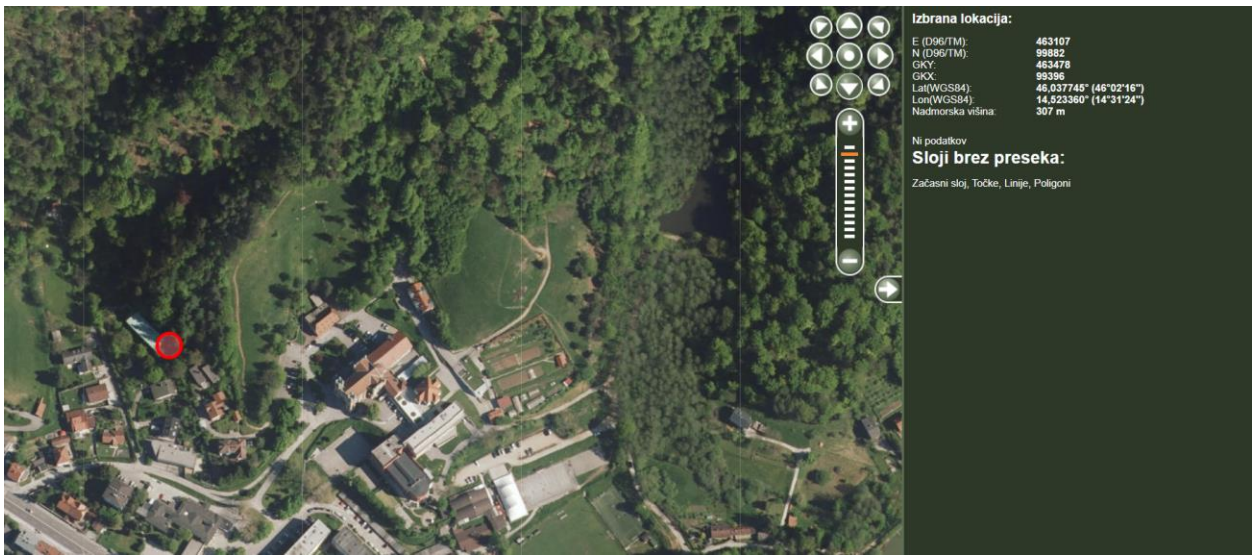
Točnost algoritma provjerena je na način da se u GIS pregledniku(https://gis.arso.gov.si) očitala vrijednost Z koordinata točke terena koja se nalazi uz zgradu te se ta vrijednost usporedila sa izračunatom visinom prikazanom u QGIS-u. Slika 15, Slika 16, Slika 17 i Slika 18 prikazuju uspoređene zgrade te se može zaključiti kako su visine slične te je time zadovoljena kontrola.



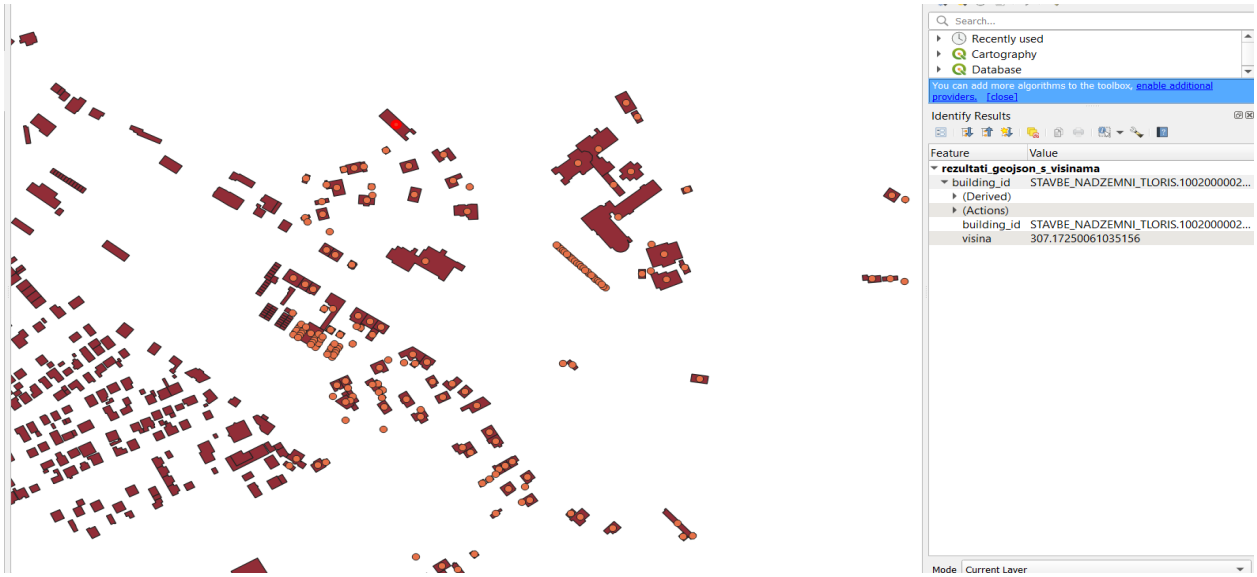
Slika 15 Kontrola točnosti web preglednik 1



Slika 16 Kontrola točnosti QGIS 1



Slika 17 Kontrola točnosti web preglednik 2



Slika 18 Kontrola točnosti QGIS 2

8. Zaključak

U ovom radu predstavljena je metoda određivanja visine najnižeg dijela zgrade iz LiDAR podataka. Metoda se temelji na računanju visine programskim jezikom Python koristeći algoritme FLANN i K-D stablo. Ako bi se za svaku zgradu pretraživao cijeli oblak točaka, postupak bi bio izuzetno spor i nepraktičan. Zbog toga je ključno korištenje KDF algoritma, koji omogućava brzo pretraživanje i filtriranje točaka unutar oblaka. KDF algoritam strukturira podatke na način koji omogućuje brzo pronalaženje najbližih susjeda, što značajno ubrzava cijeli proces računanja visine. Na temelju izdvojenih točaka terena računa se srednja visina koja u konačnici predstavlja visinu najnižeg dijela zgrade. Daljnjim usavršavanjem ove metode, kao i integracijom dodatnih algoritama za obradu podataka te poboljšanjem kvalitete ulaznih podataka moguće je dodatno poboljšati brzinu i preciznost, čime se otvaraju nove mogućnosti za primjenu u raznim područjima znanosti i industrije.

Literatura

[1] Santos, Teresa & A.M.Rodrigues, & Tenedório, José. (2013). Characterizing urban volumetry using LiDAR data. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XL-4/W1. 10.5194/isprsarchives-XL-4-W1-71-2013., dostupno 24.06.2024.

[2] Galvanin, Edinéia & Dal Poz, Aluir. (2012). Extraction of Building Roof Contours From LiDAR Data Using a Markov-Random-Field-Based Approach. IEEE T. Geoscience and Remote Sensing. 50. 981-987. 10.1109/TGRS.2011.2163823., dostupno 24.06.2024.

[3] Dukai, B., Ledoux, H., and Stoter, J. E.: A MULTI-HEIGHT LOD1 MODEL OF ALL BUILDINGS IN THE NETHERLANDS, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-4/W8, 51–57, https://doi.org/10.5194/isprs-annals-IV-4-W8-51-2019, 2019., dostupno 24.06.2024.

[4] AAMHatch, 2006. Airborne Laser Scanning Technical Specifications, http://www.aamhatch.com.au (accessed 27 Sept. 2006), dostupno 25.06.2024.

[5] https://www.danielgm.net/cc/doc/qCC/CloudCompare%20v2.6.1%20-%20User%20manual.pdf, dostupno 24.06.2024.

[6] https://www.cs.unc.edu/~isenburg/lastools/download/laszip.pdf, dostupno 24.06.2024.

[7] https://paulbourke.net/dataformats/ply/ , dostupno 25.06.2024.

[8] https://alm.engr.colostate.edu/cb/displayDocument/GeoJSON+Spec.rfc7946.pdf?doc_id=26549, dostupno 27.06.2024.

[9] Python: The Most Advanced Programming Language for Computer Science Applications Akshit J. Dhruv, Reema Patel and Nishant Doshi Computer Science and Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat ,dostupno 29.06.2024.

[10] https://en.wikipedia.org/wiki/Python_(programming_language), dostupno 29.06.2024.

[11] https://numpy.org/doc/1.20/numpy-user.pdf, dostupno 01.07.2024.

[12] https://www.open3d.org/, dostupno 01.07.2024.

[13] https://geopandas.org/en/stable/getting_started/introduction.html, dostupno 01.07.2024.

[14] https://www.geeksforgeeks.org/ball-tree-and-kd-tree-algorithms/, dostupno 01.07.2024.

[15] https://github.com/flann-lib/flann, dostupno 01.09.2024.

[16] https://www.fit.vut.cz/person/ibarina/public/pub/VGE/reading/flann_manual-1.6.pdf0, dostupno 29.07.2024.

[17] Izvedba laserskega skeniranja Slovenije Blok 35 – tehnično poročilo o izdelavi izdelkov, dostupno 29.07.2024.

[18] https://ipi.eprostor.gov.si/jgp/data, dostupno 29.07.2024.

Popis slika

Popis tablica

Dodaci

Dodatak 1-Translacija tlocrta

```
import geopandas as gpd
from shapely.affinity import translate

# Učitavanje GeoJSON datoteke
file_path = r'C:\ZAVRSNI -
SLOVENIJA\ZGRADE\ispravno\SLOVENIJA_SINGLEPARTS.geojson'
tlocrt = gpd.read_file(file_path)

#  translacije po x, y i z osima
x_shift = -463000
y_shift = -99000
z_shift = 0

#  funkcija za pomak geometrije
def shift_geometry(geometry, x_shift, y_shift, z_shift):

    shifted_geometry = translate(geometry, xoff=x_shift, yoff=y_shift, zoff=z_shift)
    return shifted_geometry

tlocrt['geometry'] = tlocrt['geometry'].apply(lambda geom: shift_geometry(geom, x_shift, y_shift,
z_shift))

# Spremanje
output_file_path = r'C:\ZAVRSNI -
SLOVENIJA\ZGRADE\ispravno\slovenija_translatirano.geojson'
tlocrt.to_file(output_file_path, driver='GeoJSON')
```

Dodatak 2-Spajanje oblaka točaka

```
import open3d as o3d
import glob
import os
import numpy as np

input_dir = r'C:\ZAVRSNI -SLOVENIJA\EXTRACT'

# nalazi sve .ply datoteke
ply_files = glob.glob(os.path.join(input_dir, '*.ply'))

# Lista za pohranu svih točaka
all_points = []
```

```
# Učitava svaku .ply datoteku i dodajte točke u all_points
for ply_file in ply_files:
    ply_cloud = o3d.io.read_point_cloud(ply_file)
    points = np.asarray(ply_cloud.points)
    all_points.append(points)

# Spojjanje u jednu numpy array
all_points = np.vstack(all_points)

# Kreiranje Open3D PointCloud objekt
point_cloud = o3d.geometry.PointCloud()
point_cloud.points = o3d.utility.Vector3dVector(all_points)

# Spremanje u PLY
output_file_path = r'C:\ZAVRSNI -SLOVENIJA\EXTRACT\combined_output.ply'
o3d.io.write_point_cloud(output_file_path, point_cloud)

print(f'Successfully saved the combined point cloud to {output_file_path}')
```

Dodatak 3-Računanje visina najnižeg dijela zgrade

```
import open3d as o3d
import geopandas as gpd
import numpy as np
import json
from shapely.geometry import box

# Putanje do datoteka
ply_teren_path = r'C:\ZAVRSNI -SLOVENIJA\LIDAR\GK_463_99.laz
TEREN_REDUCIRANO.ply'
geojson_tlocrt_path = r'C:\ZAVRSNI -SLOVENIJA\slovenijaa_pomaknuto.geojson'
output_path_geojson = r'C:\ZAVRSNI -SLOVENIJA\rezultati_geojson_s_visinama.geojson'

# Učitavanje oblaka točaka terena
pcd_teren = o3d.io.read_point_cloud(ply_teren_path)

# kreiranje bounding box-a oblaka točaka
bbox = pcd_teren.get_axis_aligned_bounding_box()

# Krajnje koordinate bounding box-a
min_bound = bbox.min_bound
max_bound = bbox.max_bound

print(f"Bounding Box Min: {min_bound}, Max: {max_bound}")

# Učitavanje geometrije tlocrta zgrada
tlocrt_zgrada = gpd.read_file(geojson_tlocrt_path)

# Definiranje područja interesa (bounding box) koristeći rubne koordinate iz .ply datoteke
area_of_interest = box(min_bound[0], min_bound[1], max_bound[0], max_bound[1])
```

```
# Filtriranje zgrada unutar bounding box-a .ply datoteke
zgrade_u_podrucju = tlocrt_zgrada[tlocrt_zgrada.intersects(area_of_interest)]

# Pretvaranje točaka terena u numpy array
points_array = np.asarray(pcd_teren.points)

# Izgradnja k-d stabla na oblaku točaka terena
pcd_terr_tree = o3d.geometry.KDTreeFlann(pcd_teren)

# Lista za pohranu rezultata u GeoJSON formatu
rezultati_geojson = {"type": "FeatureCollection", "features": []}

# Iteracija kroz svaki poligon u filtriranoj geometriji tlocrta zgrada
for index, poligon in zgrade_u_podrucju.iterrows():
    spt = poligon.geometry.centroid
    search_point = np.array([spt.x, spt.y, 0]).astype("float64")
    [k, idx, _] = pcd_terr_tree.search_knn_vector_3d(search_point, 100)

    if k > 0:  # Ako ima barem jedna točka
        # Izračunavanje aritmetičke sredine Z vrijednosti najbližih točaka
        srednja_visina = np.mean(points_array[idx, 2])
        # Generiranje GeoJSON formata
        feature = {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [spt.x, spt.y]
            },
            "properties": {
                        "building_id": poligon['FEATUREID'] if 'FEATUREID' in poligon else
poligon['EID_STAVBA'],
                "visina": srednja_visina
            }
        }
        rezultati_geojson["features"].append(feature)
    else:
        print(f"Nema dovoljno podataka za zgradu s ID: {poligon['FEATUREID'] if 'FEATUREID'
in poligon else poligon['EID_STAVBA']}")

# Spremanje rezultata u GeoJSON datoteku
with open(output_path_geojson, 'w') as geojson_file:
    json.dump(rezultati_geojson, geojson_file, indent=2)

print(f"Rezultati su spremljeni u {output_path_geojson}")
```