Elektrotehnički fakultet
Univerzitet u Sarajevu
Odsjek za automatiku i elektroniku

Mapiranje terena pomoću Xmachines X100 vozilice

Studenti:

Admir Ahmespahić
Nejla Buljina
Alen Hrbat

Kolegij: Mehatronika

Nastavnik: doc.dr.sc. Jasmin Velagić

Akademska godina: 2023/24.

Sažetak:

U ovom radu je opisan postupak izrade i način upotrebe programskog rješenja za analizu podataka prikupljenih sa lidar senzora. Program omogućava korisniku snimanje tačaka u koordinatnom sistemu neovisnom od položaja vozilice, njihovu vizualizaciju, izoliranje stacionarnih prepreka sa podesivnom osjetljivošću, te snimanje i ponovni unos tačaka u xml formatu za kasniju obradu ili obradu u drugim programskim rješenjima. Korisniku je također omogućeno kretanje kroz 3d scenu oko koordinatnog početka.

Ključne riječi:

lidar senzor, Xmachines X100, mapiranje, stacionarne i dinamičke prepreke, ROS, internet socket, OpenGL, xml

Sarajevo, Juli 2024.

Sadržaj

1 Opis zadatka

Zadatak ovog projekta je omogućavanje akvizicije podataka sa lidar senzora ugrađenog na Xmachines X100 vozilicu, te upotreba podataka za mapiranje statičkih prepreka u okolini robota. Pri tome treba voditi računa o sljedećim zahtjevima:

- Vozilica se tokom mapiranja može kretati, što treba uzeti u obzir prilikom određivanja apsolutne pozicije tačke.
- Iscrtavanje mape statičkih prepreka se treba obavljati u realnom vremenu.
- Korisnik mora imati mogućnost da jasno vidi trenutno mapirane tačke.
- Statičke tačke koje su detektovane se moraju zapamtiti iako više nisu vidljive za senzor.
- Mora biti omogućena pohrana tačaka na računar.

Vozilica i lidar senzor su bazirani na ROS okruženju.

2 Korištena oprema

Za realizaciju je korištena vozilica Xmachines X100 sa odgovarajućim lidar senzorom. Čitanje podataka sa ROS-a je izvršeno sa računarom na sistemu *Ubuntu 20.04* i instaliranim *ROS Noetic* programskim rješenjem. Obrada podataka i crtanje je izvršeno pomoću računara sa *Windows 10* operativnim sistemom i instaliranim *natID* framework-om. Svi uređaju moraju biti povezani na zajedničku mrežu. U nastavku je opisana njihova konfiguracija:

2.1 Konfiguracija opreme

2.1.1 Vozilica

Vozilicu je prvo potrebno spojiti na odgovarajuću mrežu. Zatim je potrebno provjeriti IP adresu korištenjem komande $ifconfig$ u terminalu. Na vozilici se kroz tri različita terminala pokreću sljedeći ROS čvorovi:

```
POZICIJA VOZILICE:
roslaunch laser_scan_matcher demo.launch
```

```
SNIMLJENE TAČKE
roslaunch velodyne_pointcloud VLP16_points.launch

KONTROLA VOZILICE:
roslaunch x100_base x100_base.launch
```

Za dodatne informacije, moguće je u terminalu pokrenuti komandu $rostopiclist$ da bi se dobile sve otvorene teme. Njihovi podaci i funkcionalnosti se mogu nać u ROS dokumentaciji.

2.1.2 Računar za akviziciju

Na računaru ua akviziciju podatak je potrebno da je instaliran operativni sistem Ubuntu ili instalirati aplikaciju koja simulira dati operativni sistem. U projektu je korištena apliakcija Oracle VM VirtualBox Manager koja na jendostavana način emulira Ubuntu sistem na Windows operativnom sistemu. Nakon konfiguracije Ubuntu sistema i skidanja potrebnih biblioteka pokrenut je napravljena je skripta za primanje informacija.

2.1.3 Računar za obradu

Potrebno je izvršiti build programa korištenjem CMake-a prije prvog pokretanja. Pri pokretanju programa automatski se otvara socket na portu 44201 i IP adresi računara koja se može naći upotrebom komande $ipconfig$ u Command Prompt-u. Ako client izgubi konekciju sa serverom potrebno je ponovno pokrenuti aplikaciju.

3 Opis rješenja

Rješenje problema se može podijeliti na dva dijela:

- Akvizicija podataka sa ROS sistema i njihovo prosljeđivanje na obradu u realnom vremenu (ROS[1] i python)
- Obrada podataka, iscrtavanje i filtriranje (C++[2] i OpenGL[3][4])

Komunikacija između ova dva dijela je ostvarena preko internet socket-a[5][6] upotrebom TCP protokola.

3.1 Akvizicija podataka

Prvo je bilo potrebno stvoriti datoteku i unutar nje python skriptu. kao IDE korišten je Studio Visual Code. Unutar skripte je potrebno pretplaitti se na odgovarajuću temu. Pretplaćivanje se izvrši pomoću Subscriber-a koji zahtjeva sljedeće parametre: ime teme, tip varijable koju prima i naziv funkcije koja se poziva za pirmljene podatke. Bile su potrebna dva SUbscriber-a i 2 teme. Jedno za obradu detektovanih tačaka okruženja od senzora, a drugo za poziciju vozilice.
Tema Subscruber-a za detekotvanje tačkaka je glasila: " `/velodyne_points`", a tip varijable je bio PointCloud2. Funkcija se zvala Callback i bila je zadužena za raspakivanje tih tačaka položaja na x, z, y komponente i slanja infromacija u formatu koji server može pročitati, odnosno u string varijabli. Server je već ranije bio pokrenut na računaru za obradu informacija. Tema Subscruber-a za detekotvanje pozicije vozilice je glasila: "posed2D", a tip varijable je bio Pose2D.Funkcija se zvala `Callback_pozicija` i bila je zadužena za primanje i slanje pozicje vozilice na serveru čitljiv način, odnosno u string varijabli.
Prije primanja informacija računar za akviziciju se povezivao sa serverom koji je bio pokrenut na računaru za ibradu pomoću socket biblioteke. Korištene su još biblioteke za sve pomenute varijable, kao i biblioteka za ROS sistem.

3.2 Obrada podataka

Za obradu podataka je korišten C++ programski jezik uz upotrebu OpenGL grafičke biblioteke i natID framework-a. Program se sastoji od dva procesa koji se izvršavaju paralelno:

- Pokretanje TCP servera i osvježavanje sadržaja dobijenih poruka
- Interpretacija poruka i crtanje tačaka

Ova dva procesa dijele dva podatka tipa string naziva $\_message$ i $\_message1$ koji u sebi sadrže pozicije tačaka i vozilice respektivno. Kako dva procesa dijele istu memoriju, neophodno je korištenje mutex-a (mutual exclusion lock) da bi se izbjegao deadlock ili neispravno stanje poruke. Poruke su formatirane na sljedeći način:

```
_message:
x2.55 y-2.25 z-0.54 x1.51 y-1.34 z-0.54 x4.24...
_message1:
```

```
a-0.2946369297381833 b0.004793161592389029
```

Da bi se ispunili real-time zahtjevi, bilo je neophodno smanjiti preciznost snimljenih tačaka. Sa preciznosti od 2 decimale je zadržano dovoljno podataka o snimljenim tačkama za ovu upotrebu. Svaki broj koji se nalazi u porukama prije vrijednosti ima oznaku tipa:

- x -> x koordinata snimljene tačke
- y -> y koordinata snimljene tačke
- z -> z koordinata snimljene tačke
- a -> x koordinata pozicije vozilice
- b -> y koordinata pozicije vozilice

Drugi proces vrši obradu tačaka i njihovo iscrtavanje na ekran. Prije iscrtavanja nove scene obavlja se funkcija $addPoints$ koja dodaje nove tačke u vektor tačaka i sadrži svu logiku za odabir tačaka i filtriranje. Funkcija ima sljedeći oblik:

```
1        void addPoints() {
2    //citanje nove pozicije vozilice
3        td::String origin = getMessage1();
4        std::stringstream aa(origin.c_str());
5        char cc;
6        aa >> cc >> originx >> cc >> originy;
7        iteracije++;
8
9    //inicijalizacija buffera
10       _gpuBuffer.reset();
11       _gpuBuffer.init(64, 100, 100, { gui::gl::DataType::vec3});
12       pMtxSetterCmd = _gpuBuffer.createCommand();
13       pMtxSetterCmd->createMVPSetter(&_mvpMatrix);
14       _gpuBuffer.encode(pMtxSetterCmd);
15
16   //citanje novih tacaka
17       td::String tackepom = getMessage();
18       //if (tackepom == "") return;
19       std::string tacke = tackepom.c_str();
```

```
20        std::stringstream ss(tacke);
21
22        float x, y, z;
23        char c;
24
25        while (1) {
26            ss >> c;
27      //citaj dok se ne dobije x koordinata
28            while (c != 'x') {
29
30                ss >> c;
31                if (ss.eof()) break;
32            }
33            ss >> x;
34            if (ss.eof()) break;
35            ss >> c;
36      //citaj dok se ne dobije y koordinata
37            while (c != 'y') {
38                ss >> c;
39                if (ss.eof()) break;
40            }
41            ss >> y;
42            if (ss.eof()) break;
43            ss >> c;
44      //citaj dok se ne dobije z koordinata
45            while (c != 'z') {
46                ss >> c;
47                if (ss.eof()) break;
48            }
49            ss >> z;
50            if (ss.eof()) break;
51
52      //dodaj poziciju vozilice na x i y koordinate
53            x = x + originx;
54            y = y + originy;
55            bool found = false;
56
```

```
57      //pokusaj naci tacku koja je blizu, ako je nema dodaj novu tacku
58            for (int i = 0; i < vec.size(); i += 3) {
59                if (abs(x - vec[i]) <= EPS && abs(y - vec[i + 1]) <= EPS && abs(
    z - vec[i + 2]) <= EPS) {
60                    repeat[i/3]++;
61                    lastv[i / 3] = true;
62                    found = true;
63                    break;
64                }
65            }
66            if(!found) {
67                vec.emplace_back(x);
68                vec.emplace_back(y);
69                vec.emplace_back(z);
70                repeat.push_back(0);
71                lastv.push_back(true);
72            }
73
74      //dopuni do tacke (fail safe)
75            while (vec.size() % 3 != 0) {
76                vec.emplace_back(0);
77            }
78        }
79
80    //ako se tacka nije pojavila granica puta i
81    //nije se pojavila u ovoj iteraciji, izbrisi iteracije
82        for (int i = 0; i < repeat.size(); i++) {
83            if (repeat[i] >= granica) continue;
84            if (!lastv[i]) repeat[i] = 0;
85            lastv[i] = false;
86        }
87    //dinamicki alociraj niz i ubaci sve staticke tacke
88        float* niz = new float[vec.size()];
89        int ind = 0;
90        for (int i = 0; i < vec.size(); i+=3) {
91            if (repeat[i/3] >= granica) {
92                niz[ind]=vec[i];
```

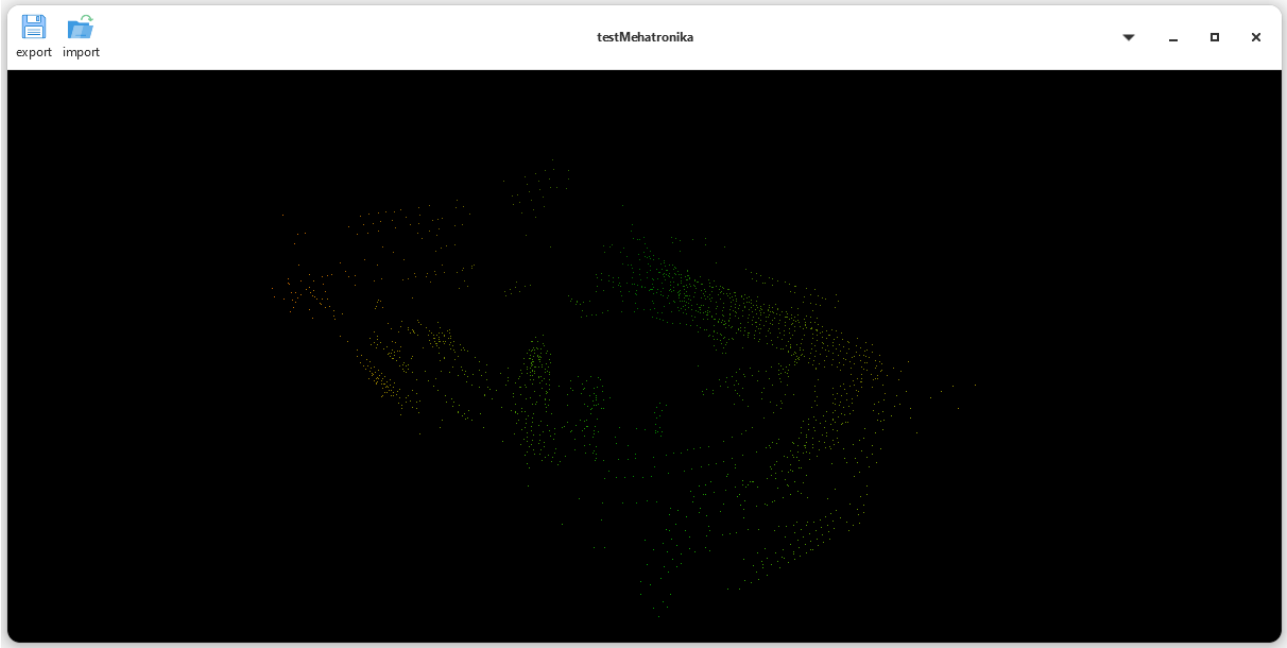
```
93                  niz[ind + 1] = vec[i + 1];
94                  niz[ind + 2] = vec[i + 2];
95                  ind += 3;
96              }
97          }
98
99          nVertices = ind/3;
100     //dodavanje tacaka u buffer
101         _gpuBuffer.appendVertices(niz, nVertices);
102
103         delete[] niz;
104
105     //komanda za crtanje tacaka
106         pCubeTextureCmd = _gpuBuffer.createCommand();
107         pCubeTextureCmd->createDrawArrays(gui::gl::Primitive::Points, 0,
    nVertices);
108         _gpuBuffer.encode(pCubeTextureCmd);
109         if (!_gpuBuffer.commit())
110         {
111             mu::dbgLog("ERROR! Cannot commit buffer to GPU");
112             return;
113         }
114
115         _program.setBuffer(&_gpuBuffer);
```

Za kod su korištene sljedeće varijable koje se čuvaju i nakon završetka iteracije:

```
std::vector<float> vec;
std::vector<int>repeat;
std::vector<bool>lastv;
float EPS = 0.06;
int granica = 50;
```

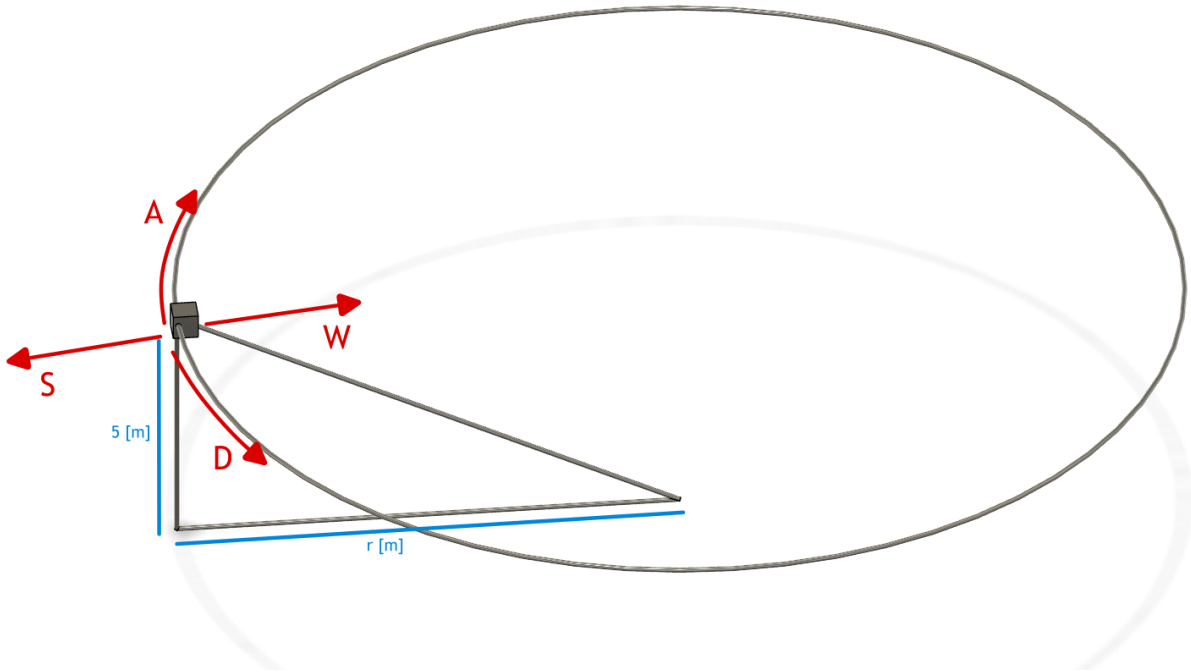
vec čuva sve snimljene tačke u formatu $x, y, z, x, y...$ $repeat$ pamti koliko puta se koja tačka ponovila (podatku $repeat[i]$ odgovara tačka sa koordinatama $vec[3 * i], vec[3 * i + 1], vec[3 * i + 2]$). $lastv$ pamti da li je u zadnjoj iteraciji detektovana tačka (podatku $lastv[i]$ odgovara tačka sa koordinatama $vec[3 * i], vec[3 * i + 1], vec[3 * i + 2]$). EPS označava koliko daleko po bilo kojoj koordinati smije

biti udaljena snimljena tačka od prethodno snimljene tačke da bi se smatrala jednakom. $granica$ je broj uzastopnih ponavljanja tačke koja se moraju desiti da bi se tačka smatrala statičkom. Zadnja dva parametra je moguće modifikovati da bi se postigla drugaćija preciznost i brzina izvršavanja koda. Na slici 3.1 se mogu vidjeti iscrtane tačke.



Slika 3.1: Prikaz iscrtanih tačaka

Eksperimentalno je pokazano da je sa ovim parametrima sistemu potrebno da tačka miruje između 6 i 10 sekundi da bi se smatrala statičkom. Vrijeme može varirati u zavisnosti od udaljenosti tačke od senzora (što je veća udaljenost, više vremena je potrebno da se tačka detektuje). Korisniku je radi lakše vizualizacije mape na raspolaganju i mogućnost pomjeranja kamere. kamera se može okretati oko tačke $(0, 0, 0$ u pozitivnom i negativnom smjeru pritiskom na tipke a i d po kružnici čiji se radius može podešavati tipkama w i s (slika 3.2).



Slika 3.2: Pozicija i upotreba kamere

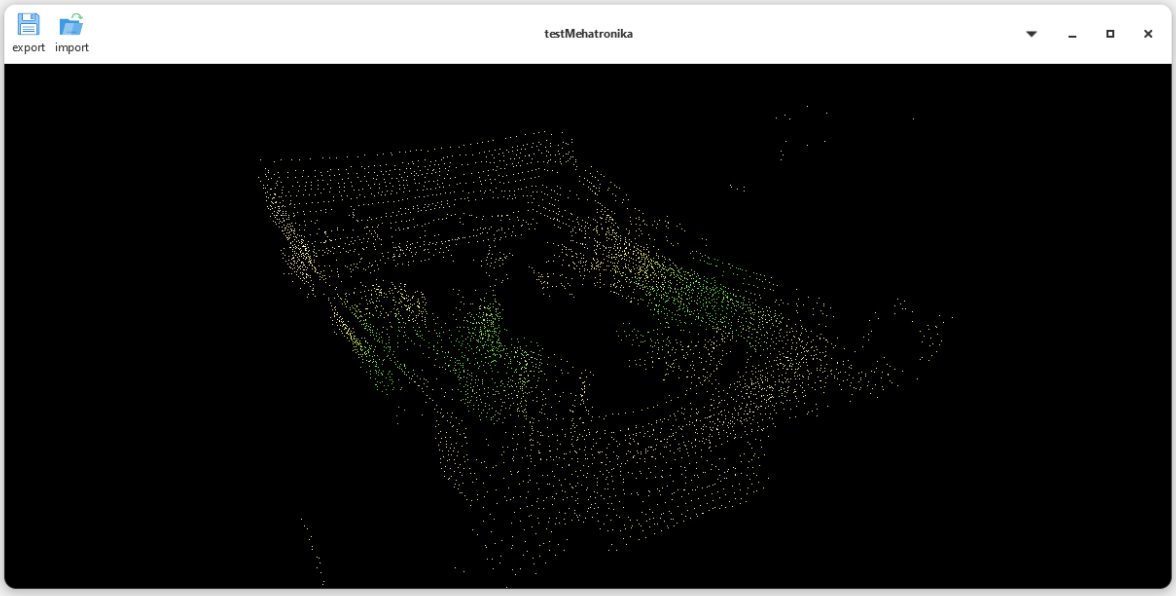
Da bi se dodatno olakšala vizualizacija, napisan je shader program koji boji svaku tačku u zavisnosti od njene udaljenosti od početne tačke vozilice (koordinatnog početka). Tačke bliže koordinatnom početku su obojene zelenom bojom, dok udaljenije polako prelaze u crvenu boju.

4 Post-processing

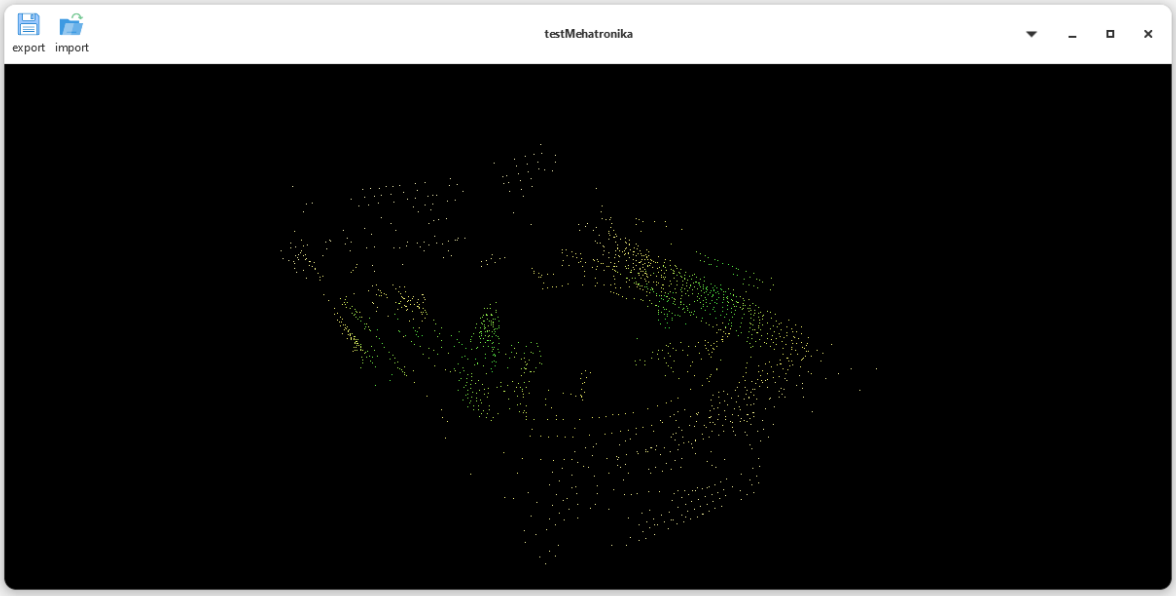
Pored analize tačaka za vrijeme mapiranja, moguće je izvršiti snimanje istih korištenjem opcije *export*. Te se tačke kasnije mogu ponovno učitati korištenjem funkcije *import*. Pohrana tačaka se vrši u xml formatu, što omogućava mnogo brže učitavanje u program nego samo spašavanje skeniranih tačaka za vrijeme slanja sa ROS-a. Xml fajla ima sljedeći format:

```
 <Points>
<Point x="0.294088" y="-1.73505" z="-0.52" weight="58"/>
<Point x="0.304088" y="-1.75505" z="-0.17" weight="138"/>
<Point x="0.234088" y="-1.63505" z="0.16" weight="139"/>
       ...
 </Points>
```

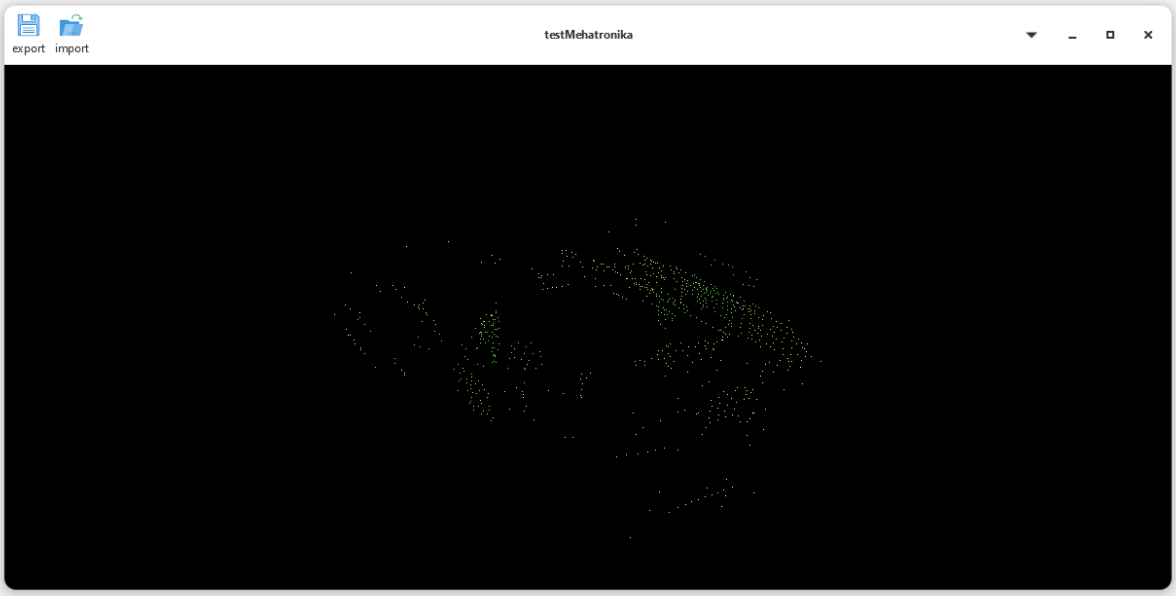
Ovaj pristup također omogućava prikaz tačaka za različitu graničnu vrijednost ponavljanja. Primjer toga možemo vidjeti na slikama 4.1, 4.2 i 4.3.



Slika 4.1: Snimak za granica=10



Slika 4.2: Snimak za granica=50



Slika 4.3: Snimak za granica=90

5 Zaključak i diskusija

Kroz izradu projektnog zadatka uspješno je napisan program koji omogućava korisniku da pregleda, sačuva i manipuliše podacima dobijenim sa lidar senzora za mapiranje. Uz ovaj dokument se prilaže i sljedeći drive link:

`https://drive.google.com/drive/folders/1P0Zy7e6Eg5bnusfRZo-studrwRMFwKAy?usp=sharing`

na kojem je dostupan video snimak funkcionalnosti programa, izvorni kod i snimljeni podaci. Također je uspješno implementiran filter dinamičkih prepreka koji omogućuje da se svaka prepreka snima nakon bar 6 sekundi prisustva u blizini robota. Ovaj filter radi dovoljno dobro za svrhu demonstracije programa, ali se može dodatno poboljšati primjenom statističkih metoda eliminacije dinamičkih tačaka. Dio zadatka koji nije uspješno riješen je iscrtavanje površine na osnovu dobijenih tačaka. Ovaj problem se pokušao riješiti upotrebom *Ball Pivoting* algoritma koji nije ponudio poželjne rezultate.

Literatura

[1] ROS Noetic Documentation, dostupno na: https://wiki.ros.org/noetic

[2] C++ reference, dostupno na: https://en.cppreference.com/w/

[3] OpenGL Documentation. The Khronos Group Inc., dostupno na: https://www.khronos.org/opengl/

[4] Ginsburg, D., Purnomo, B., OpenGL ES 3.0 Programming Guide, 2014, dostupno na: https://www.rose-hulman.edu/class/csse/csse351/reference/OpenGL_ES_3.0_Programming_Guide.pdf

[5] Getting started with Winsock. Microsoft corp., dostupno na: https://learn.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock

[6] socket — Low-level networking interface. Python Software Foundation, dostupno na: https://docs.python.org/3/library/socket.html