SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

MRAČNO BODOV

ZADANIE

2024 **Bc. Maroš Kocúr, Bc. Eduard Zelenay**

Obsah

Zoznam obrázkov a tabuliek

Úvod

Cieľom zadania je oboznámiť sa s prácou s mračnom bodov (point cloud) a segmentácia objektov v priestore. V zadaní si vyskúšame vytvorenie vlastného mračna bodov a aplikáciu metód na získanie segmentovaného priestoru.

1 Úlohy

1. Vytvorenie mračna bodov pomocou Kinect v2 pre testovanie. Nájdite online na webe mračno bodov popisujúce väčší priestor (väčší objem dát aspoň 4x4 metre) pre testovanie algoritmov a načítajte mračno dostupného datasetu.

2. Pomocou knižnice (open3d - python) načítate vytvorené mračno bodov a zobrazíte.

3. Mračná bodov očistite od okrajových bodov. Pre túto úlohu je vhodné použiť algoritmus RANSAC.

4. Segmentujte priestor do klastrov pomocou vhodne zvolených algoritmov (K-means, DBSCAN, BIRCH, Gausian mixture, mean shift ...). Treba si zvoliť aspoň 2 algoritmy a porovnať ich výsledky

5. Detailne vysvetlite fungovanie zvolených algoritmov. (Keďže neimplementujete konkrétny algoritmus ale používate funkcie tretích strán je potrebné rozumieť aj ako sú funkcie implementované)

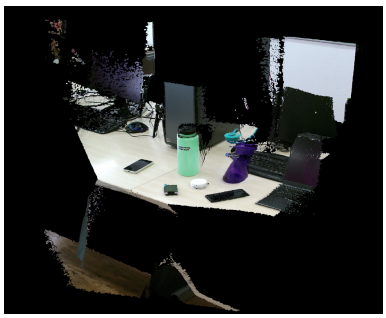
6. Vytvorte dokumentáciu zadania:

    - popis implementovaných algoritmov,
    - grafické porovnanie výstupov,
    - vysvetlite rozdiel v kvalite výstupov pre rozdielne typy algoritmov.

2 Vytvorenie a získanie mračna bodov

2.1 Mračno bodov - Kinect

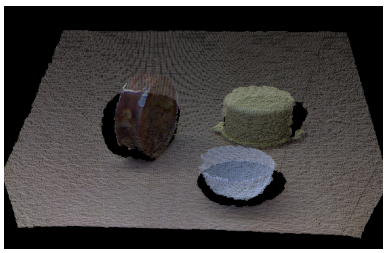
Pomocou tutoriálu [1] sa nám podarilo rozbehať Kinect na Linuxe. Pomocou python scriptu *dump_pcd.py* sme vytvorili mračno bodov, ktoré môžeme vidieť na 2.1



Obr. 2.1: Mračno bodov z Kinectu

2.2 Mračno bodov - online

Na internete sa nachádza obrovské množstvo mračien bodov, z ktorých niektoré sú platené, naopak, iné (tie ktoré sa nám páčia viac) sú zadarmo. Podarilo sa nám spomedzi obrovského kvanta mračien bodov nájsť, respektíve vybrať si, jedno, ktoré je využiteľné pre naše účely. Jeho vyobrazenie môžeme vidieť na 2.2



Obr. 2.2: Mračno bodov z internetu

3 Načítanie a zobrazenie mračna bodov

Načítanie a zobrazenie mračna bodov sme vykonali za pomoci voľne dostupnej knižnice Point Cloud Library. Táto knižnica je napísaná v jazyku C++.[2]

Samotné načítanie môžeme vidieť v podsekcii 3.1, pričom zobrazenie v podsekcii 3.2.

3.1 Načítanie

```
pcl::PointCloud <pcl::PointXYZRGB>::Ptr cloud
      (new pcl::PointCloud <pcl::PointXYZRGB>);
if ( pcl::io::loadPCDFile <pcl::PointXYZRGB>
      ("../learn34.pcd", *cloud) == -1 )
{
  std::cout << "Cloud reading failed." << std::endl;
  return (-1);
}
```

3.2 Zobrazenie

```
pcl::visualization::CloudViewer viewer ("Cluster viewer");
viewer.showCloud (colored_cloud);
while (!viewer.wasStopped ())
{
  std::this_thread::sleep_for(100us);
}
```

4 Očistenie od okrajových bodov

Mračná bodov sme očistili od okrajových bodov prostredníctvom algoritmu RANSAC, ktorého implementácia sa nachádza už v predom spomínanej knižnici Point Cloud Library. [2]

Jednoduché vysvetlenie algoritmu je popísané v podsekciách 4.1 a 4.2.

4.1 RANSAC

RANSAC je algoritmus vytvorený Fischlerom a Bollesom, určuje všeobecný prístup k odhadu parametrov, s veľkým podielom outliers v stupnom datasete. Na rozdiel od iných výkonných algoritmov odhadu, ako napríklad M-odhady a metódou najmenších štvorcov s prepojením na strojové učenie. RANSAC bol vytváraný komunitov ľudi používajúci strojové učenie. RANSAC je vzorkovacia technika ktorá generuje kandidáta na minimálny počet pozorovaní potrebných na zistenie odhadu parametrov leziacich pod modelom. Narozdiel od ostatných vzorkovacích algoritmov, ktoré používajú čo najviac bodov ako môžu, RANSAC používa najmenší počet bodov ako môže.[3]

4.2 RANSAC - Algoritmus

1. Vybranie minimum náhodných bodov potrebných na určenie parametrov modelu
2. Vypočítanie parametrov pre model
3. Určenie koľko bodov z množiny všetkých bodov leží v preddefinovanom prahovom rozsahu.
4. opakuj kroky 1 až 3, s maximálnym N opakovaniami.
5. Vráti model s najväčším počtom bodov ležiacich v prahovej hodnote. [3]

4.3 Implementácia

```
pass.setInputCloud (cloud);
pass.setFilterFieldName ("z");
pass.setFilterLimits (0, 1.2);
pass.filter (*cloud_filtered);
```

V tejto časti kódu sme nastavili a aplikovali priechodný filter na mračno bodov. Tým sme ho teda očistili od okrajových bodov.

```
seg.setOptimizeCoefficients (true);
seg.setModelType (pcl::SACMODEL_NORMAL_PLANE);
seg.setNormalDistanceWeight (0.1);
seg.setMethodType (pcl::SAC_RANSAC);
seg.setMaxIterations (100);
seg.setDistanceThreshold (0.03);
seg.setInputCloud (cloud_filtered);
seg.setInputNormals (cloud_normals);
// Obtain the plane inliers and coefficients
seg.segment (*inliers_plane, *coefficients_plane);
std::cerr << "Plane coefficients: " << *coefficients_plane
      << std::endl;
```

Táto časť kódu konfiguruje a využíva algortimus RANSAC, pomocou ktorého je nájdená rovina.

5 Segmentácia

Na segmentáciu priestoru klastrov sme si zvolili dva algoritmy - Region growing segmentation (Segmentácia rastúceho regiónu) a Euclidean cluster extraction (Euklidovská extrakcia klastrov). Pri prvom z nich sme sa rozhodli vyskúšať a zároveň porovnať aj algoritmus odvodený od pôvodného algoritmu rastúceho regiónu, ktorým bol algoritmus Color-based region growing segmentation (Segmentácia rastúceho regiónu na základe farieb).

5.1 Segmentácia rastúceho regiónu

Segmentačný algoritmus rastúceho regiónu je metóda používaná pri spracovaní mračna bodov, obzvlášť výhodná na segmentovanie mračna bodov do viacerých oblastí. Účelom algoritmu je zlúčiť body, ktoré sú dostatočne blízko z hľadiska obmedzenia hladkosti. Výstupom tohto algoritmu je teda množina zhlukov, kde každý zhluk je množinou bodov, ktoré sa považujú za časť rovnakého hladkého povrchu. Práca tohto algoritmu je založená na porovnávaní uhlov medzi bodovými normálami.

5.1.1 Algoritmus rastúceho regiónu

1. **Inicializácia**

   - Vstupom do algoritmu sú: mračno bodov, normály a zakrivenia bodov
   - Body v mračne sú spočiatku zoradené na základe ich hodnôt zakrivenia, pričom algoritmus začína od bodu minimálneho zakrivenia, ktorý zvyčajne označuje plochejšiu oblasť.

2. **Proces rastu regiónu**

   - Región začína rásť od bodu s najmenším zakrivením.
   - Tento počiatočný bod sa pridá k aktuálnemu regiónu a preskúmajú sa jeho susedné body.
   - Susedné body sa pridajú do oblasti, ak je uhol medzi ich normálami a normálou počiatku menší ako vopred definovaný prah (prah hladkosti).
   - Okrem toho, ak je zakrivenie suseda pod určitým prahom (prah zakrivenia), je tiež pridané do zoznamu počiatkov pre ďalšie rozšírenie regiónu.
   - Tento proces sa opakuje, kým do oblasti nemožno pridať ďalšie body.
   - Proces sa opakuje s novým počiatočným bodom (ďalší bod s minimálnym zakrivením v zostávajúcej skupine bodov), kým sa všetky body buď nepriradia k oblasti, alebo sa nezrušia.

3. **Ukončenie**

   - Algoritmus sa ukončí, keď sú všetky body buď priradené k regiónu, alebo už nezostávajú žiadne body na spracovanie.
   - Výsledkom je súbor zhlukov bodov, z ktorých každý predstavuje segment mračna bodov, ktorý sa považuje za súčasť rovnakého hladkého povrchu.

Podrobnejšie vysvetlenie nájdete na webovej stránke Point Cloud Library.[4]

5.1.2 Implementácia rastúceho regiónu

```
  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud
       (new pcl::PointCloud<pcl::PointXYZ>);
if ( pcl::io::loadPCDFile <pcl::PointXYZ>
       ("../learn34.pcd", *cloud) == -1)
{
  std::cout << "Cloud reading failed." << std::endl;
  return (-1);
}
```

Načítanie mračna bodov.

```
pcl::search::Search<pcl::PointXYZ>::Ptr tree
      (new pcl::search::KdTree<pcl::PointXYZ>);
pcl::PointCloud <pcl::Normal>::Ptr normals
      (new pcl::PointCloud <pcl::Normal>);
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> normal_estimator;
normal_estimator.setSearchMethod (tree);
normal_estimator.setInputCloud (cloud);
normal_estimator.setKSearch (50);
normal_estimator.compute (*normals);

pcl::IndicesPtr indices (new std::vector <int>);
pcl::removeNaNFromPointCloud(*cloud, *indices);
```

Vytvorenie stromovej štruktúry rozdeľujúcej priestor na organizáciu bodov v k-rozmernom priestore. Výpočet normál bodov, odstránenie neplatných bodov NaN.

```
pcl::RegionGrowing<pcl::PointXYZ, pcl::Normal> reg;
reg.setMinClusterSize (50);
reg.setMaxClusterSize (1000000);
reg.setSearchMethod (tree);
reg.setNumberOfNeighbours (30);
reg.setInputCloud (cloud);
reg.setIndices (indices);
reg.setInputNormals (normals);
reg.setSmoothnessThreshold (3.0 / 180.0 * M_PI);
reg.setCurvatureThreshold (1.0);
```

Nastavenie parametrov algoritmu.

```
std::vector <pcl::PointIndices> clusters;
reg.extract (clusters);
```

Extrakcia klastrov.

```
pcl::PointCloud <pcl::PointXYZRGB>::Ptr
      colored_cloud = reg.getColoredCloud ();
pcl::PCDWriter writer;
writer.write ("region_learn.pcd", *colored_cloud, false);
```

Uloženie segmentovaného mračna bodov.

5.2 Segmentácia rastúceho regiónu na základe farieb

Algoritmus rastúceho regiónu podľa farieb, je pokročilejší variant štandardného algoritmu rastúceho regiónu. Využíva informácie o farbách na vykonanie segmentácie mračien farebných bodov, pričom sa odlišuje tým, že sa zameriava skôr na atribúty farieb než na geometrické vlastnosti, ako sú normály alebo zakrivenie.

Rastúci región podľa farieb používa zlučovací algoritmus na kontrolu nadmernej a nedostatočnej segmentácie. Po segmentácii sa uskutoční pokus o zlúčenie zhlukov s blízkymi farbami. Dva susedné zhluky s malým rozdielom medzi priemernou farbou sú zlúčené. Potom sa uskutoční druhý krok spájania. Počas tohto kroku je každý jednotlivý klaster overený počtom bodov, ktoré obsahuje. Ak je toto číslo menšie ako hodnota definovaná používateľom, aktuálny klaster sa zlúči s najbližším susedným klastrom.

5.2.1 Algoritmus rastúceho regiónu na základe farieb

1. **Inicializácia**

    - Algoritmus vyžaduje ako vstup farebné mračno bodov, ktoré obsahuje priestorové súradnice aj farebné údaje RGB pre každý bod.

2. **Proces rastu regiónu**

    - Podobne ako pri štandardnom algoritme rastu regiónu, táto metóda iniciuje oblasti z počiatočných bodov. Výber týchto bodov však môže byť ovplyvnený ďalšími farebnými kritériami.
    - Regióny rastú začlenením susedných bodov, ktoré spĺňajú špecifické kritériá týkajúce sa farebnej vzdialenosti a priestorovej blízkosti.
    - Dva hlavné prahy definujú začlenenie bodov do regiónu:
        - **Prah farby bodu:** Určuje, ako blízko musí byť bod farebne k aktuálnemu zdroju, aby sa mohlo zvážiť jeho zahrnutie do aktuálnej oblasti.
        - **Prah vzdialenosti:** Definuje maximálnu priestorovú vzdialenosť pre bod, ktorý sa má považovať za suseda.

3. **Zlučovanie regiónov**

    - Po počiatočnom vytvorení regiónov sa začína proces spájania, ktorý rieši presegmentovanie:
        - **Prah farby oblasti:** Používa sa na zlúčenie susedných oblastí, ak sú ich priemerné farby dostatočne podobné.
        - **Prah minimálnej veľkosť klastra:** Ak je veľkosť regiónu pod týmto prahom, môže sa zlúčiť so susedným regiónom, aby sa zabezpečili zmysluplné výsledky segmentácie.

4. **Ukončenie**

    - Algoritmus sa ukončí, keď nie je možné pridať ďalšie body do žiadnej oblasti podľa definovaných kritérií a nie je možné žiadne ďalšie zlučovanie.

Podrobnejšie vysvetlenie nájdete na webovej stránke Point Cloud Library.[5]

5.2.2 Implementácia rastúceho regiónu na základe farieb

```
pcl::PCDWriter writer;
pcl::search::Search <pcl::PointXYZRGB>::Ptr tree
      (new pcl::search::KdTree<pcl::PointXYZRGB>);
pcl::PointCloud <pcl::PointXYZRGB>::Ptr cloud
      (new pcl::PointCloud <pcl::PointXYZRGB>);
if ( pcl::io::loadPCDFile <pcl::PointXYZRGB>
      ("../learn34.pcd", *cloud) == -1 )
{
  std::cout << "Cloud reading failed." << std::endl;
  return (-1);
}
pcl::IndicesPtr indices (new std::vector <int>);
pcl::removeNaNFromPointCloud (*cloud, *indices);
```

Načítanie mračna bodov, vytvorenie stromovej štruktúry a odstránenie neplatných bodov NaN.

```
pcl::RegionGrowingRGB<pcl::PointXYZRGB> reg;
reg.setInputCloud (cloud);
reg.setIndices (indices);
reg.setSearchMethod (tree);
reg.setDistanceThreshold (10);
reg.setPointColorThreshold (6);
reg.setRegionColorThreshold (5);
reg.setMinClusterSize (600);
```

Nastavenie parametrov algoritmu.

```
std::vector <pcl::PointIndices> clusters;
reg.extract (clusters);
```

Extrakcia klastrov.

```
pcl::PointCloud <pcl::PointXYZRGB>::Ptr
      colored_cloud = reg.getColoredCloud ();
  if (colored_cloud->points.empty ())
      std::cerr << "Can't find the cylindrical component."
              << std::endl;
```

```
else
{
        std::cerr << "PointCloud representing the component: "
        << colored_cloud->size () << " data points." << std::endl;
          writer.write ("region_learn.pcd", *colored_cloud, false);
}
```

Uloženie segmentovaného mračna bodov.

5.3 Euklidovská extrakcia klastrov

Euklidovská extrakcia je metóda na segmentovanie neorganizovaných mračien bodov do menších zhlukov na základe ich euklidovskej vzdialenosti. Vďaka spracovaniu do menších zhlukov sa celkový čas spracovania mračna výrazne skráti. Jednoduchý prístup zhlukovania údajov v euklidovskom zmysle možno implementovať použitím 3D mriežkového rozdelenia priestoru pomocou polí s pevnou šírkou.

5.3.1 Algoritmus euklidovskej extrakcie

1. **Inicializácia**

   - Vstupom do algoritmu je predspracované mračno bodov.
   - Zostaví sa Kd-strom z mračna bodov pre efektívne vyhľadávanie susedov. Kd-strom je dátová štruktúra optimalizovaná pre dopyty najbližšieho suseda vo viacrozmernom priestore, ktorý je rozhodujúci pre fázu klastrovania.

2. **Extrakcia klastrov**

   - Pre každý bod v mračne bodov, ktorý ešte nebol priradený do klastra sa vytvorí nový klaster, do ktorého sa rekurzívne pridajú všetci jeho nenavštívení susedia v rámci špecifikovaného okruhu. Pomocou Kd-stromu sa efektívne nájdu susedia.
   - Pokračuje sa v tomto procese, kým sa nenavštívia všetky body v dosahu a nepridajú sa do aktuálneho klastra.
   - Pre definovanie klastra sa využívajú dva hlavné parametre:
     - **Tolerancia klastra:** Definuje maximálnu vzdialenosť medzi dvoma bodmi, ktoré sa považujú za patriace do rovnakého klastra.
     - **Minimálna a maximálna veľkosť klastra:** Používa sa na odfiltrovanie klastrov, ktoré sú príliš malé (pravdepodobne šum) alebo príliš veľké (ktoré môžu zahŕňať viacero odlišných objektov).

3. **Ukončenie**

   - Proces sa ukončí, keď sú všetky body v mračne bodov buď priradené k platnému zhluku alebo označené ako odľahlé body.

Podrobnejšie vysvetlenie nájdete na webovej stránke Point Cloud Library.[6]

5.3.2 Implementácia euklidovskej extrakcie

```
pcl::PCDReader reader;
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud
      (new pcl::PointCloud<pcl::PointXYZ>), cloud_f
      (new pcl::PointCloud<pcl::PointXYZ>);
reader.read ("../output_big.pcd", *cloud);
std::cout << "PointCloud before filtering has: "
      << cloud->size () << " data points." << std::endl;
pcl::VoxelGrid<pcl::PointXYZ> vg;
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filtered
      (new pcl::PointCloud<pcl::PointXYZ>);
vg.setInputCloud (cloud);
vg.setLeafSize (0.01f, 0.01f, 0.01f);
vg.filter (*cloud_filtered);
std::cout << "PointCloud after filtering has: "
      << cloud_filtered->size ()  << " data points." << std::endl;
```

Načítanie a predspracovanie mračna bodov.

```
pcl::SACSegmentation<pcl::PointXYZ> seg;
pcl::PointIndices::Ptr inliers (new pcl::PointIndices);
pcl::ModelCoefficients::Ptr coefficients
      (new pcl::ModelCoefficients);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_plane
      (new pcl::PointCloud<pcl::PointXYZ> ());
pcl::PCDWriter writer;
seg.setOptimizeCoefficients (true);
seg.setModelType (pcl::SACMODEL_PLANE);
seg.setMethodType (pcl::SAC_RANSAC);
seg.setMaxIterations (100);
seg.setDistanceThreshold (0.01);
```

Príprava pre segmentáciu pomocou RANSACu na identifikáciu rovín v mračne bodov.

```
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud_output
        (new pcl::PointCloud<pcl::PointXYZRGB>);
  std::vector<pcl::PointXYZRGB> new_points;
  for(const auto& p : cloud->points) {
    pcl::PointXYZRGB p_rgb;
    p_rgb.x = p.x;
    p_rgb.y = p.y;
    p_rgb.z = p.z;
    // You may also want to set the RGB values
    p_rgb.r = 0; // Red
    p_rgb.g = 0; // Green
    p_rgb.b = 0; // Blue
    new_points.push_back(p_rgb);
  }
  cloud_output->points.insert(cloud_output->points.end(),
        new_points.begin(), new_points.end());
```

Inicializácia mračna bodov RGB.

```
  uint8_t r, g, b;
  int i=0, nr_points = (int) cloud_filtered->size ();
while (cloud_filtered->size () > 0.4 * nr_points)
  {
    // Segment the largest planar component from the remaining cloud
    seg.setInputCloud (cloud_filtered);
    seg.segment (*inliers, *coefficients);
    if (inliers->indices.size () == 0)
    {
      break;
    }

    // Extract the planar inliers from the input cloud
    pcl::ExtractIndices<pcl::PointXYZ> extract;
    extract.setInputCloud (cloud_filtered);
    extract.setIndices (inliers);
```

```
  extract.setNegative (false);

  // Get the points associated with the planar surface
  extract.filter (*cloud_plane);
  r = static_cast<uint8_t>(rand() % 256);
  g = static_cast<uint8_t>(rand() % 256);
  b = static_cast<uint8_t>(rand() % 256);

  for(const auto& p : cloud_plane->points) {
    pcl::PointXYZRGB p_rgb;
    p_rgb.x = p.x;
    p_rgb.y = p.y;
    p_rgb.z = p.z;
    // Set the RGB values for yellow color
    p_rgb.r = r; // Red
    p_rgb.g = g; // Green
    p_rgb.b = b; // Blue
    cloud_output->points.push_back(p_rgb);
  }
  // Remove the planar inliers, extract the rest
  extract.setNegative (true);
  extract.filter (*cloud_f);
  *cloud_filtered = *cloud_f;
}
```

Iteratívna segmentácia roviny z filtrovaného mračna, extrakcia bodov, ktoré tvoria najväčšiu rovinu zistenú v mračne.

```
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree
      (new pcl::search::KdTree<pcl::PointXYZ>);
tree->setInputCloud (cloud_filtered);
pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
ec.setClusterTolerance (0.02); // 2cm
ec.setMinClusterSize (100);
ec.setMaxClusterSize (25000);
ec.setSearchMethod (tree);
```

```
ec.setInputCloud (cloud_filtered);
ec.extract (cluster_indices);
```

Nastavenie na zoskupovanie zostávajúcich bodov pomocou stromu k-d pre rýchle priestorové vyhľadávanie.

```
int j = 0;

for (std::vector<pcl::PointIndices>::const_iterator
        it = cluster_indices.begin ();
        it != cluster_indices.end (); ++it)
{
  pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud_cluster
        (new pcl::PointCloud<pcl::PointXYZRGB>);
    r = static_cast<uint8_t>(rand() % 256);
    g = static_cast<uint8_t>(rand() % 256);
    b = static_cast<uint8_t>(rand() % 256);
    for (std::vector<int>::const_iterator
            pit = it->indices.begin ();
            pit != it->indices.end (); ++pit)
    {
      pcl::PointXYZRGB point;
      point.x = (*cloud_filtered)[*pit].x;
      point.y = (*cloud_filtered)[*pit].y;
      point.z = (*cloud_filtered)[*pit].z;
      point.r =  r;
      point.g =  g;
      point.b =  b;
      cloud_cluster->push_back(point);
    }
    cloud_cluster->width = cloud_cluster->size ();
    cloud_cluster->height = 1;
    cloud_cluster->is_dense = true;

  if(cloud_cluster->size() >= 100) {
    std::stringstream ss;
    ss << "cloud_cluster_" << j << ".pcd";
```

```
        writer.write<pcl::PointXYZRGB>
                (ss.str (), *cloud_cluster, false);
        j++;
      }

        // Merge cloud_cluster into cloud_output
        cloud_output->points.insert(cloud_output->points.end(),
                cloud_cluster->points.begin(),
                cloud_cluster->points.end());
  }
  cloud_output->width = cloud_output->points.size();
  cloud_output->height = 1;
  writer.write<pcl::PointXYZRGB>
        ("cloud_clusterxx.pcd", *cloud_output, false);
```

Zoskupenie zostávajúcich bodov z filtrovaného mračna bodov, priradenie náhodnej farby každému zhluku, uloženie klastrov.

5.4 Porovnanie výsledkov

V tejto sekcii porovnáme výstupy z nami implementovaných algoritmov popísaných v podsekciách 5.1, 5.2 a 5.3.

5.4.1 Algoritmy rastúceho regiónu

Výstup zo segmentácie mračna bodov pomocou štandardného algoritmu rastúceho regiónu môžeme vidieť na 5.1.

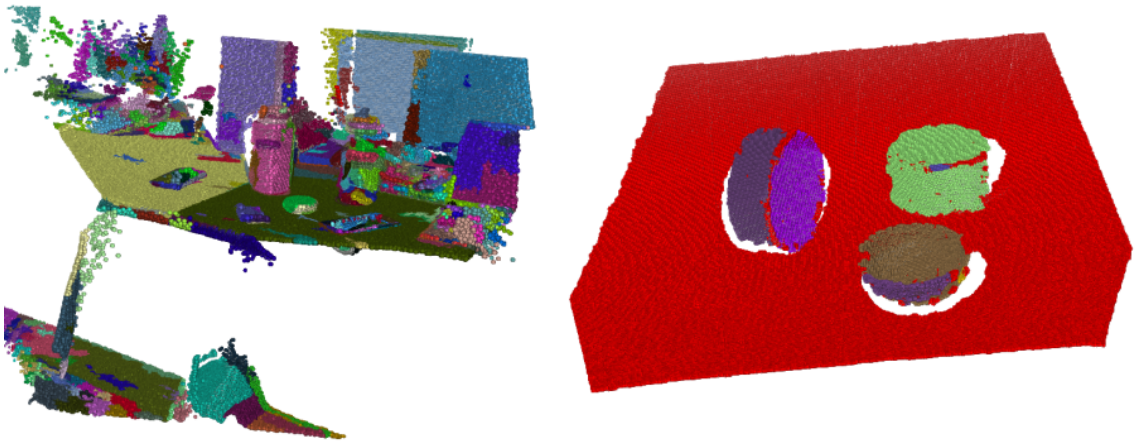
Výstup zo segmentácie mračna bodov pomocou algoritmu rastúceho regiónu podľa farieb naopak vidíme na 5.2.

Ako vidíme, napriek tomu, že algoritmy sú veľmi podobné, alebo aspoň založené na podobnom princípe rastu regiónu, výstupy týchto algoritmov sa veľmi líšia. Algoritmus rastúceho regiónu podľa farieb dokázal vykonať segmentáciu mračna bodov výrazne lepšie ako rastúceho regiónu.

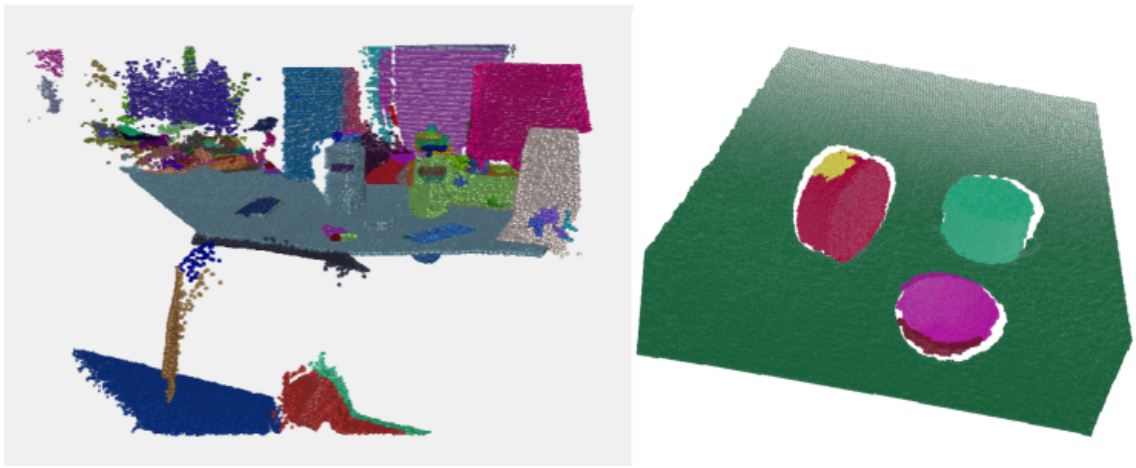
5.4.2 Rastúci región vs Euklidovská extrakcia

Porovnanie rovnakých typov algoritmov segmentácie dalo diametrálne odlišné výsledky. Nebude teda prekvapením, že tomu nie je inak ani v tomto prípade. Výstup zo segmentácie pomocou algoritmu Euklidovskej extrakcie môžeme vidieť na 5.3.

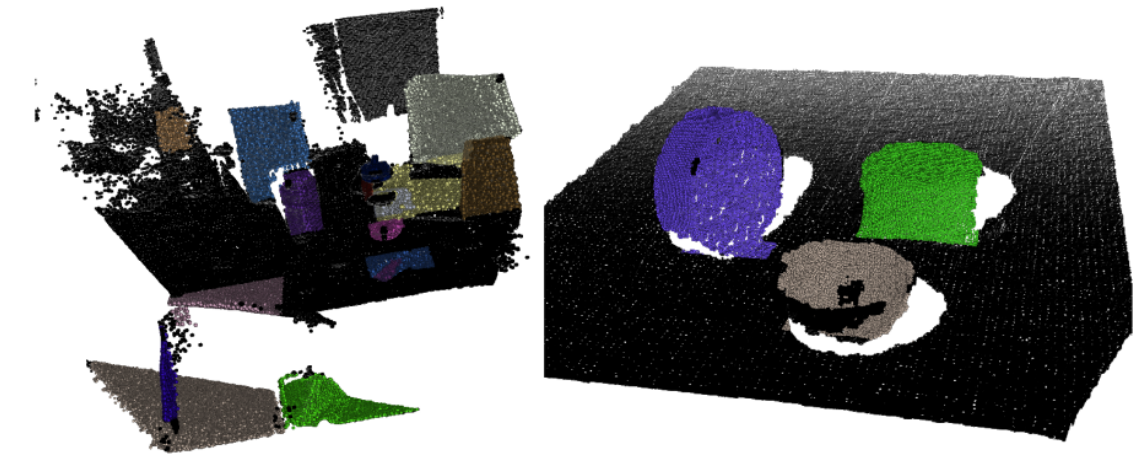
Ak teda porovnáme výstup z algoritmu Euklidovskej extrakcie s výstupom z lepšieho z



Obr. 5.1: Segmentácia algoritmom rastúceho regiónu



Obr. 5.2: Segmentácia algoritmom rastúceho regiónu podľa farieb



Obr. 5.3: Segmentácia algoritmom euklidovskej extrakcie

dvoch algoritmov rastúceho regiónu, ktorý môžeme vidieť na 5.2, môžeme vidieť že algoritmus rastúceho regiónu podľa farieb dokáže vykonať segmentáciu mračna bodov lepšie ako euclidovská extrakcia.

6 Porovnanie v zložitosti nastavovania parametrov

Najľahšie na nastavovanie parametrov bol algoritmus rastúceho regiónu podľa farieb a dostali sme najlepšie výsledky. Najťažšie sa nastavoval rastúci algoritmus. Euklidov algoritmus vracal n-bodov opisujúci objekt takže sme museli dokodiť pridanie bodov aby sme mali opis celého objektu.

Záver

V tomto zadaní sme si osvojili prácu s mračnom bodov - vytvorenie, načítanie, zobrazenie a taktiež sme si vyskúšali rôzne techniky segmentácie objektov v priestore. Popri tom sme využili a vyskušali sme tiež prácu s Kinectom. V tomto dokumente je popísaný proces riešenia jednotlivých úloh, ako aj zbežné vysvetlenie využívaných algoritmov na segmentáciu, respektíve extrakciu klastrov.

Zoznam použitej literatúry

1. OLAZABAL, Oscar. Tutorial for running Kinect on Ubuntu. In: NAML, 2021. Dostupné tiež z: https://www.notaboutmy.life/posts/run-kinect-2-on-ubuntu-20-lts/#hardware-tested.
2. RUSU, Radu Bogdan a COUSINS, Steve. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, 2011.
3. KOCÚR, Maroš. POROVNANIE SPRACOVANIA 3D DÁT ANALYTICKÝM METÓDAMI A NEURÓNOVÝMI SIETAMI. In: STU, FEI: STUFEI, 2023.
4. RUSU, Radu Bogdan a COUSINS, Steve. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, 2011. Dostupné tiež z: https://pcl.readthedocs.io/projects/tutorials/en/latest/region_growing_segmentation.html#region-growing-segmentation.
5. RUSU, Radu Bogdan a COUSINS, Steve. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, 2011. Dostupné tiež z: https://pcl.readthedocs.io/projects/tutorials/en/latest/region_growing_rgb_segmentation.html#region-growing-rgb-segmentation.
6. RUSU, Radu Bogdan a COUSINS, Steve. 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, 2011. Dostupné tiež z: https://pcl.readthedocs.io/projects/tutorials/en/master/cluster_extraction.html#cluster-extraction.