SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**DIPLOMSKI PROJEKT**

# Detekcija ključnih točaka iz LIDAR snimki s bespilotne letjelice

*Marko Ćurlin*
Voditelj: *prof. dr. sc. Zdenko Kovačić*

Zagreb, ožujak 2022.

# SADRŽAJ

# 1. Uvod

Problem određivanja ključnih točaka u vinogradu se svodi na pronalazk ulaznih i izlaznih točaka u redove vinograda. Ovo se radi tako da bi robot znao gdje se pozicionirati u vinogradu tako da može započeti s prskanjem vinograda ili obiranjem trsova. Jednom kada se robot pozicionira na ulaz vinograda, iz ovog riješenja zna za taj isti ulaz i odgovarajući izlaz. Također je odmah definiran i sljedeći red u koji treba krenuti.

Implementacija algoritma je izvršena u programskom jeziku *Python* uz pomoć knjižnica otvorenog koda *open3d* [1], *numpy* [2] i *sklearn* [3]. Pri dizajniranju programskog riješenja uzela se u obzir modularnost riješenja, ostavljajući mjesta za lagane izmjene dijelova koda, uključujući i ubacivanje koda drugačijeg programskog jezika (npr. C++ se također često koristi u obradi podataka oblaka točaka). Glavni koraci algoritma su definirani sljedećim redoslijedom:

1. Određivanje nagiba vinograda - Ovaj podatak će nam služiti kao ulaz u sljedećem koraku.

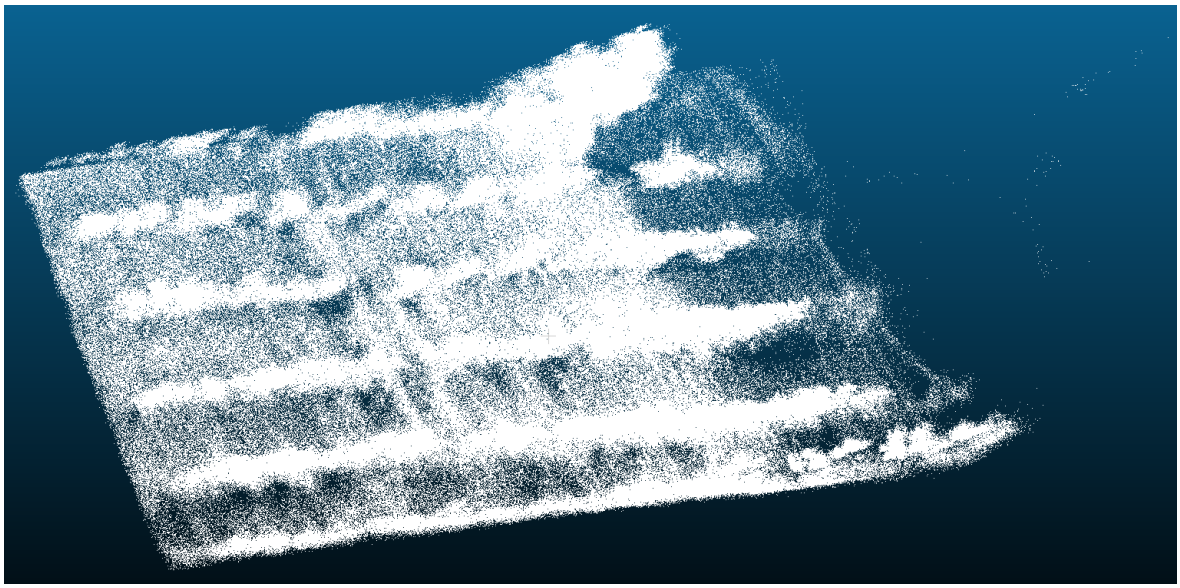
2. Uklanjanje točaka oblaka koje reprezentiraju tlo - Obavlja se pomoću podatka o nagibu vinograda. Tlo u našem slučaju predstavlja smetnju, kako pokušavamo isključivo odrediti gdje se nalaze ulazi u redove vinograda, relativno prema trsovima.

3. Klasificirati točke oblaka u trsove - Iako ćemo vidjeti da je golom oku očito koje točke u oblaku pripadaju zajedničkom trsu, tako nešto je malo teže grupirati računalno.

4. Pronaći ulaze između redova trsova - U konačnom koraku dolazimo do traženog riješenja.

# 2. Određivanje nagiba vinograda

Prije nego krenemo na sam algoritam, predstaviti ćemo podatke te njihova svojstva koja su iskorištena u razvitku ovog programskog riješenja. Ulazni podaci su bilo kakva reprezentacija oblaka točaka. Konkretno se ovdje radilo o unaprijed snimljenom oblaku točaka vinograda, spremljenog u PLY format [4], bez korištenja *face* svojstva. Primjer takvog vinograda prikazanog pomoću CloudCompare programa se vidi na slici 2.1.



**Slika 2.1:** Primjer oblaka točaka vinograda koji služi kao ulaz

Iz slike 2.1 se može primjetiti par stvari: tlo, linije trsova okomite na tlo te nekolicina točaka koje su razbacane i predstavljaju šum LIDAR kamere. To zadnje zapažanje nas dovodi do prvog modula ovog programskog riješenja: *remove_noise.py*. Ono sadrži glavnu funkciju *remove_noise* koja je prikazan u isječku 1.

```
def remove_noise(point_cloud, nr_neighbours=50, std_ratio=1):
    inlier_cloud, _ = point_cloud.remove_statistical_outlier(
    nb_neighbors=nr_neighbours, std_ratio=std_ratio)
    return inlier_cloud
```

**Isječak 1:** Uklanjanje šuma

Ulazni argument `point_cloud` je tip objekta `open3d.geometry.PointCloud`, te se koristi njegova ugrađena funkcija[1] za statističko uklanjanje točaka van zajednice koristeći normalnu distribuciju. Vrijednosti za broj susjeda i distribuciju su dobivene eksperimentalno, ali vrijednosti od 50 susjeda i 1 standardne devijacije su se pokazale kao dobro riješenje za očuvanje točaka s mnogo susjeda. Znamo da su nam elementi od važnosti (tlo i redovi) gusto uzorkovani, stoga to daje dobre rezultate.

Sljedeći korak je odrediti nagib vinograda. Time se zapravo određuje ravnina paralelna s tlom koja može poslužiti uklanjanju tla iz oblaka točaka. Ovdje se ujedno pokazuje i prvo ograničenje ovog programskog riješenja. Da bi se uspješno uklonila površina, i nastavilo s izvođenjem programa, potrebno je da se vinograd nalazi na relativno ravnoj plohi (bez obzira na nagib). Ovaj dio bi se mogao poboljšati tako da se definira tlo kao neki kompleksniji oblik, ali to nije bilo obuhvaćenom ovim riješenjem. Pokušana su tri načina određivanja nagiba, sva tri su se pokazala učinkovita.

- Linearna regresija
- Analiza glavnih komponenti (PCA - Principal component analysis)
- RANSAC - Random sample consensus

Linearna regresija i RANSAC su obje implementirane programski, dok se rezultat analize glavnih komponenti dobio iz programa CloudCompare, koristeći implementiranu metodu *Fit*. Ista se može dobiti programski pozivajući iz terminala naredbu (i parsirajući nastalu datoteku):

```
$ CloudCompare -O <point_cloud_file> -BEST_FIT_PLANE
```

Linearna regresija je implementirana pomoću knjižnice *sklearn* unutar modula *find_plane.py*. Ondje funkcija `find_plane` vraća normalu ravnine te točku na ravnini, što unikatno definira ravninu u 3D prostoru. Kod funckije je vidiljiv u isječku 2.

Ovakav pristup odgovara pronalasku ravnine koja se nalazi nešto poviše ravnine tla. Razlog tome možemo pronaći ako promatramo slučaju u kojem postoji samo tlo. Znamo da bi linearna regresija u tom slučaju točno posložila tražene parametre tako da odgovaraju ravnini. Dodamo li tome uniformno raspoređeni skup točaka koji predstavljaju trsove, estimator linearne regresije će pokušati smanjiti srednju pogrešku tako da se približi tim točkama, time se podižući iznad tla. Pošto su te točke uniformno raspoređene po trsovima, to podizanje će se desiti kao jedna vrsta okomite translacije.

Za pronalažanje ravnine pomoću RANSAC algoritma se koristi funkcija `PointCloud` objekta, `segment_plane`. Daje slične rezultate, uz najprimjetniju razliku toga da je u

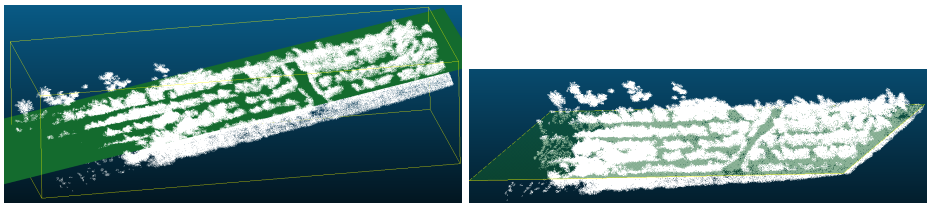
[1] Isti rezultat se dobio i koristeći poznatu PCL knjižnicu [5] u programskom jeziku C++. Riješenje je bilo ekspresivnije, te je trajalo dulje, stoga se zabralo riješenje u Pythonu.

```
from sklearn.linear_model import LinearRegression

def find_plane(point_cloud_data):
    xy, z = point_cloud_data[:, :2], point_cloud_data[:, 2]
    estimator = LinearRegression().fit(xy, z)

    plane_normal = util.get_normal(estimator.coef_)
    point_on_plane = util.get_point_on_cloud(estimator.coef_, estimator.intercept_)

    return plane_normal, point_on_plane
```

**Isječak 2:** Određivanje ravnine paralelne s tlom

slučaju linearne regresije i analize glavnih komonenti ravnina već odmaknuta od tla, dok je kod RANSAC-a ta ploha točno na tlu. Rezultati su prikazani na slici 2.2.

U konačnom riješenju se koristila metoda linearne regresije. Razlog biranja umjesto RANSAC je brzina. RANSAC mora proći kroz mnogo iteracija u kojima računa različite i nasumične kombinacije ravnina, dok linearna regresija to jednostavno riješi matematički. Razlog biranja umjesto PCA je jednostavnost korištenja programskog riješenja u Pythonu, bez potrebe za vanjskim ovisnostima. U slučaju želje za korištenjem nekog od druga dva riješenja, lako se zamijene u konačnom riješenju umjesto sadašnje korištenog modula.



**(a)** Linerna regresija **(b)** Analiza glavnih komponenti

**(c)** RANSAC

**Slika 2.2:** Rezultati drugačijih metoda pronalska ravnine

# 3. Uklanjanje točaka oblaka koje reprezentiraju tlo

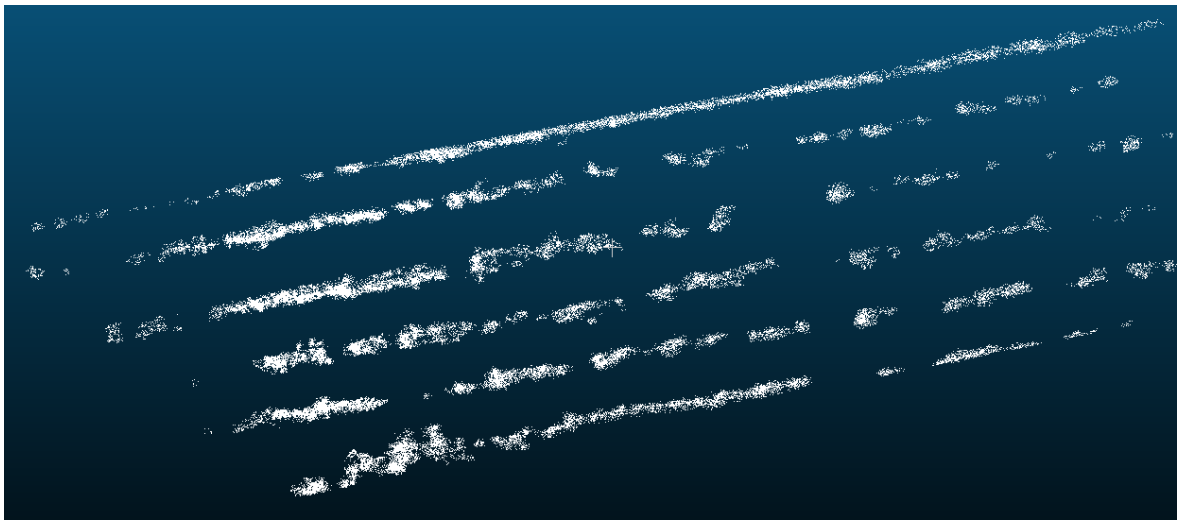
Jednom kad smo dobili rezultat paralelne ravnine poviše tla, možemo je koristit za uklanjanje tla. Konkretnije, uzeti ćemo presjek točaka na razini te ravnine, uz neko ekstra odstupanje, zbog toga što radimo s diskretnim skupom točaka. Ovaj dio je u programskom riješenju implementiran unutar modula *remove_ground.py* gdje se koristi funckija `remove_ground`, koja je zapravo *proxy* funkcije `get_points_above_plane` prikazane u isječku 3.

Jednostavnim matričnim računom se pronalazi vektorska udaljenosts od ravnine, shodno tome puni *numpy* polje, te se na kraju izbrišu dijelovi polja koji su višak (ovo se radi da bi se izbjeglo stvaranje novih polja u petlji, pošto znamo maksimalnu moguću veličinu polja).

```
def get_points_above_plane(plane_normal, point_on_plane, point_cloud,
↪  max_distance=.5):
    plane_slice = np.empty((len(point_cloud), 3))

    points_above_plane_counter = 0
    for point in point_cloud:
        p_ = point - point_on_plane

        dist_to_plane = np.dot(p_, plane_normal)
        # dist_to_plane is vector distance <-inf, +inf>
        if 0 < dist_to_plane < max_distance:
            plane_slice[points_above_plane_counter] = point
            points_above_plane_counter += 1

    plane_slice = np.delete(plane_slice,
    ↪  slice(points_above_plane_counter, len(plane_slice)), axis=0)

    return plane_slice
```

**Isječak 3:** Stvaranje presjeka oblaka točaka pomoću ravnine

Rezultat ovog modula je prikazan na slici 3.1. Ovdje se još uvijek radi o oblaku točaka u

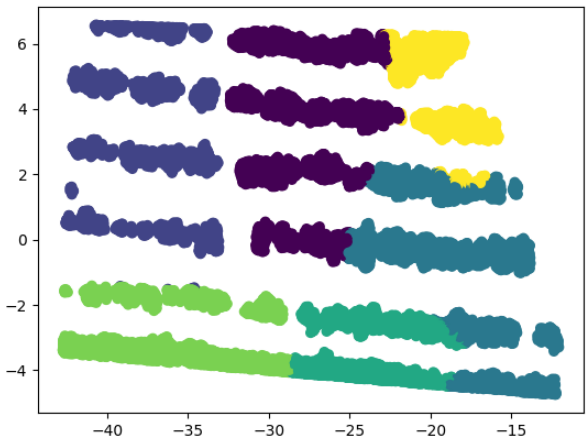


**Slika 3.1:** Presjek oblaka točaka samo s trsovima

3D, ali se to lako pretvori u 2D brišući informaciju o z-koordinati.

# 4. Klasifikacija točaka oblaka u trsove

Kao što je već napisano, trebamo pronaći način da programsko riješenje poveže točke u oblaku u skup trsova. Iz tih informacija kasnije ćemo lagano pronaći ulaze u samo redove. Prva ideja klasifikacijama su dva algoritma: grupiranje metodom k srednjih vrijednosti, te grupiranje modelom miješane gustoće (MMG). Pošto su redovi izduženog oblika, to čini grupiranje MMG-om kao potecijalno dobro riješenje. Nažalost pokušaji takvog riješenja ne daju grupiranja uz duž redova, već pronalaze centroide te pridružene normalne distribucije koje minimiziraju grešku na drugačiji način. Primjer jednog od takvih riješenja je prikazan na slici 4.1. Jedna od stvari koja se da primjetiti iz ovakvog rezultata: umjesto da pustimo MMG-u da pronađe izdužene distibucije za svaki od trsova, možemo mi biti ti koji ćemo direktno pronaći razdiobu. Može se primjetiti da su trsovi pozdoženi uzduž jedne osi. Stoga ćemo našu klasifikaciju obaviti jednostavno uzduž te osi, u ovom slučaju y osi. No, to neće biti moguće dok god redovi nisu idealno okomiti na y os. To nas dovodi do sljedećeg koraka.




**Slika 4.1:** Pokušaj klasifikacije modelom miješane gustoće

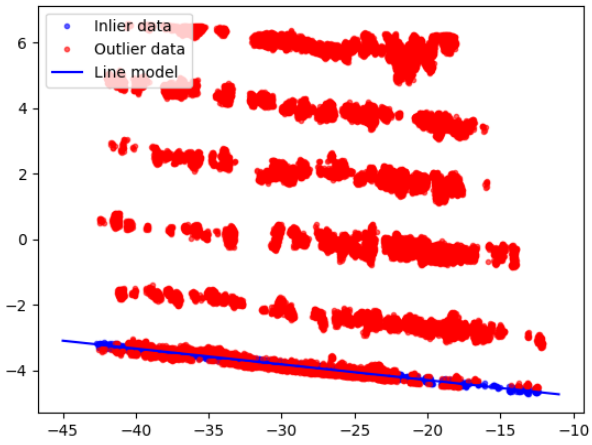
## 4.1. Određivanje nagiba redova vinograda

Da bi odredili kako poravnati vinograd da je okomit s y osi, treba prvo odrediti koji je nagib redova. Uz pretpostavku da su redovi međusobno paralelni, treba odrediti nagib samo jednog reda. U programskom riješenju to se radi u modulu *find_vineyard_angle_ransac.py*. Kao što sam naziv modula to nagoviještava, korisiti će se još jednom RANSAC. Razlika je što se ovdje neće tražiti ravnina (opet bi riješenje bilo ravnina paralelna s tlom), već pravac. Korisiti će se funkcija `ransac` iz `skimage.measure` uz definiran `LineModelND` kao model kojem se `ransac` treba prilagoditi. Dio programskog riješenja koji to radi je prikazan u isječku 4.

```
from skimage.measure import ransac, LineModelND

def find_vineyard_angle_ransac(point_cloud_data, residual_threshold=0.1,
↪  max_trials=500):
    point_cloud_2d = util.remove_z_axis(point_cloud_data)

    model, _ = ransac(point_cloud_2d, LineModelND, min_samples=2,
                      residual_threshold=residual_threshold,
                      ↪  max_trials=max_trials)

    origin, direction = model.params
    return origin, direction
```

**Isječak 4:** Pronalazak pravca koji definira nagib redova vinograda

Primjena ovog koda na primjeru koji se provlači kroz ovaj dokument vidi se na slici 4.2.




**Slika 4.2:** Pronađen nagib redova vinograda

## 4.2. Rotacija vinograda

Jednom kad smo dobili informacije o nagibu, možemo cijelu sliku rotirati tako da pronađeni pravac bude okomit na y os. To se obavlja u modulu *rotate_point_cloud.py* pomoću funkcije `level_off_point_cloud`, koja kao rezultat daje vinograd s osima okomitim na y os, i izbačenim podacima o z osi (pretvarajući oblak podataka i eksplicitno u 2D).

## 4.3. Klasifikacija trsova

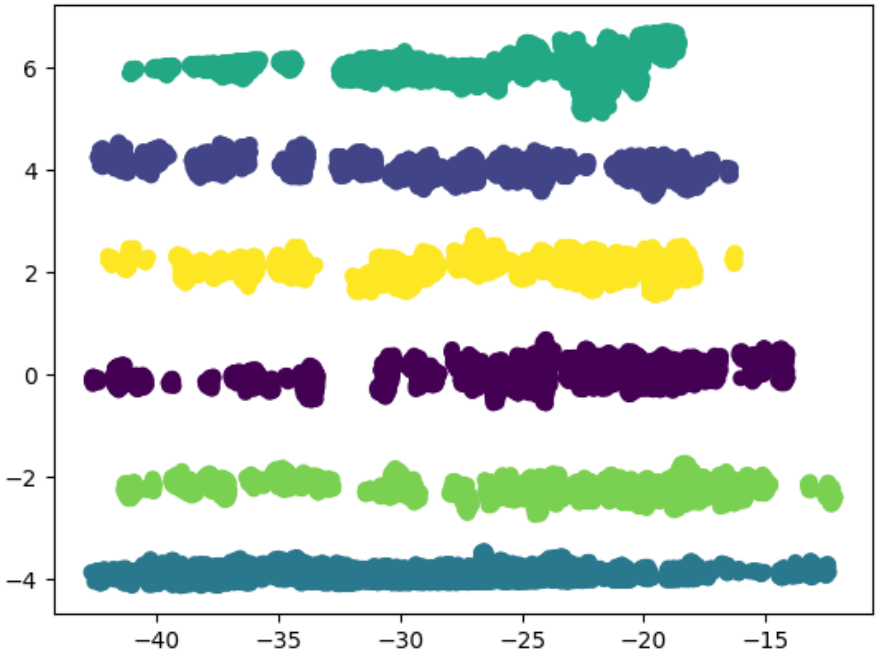
Sada kada su redovi okomiti, lagano se primjeni grupiranje metodom k srednjih vrijednosti. Treba napomenuti da se klasifikacija u ovom slučaju izvodi isključivo s jednom značajkom, a to je lokacija točke na y osi. Ovaj dio programa se odvija u modulu *group_vineyard_rows_KMeans* unutar funkcije `group_vineyard_rows` prikazane u isječku 5. Za izvođenje samog grupiranja se koristi klasa `sklearn.cluster.KMeans`.

```
from sklearn.cluster import KMeans

def group_vineyard_rows(point_cloud_data, nr_of_groups):
    classifier = KMeans(nr_of_groups).fit(point_cloud_data[:,
    ↪  1].reshape(-1, 1))

    y = classifier.predict(point_cloud_data[:, 1].reshape(-1, 1))

    return point_cloud_data, y
```

**Isječak 5:** Pronalazak pravca koji definira nagib redova vinograda

Iz argumenata funkcije je vidljivo drugo ograničenje ovog programskog riješenja: treba unaprijed znati broj redaka trsova, kako ga nigdje do sada nismo programski otkrili. To ne bi trebao biti veći problem kako vinogradari tu informaciju uglavnom znaju za svaki svoj vinograd. Ipak, postoje riješenja koja služe otkrivanju idealnog broja klasa, te bi se mogla implementirati u sklopu ovog programskog riješenja.

Rezultat ovog koda nad primjerom iz ovog poglavlja je vidljiv na slici 4.3.




**Slika 4.3:** Grupirani redci trsova

# 5. Pronalazak ulaza između redova trsova

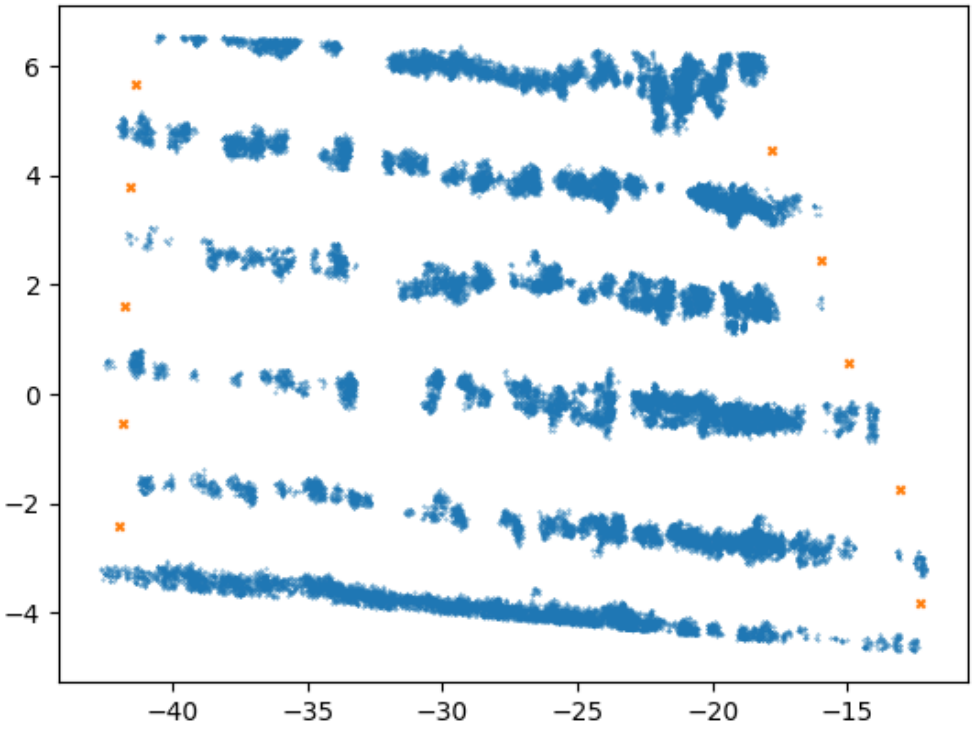
Sada kada konačno imamo definirane redove trsova, možemo pronaći i ulaze i izlaze između njih. Krećući redom, znamo odmah da će samih ulaza i izlaza biti $2 \cdot (broj\_redaka\_trsova - 1)$. Oduzimamo 1 jer zadnji red dijeli svoje ulaze i izlaze s predzadnjim, dok množimo dva za svaki ulaz i izlaz.

Same ulaze/izlaze možemo definirati na različite načine, ali u ovom programskom rješenju su uzete rubne točke svakog reda, te zatim za dva susjedna reda je pronađena točka koja je na pola puta između ta dva ekstremiteta. Jedna od opcija je da su uzme i najudaljenija točka po x osi, dok se za y os uvijek uzme prosječna vrijednost, ali to ostaje na odabir korisniku.

Ova funkcionalnost je implementirana u modulu *find_entrances_from_grouped_vineyard_rows.py* unutar istoimene funkcije. Funkcija za svaku od grupa koristi istu `ransac` metodu iz isječka 4 da bi degradirala skup točaka jednog trsa u jednu crtu. Zatim iz te crte uzima krajnje dvije točke i time definira dužinu od jedne do druge točke. Te točke su zatim sortiranje po y osi, da bi dobili niz dužina koje su sve međusobno susjedne. Nakon toga se obavlja u prosjećivanje ekstremiteta susjednih točaka kako bi se dobili ulazi/izlazi, što je u konačnici i željeni rezultat.

Na kraju ne smijemo zaboraviti primjeniti inverzu transformaciju onoj koju smo prije napravili da bi lakše grupirali trsove. Nakon te transformacije, dobivamo konačno riješenje ovog problema, prikazano na slici 5.1. [1]

[1] Sav kod ovog riješenja se može pronaći na github.com/marko-curlin/PROJEKT




**Slika 5.1:** Označeni ulazi i izlazi između redaka trsova

# LITERATURA


[1] Qian-Yi Zhou, Jaesik Park i Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". *arXiv:1801.09847* (2018.).

[2] Charles R. Harris i dr. "Array programming with NumPy". *Nature* 585.7825 (rujan 2020.), str. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[3] F. Pedregosa i dr. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011.), str. 2825–2830.

[4] Greg Turk. *Specifikacija PLY formata*. 1998. URL: https://people.math.sc.edu/Burkardt/data/ply/ply.txt.

[5] R.B. Rusu i S. Cousins. "3D is here: Point Cloud Library (PCL)". *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. Svibanj 2011., str. 1–4. DOI: 10.1109/ICRA.2011.5980567. URL: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5980567&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5980567.