8. Segmentarea prin partiţionare a norilor de puncte

Identificarea suprafeţelor plane din scena de lucru
Segmentarea prin partiţionare a datelor RGB-D

În această secțiune se va prezenta o metodă de segmentare a norilor de puncte în funcție de densitățile de puncte ce formează diferite obiecte în scena vizualizată prin senzori de lumină structurată.

8.1 Baze teoretice

Scenele complexe conţin adesea un număr mare de obiecte sau de elemente necunoscute. Reprezentarea unei astfel de scene într-un mediu virtual implică utilizarea unui nor de puncte de o dimensiune mare. În consecinţă, operaţia de manipulare sau analiză a norului devine costisitoare, atât din punct de vedere al timpului de procesare, cât și din punct de vedere al puterii de calcul. O soluţie posibilă a acestor probleme constă în partiționarea norului P, ce descrie scena iniţială, într-o serie de grupuri compacte de puncte 3D[1]. Avantajele unei astfel de abordări sunt date de interpretarea rapidă a datelor și de posibilitatea grupării punctelor cu proprietăţi similare (culoare, curbură, etc.).

Procesul de grupare compactă a punctelor 3D poate fi considerat ca o segmentare trivială a spațiului Euclidian. Rezultatul unei astfel de segmentări este un set O de grupuri compacte, fiecare conţinând puncte cu proprietăți similare.

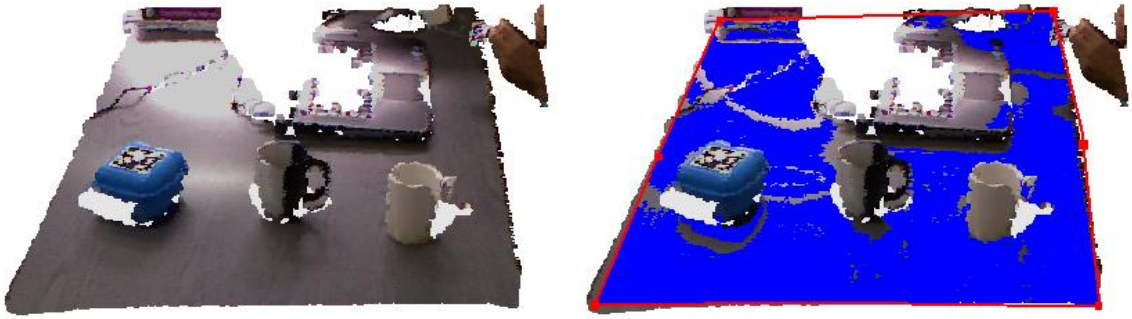
8.1.1 Identificarea suprafeţelor plane din scena de lucru

O primă problemă, în contextul segmentării norilor de puncte, constă în identificarea suprafețelor plane din scenă. Exceptând cazurile în care suprafeţele plane sunt folosite la navigarea roboţilor sau în evitarea obstacolelor, acestea constituie date redundante și pot fi eliminate. O metodă robustă de identificare a suprafețelor plane utilizând imagini RGB-D (*Red, Green, Blue, Depth*) este prezentată în lucrările [**?** **?**] şi presupune parcurgerea următoarelor etape:

- selectarea aleatoare a trei puncte ne-colineare $\{p_i,\ p_j,\ p_k\}$ din norul de puncte P al scenei;
- calcularea coeficienţilor modelului planului, utilizându-se cele trei puncte definite anterior ($a\mathbf{x} + b\mathbf{y} + c\mathbf{z} + d = 0$);
- determinarea distanţei euclidiene dintre toate punctele $p \in P$ şi modelul planului ($a,\ b,\ c,\ d$);
- calcularea numărului de puncte $p^* \in P$, puncte a căror distanţă până la plan respectă condiția $0 \leq |d| \leq |d_{th}|$, unde d_{th} reprezintă un prag de distanţă definit de utilizator.

[1] Eng. *Clusters*

Etapele prezentate anterior sunt efectuate de k ori. Rezultatul unui proces de segmentare a suprafeţelor plane este prezentat în Fig. 8.1. Cu lini roşi sunt definite laturile paralelogramului care defineşte suprafaţa plană estimată, în timp ce cu albastru sunt colorate punctele 3D a căror distantă faţă de planul considerat respectă condiţiile impuse (*inliers*).



(a) (b)

Fig. 8.1 Identificarea suprafeţelor plane din scena de lucru. (a) Norul de puncte iniţial. (b) Suprafaţa plană estimată.

8.1.2 Segmentarea prin partiţionare a datelor RGB-D

Procesul de segmentare a suprafeţelor plane prezentat anterior este doar un pas premergător obţinerii grupurilor compacte de puncte. Obiectivul real este acela de a se partiţiona un nor de puncte P într-o serie de grupuri cu dimensiuni mai mici. O metodă simplă și eficientă de formare a grupurilor compacte de puncte este prezentată în [**?**]. Metoda poate fi folosită doar pentru aplicaţiile care necesită subdiviziuni spațiale egale. Pentru grupuri compacte cu dimensiuni variabile, algoritmul devine mai complex. Avându-se în vedere obiectivul de partiţionare a spaţiului, cu scopul identificării obiectelor care pot fi manipulate, sistemul trebuie să facă distincţia între un *nor de puncte obiect* şi celelalte densități de puncte scenă. Un grup de puncte $O_i = \{P_i \in P\}$ poate fi considerat, din punct de vedere matematic, distinct faţă de un alt grup $O_j = \{P_j \in P\}$ dacă şi numai dacă:

$$\min ||P_i - P_j|| \geq d_{th} \tag{8.1}$$

unde, d_{th} este distanța maximă dintre două puncte ce aparţin aceluiaşi grup compact. Apropierea dintre două sau mai multe puncte 3D poate fi determinată utilizându-se o structură de tipul *kd-tree* [**?**]. Algoritmul de extragere al grupurilor compacte de puncte presupune parcurgerea următoarelor etape:

1. crearea unei structuri arborescente de căutare, de tipul *kd-tree*, utilizându-se norul P al scenei;
2. crearea unei liste goale C, menită să stocheze grupurile compacte care vor fi identificate, cât şi definirea unei stive Q cu punctele care urmează a fi verificate;
3. parcurgerea, pentru fiecare punct p_i din setul P_i, a următorilor paşi:
   - adăugarea lui p_i la stiva Q;
   - pentru fiecare $p_i \in Q$ trebuie să se realizeze:
     - o căutare a unui set de vecini P_i^k, într-un perimetru sferic, cu raza $r < d_{th}$;
     - verificarea, pentru fiecare punct vecin $p_i^k \in P_i^k$, procesării punctului p_i. În situaţia în care punctul considerat nu a fost procesat acesta este adăugat în lista Q;

- verificarea procesării tuturor punctelor din Q. Dacă această operaţie a fost făcută cu succes Q este adăugat în lista grupurilor compacte C. Se şterge Q pentru a se începe o nouă iteraţie;

4. terminarea algoritmului presupune că toate punctele din P_i să fi fost procesate, cât și includerea acestora în lista de grupuri compacte C.

În Fig. 8.2 se pot observa mai multe densităţi de puncte compacte, reprezentând diferite obiecte de uz general, extrase cu ajutorul algoritmului de estimare a grupurilor compacte de puncte. Conform premizelor expuse anterior, obiectele au fost situate pe o suprafaţă plană (masă).

(a) (b)

Fig. 8.2 Segmentarea prin partiţionare a norilor de puncte. (a) Nor de puncte iniţial. (b) Grupuri (clustere) de puncte segmentate.

8.2 Cerinţe

1. Să se încarce un nor de puncte de pe HDD;
2. Să se segmenteze suprafețele plane din norul de puncte;
3. Să se grupeze densitățiile de puncte în funcție de distanța euclidiană dintre ele;
4. Să se salveze și să se vizualizeze scene segmentată.

8.3 Codul sursă al aplicaţiei

```
1  #include <stdio.h>
2  #include <iostream>
3  #include <cstdlib>
4  #include <boost/thread.hpp>
5  #include <pcl/visualization/cloud_viewer.h>
6  #include <pcl/io/ply_io.h>
7  #include <pcl/ModelCoefficients.h>
8  #include <pcl/point_types.h>
9  #include <pcl/io/pcd_io.h>
10 #include <pcl/filters/extract_indices.h>
11 #include <pcl/filters/voxel_grid.h>
12 #include <pcl/features/normal_3d.h>
13 #include <pcl/kdtree/kdtree.h>
14 #include <pcl/sample_consensus/method_types.h>
15 #include <pcl/sample_consensus/model_types.h>
16 #include <pcl/segmentation/sac_segmentation.h>
17 #include <pcl/segmentation/extract_clusters.h>
```

```
18
19 using namespace pcl;
20 using namespace std;
21
22 typedef PointXYZRGBA        PclPointType;
23 typedef PointCloud<PclPointType>  PclCloud;
24
25 PclCloud::Ptr cloud (new PclCloud);
26 PclCloud::Ptr cluster_cloud (new PclCloud);
27
28 void extractClusters(PclCloud::Ptr cloud, float fGranularitate, ↩
      vector<PointIndices>& indecsi_clustere, PclCloud &cloud_filtrat, ↩
      bool bDownsample)
29 {
30   indecsi_clustere.clear();
31   PclCloud::Ptr pCloud_filtrat (new PclCloud);
32
33   if (bDownsample == true)
34   {
35     // Crearea structurii de date ce va contine norul de puncte ↩
        filtrat
36     VoxelGrid<PclPointType> vg;
37
38     vg.setInputCloud (cloud);
39
40     // Aplicarea unei granularitati de 1 cm
41     vg.setLeafSize (fGranularitate, fGranularitate, fGranularitate);
42     vg.filter (*pCloud_filtrat);
43   }
44   else
45   {
46     vector< int > index;
47     removeNaNFromPointCloud(*cloud, *pCloud_filtrat, index);
48   }
49
50   // Crearea structurii de date necesara segmentarii suprafetelor ↩
        plane
51   SACSegmentation<PclPointType> seg;
52   PointIndices::Ptr inliers (new PointIndices);
53   ModelCoefficients::Ptr coeficienti (new ModelCoefficients);
54   PointCloud<PclPointType>::Ptr cloud_suprafata_plana (new PointCloud↩
       <PclPointType> ());
55   seg.setOptimizeCoefficients (true);
56   seg.setModelType (SACMODEL_PLANE);
57   seg.setMethodType (SAC_RANSAC);
58   seg.setMaxIterations (100);
59   seg.setDistanceThreshold (0.02);
60
61   int i=0, nr_points = (int) pCloud_filtrat->points.size ();
62   while (pCloud_filtrat->points.size () > 0.3 * nr_points)
63   {
64     // Segmentarea celei mai mari suprafete plane
```

```
65      seg.setInputCloud (pCloud_filtrat);
66      seg.segment (*inliers, *coeficienti);
67      if (inliers->indices.size () == 0)
68        break;
69
70      // Extragerea punctelor din scena ce se afla cel mai aproape de ↩
           planul model estimat
71      ExtractIndices<PclPointType> extract;
72      extract.setInputCloud (pCloud_filtrat);
73      extract.setIndices (inliers);
74      extract.setNegative (false);
75
76      // Salvarea norului de puncte ce descrie suprafata plana
77      extract.filter (*cloud_suprafata_plana);
78
79      // Eliminarea punctelor suprafetei plane
80      extract.setNegative (true);
81      extract.filter (*pCloud_filtrat);
82      cloud_filtrat = *pCloud_filtrat;
83    }
84
85    // Crearea unei structuri arborescenta kd-tree pentru identificarea↩
         celui mai apropiat vecin
86    search::KdTree<PclPointType>::Ptr arbore (new search::KdTree<↩
        PclPointType>);
87    arbore->setInputCloud (pCloud_filtrat);
88
89    EuclideanClusterExtraction<PclPointType> ec;
90    ec.setClusterTolerance (0.02);  // Distanta maxima dintre puncte
91    ec.setMinClusterSize (100);   // Dimensiunea minima a unei ↩
        densitati de puncte
92    ec.setMaxClusterSize (50000); // Dimensiunea maxima a unei ↩
        densitati de puncte
93    ec.setSearchMethod (arbore);
94    ec.setInputCloud (pCloud_filtrat);
95    ec.extract (indecsi_clustere);
96  }
97
98  int main(int argc, char ** argv)
99  {
100   cout << "Laborator 8: Segmentarea norilor de puncte 3D" << endl;
101
102   string strCaleCloud;
103   PclCloud::Ptr cloud_filtrat (new PclCloud);
104   vector<PointIndices> indecsi_clustere;  // Contine indicii ↩
        punctelor clusterelor estimate
105   float fGranularitate = 0.001;      // Granularitatea norului de ↩
        puncte (1 mm)
106   char buffer[10], buffer2[10];
107
108   if(argc !=2)
109   {
```

```
110     cout<<"Utilizare: aplicatie <cale_nor_sursa.pcd>"<<endl;
111     exit(0);
112   }
113   else
114   {
115     strCaleCloud = argv[1];
116   }
117
118   // Incarcarea unui nor de puncte
119   if( io::loadPCDFile(strCaleCloud.c_str(), *cloud))
120     cout << "Norul de puncte nu a putut fi citit." << endl;
121
122   // Partitionarea norului de puncte
123   extractClusters(cloud, fGranularitate, indecsi_clustere, *↩
        cloud_filtrat, true);
124   cout << "Numarul de clustere = " << indecsi_clustere.size() << endl↩
        ;
125
126   // Salvarea si vizualizarea clusterelor
127   for (unsigned int i = 0; i < indecsi_clustere.size(); i++)
128   {
129     sprintf(buffer, "%d.pcd", i);
130     sprintf(buffer2, "%d.ply", i);
131     copyPointCloud (*cloud_filtrat, indecsi_clustere[i], *↩
          cluster_cloud);
132
133     visualization::CloudViewer vizualizator ("Viualizarea grafica a ↩
          norului de puncte segmentat");
134     vizualizator.showCloud (cluster_cloud);
135     while (!vizualizator.wasStopped ())
136     {}
137
138     io::savePLYFile<PointXYZRGBA>(buffer2, *cluster_cloud, true);
139     io::savePCDFile<PointXYZRGBA>(buffer, *cluster_cloud, true);
140     cluster_cloud->clear();
141   }
142
143   // Salvarea intregii scene
144   io::savePLYFile<PointXYZRGBA>("scena.ply", *cloud, true);
145
146   return 0;
147 }
```

8.4 Descriere codului sursă

```
28 extractClusters (cloud, fGranularitate, indecsi_clustere, ↩
     cloud_filtrat);
```

Partiţionează norul de puncte al scenei de lucru într-o serie de grupuri compacte de uncte 3D[2]. Fiecare grup de puncte segmentat descrie un obiect aflat pe o suprafață plană. Codul sursă al acestei metode poate fi găsit la adresa `http://pointclouds.org/documentation/tutorials/cluster_extraction.php#cluster-extraction`

[2]Eng. *Clusters*

- `cloud`: nor de puncte înfăţişând scena de lucru;
- `fGranularitate`: granularitatea finală a grupurilor de puncte extrase (de exemplu $0.01m = 1cm$);
- `indecsi_clustere`: indecşii punctelor fiecărui grup de puncte segmentat;
- `cloud_filtrat`: norul de puncte ce descrie grupurile de puncte segmentate.

131 `copyPointCloud (const pcl::PointCloud< PointT > &cloud_in, const std::vector< int > &indices, pcl::PointCloud< PointT > &cloud_out);`

Copiază o structură de tip nor de puncte într-o altă structură de acelaşi tip.

- `cloud_in`: nor de puncte sursă;
- `indices`: indecşii punctelor care urmează a fi copiate. Dacă nu este furnizat nici un index, atunci toate punctele din norul sursă vor fi copiate în norul destinaţie;
- `cloud_out`: nor de puncte destinaţie.

138 `io::savePLYFile<PointXYZRGBA>(const std::string &file_name, const pcl::PointCloud< PointT > &cloud);`

Salvează într-un fişier cu extensia *.ply* norul de puncte segmentat. Acest fişier poate fi citit mai departe de majoritatea programelor de modelare grafică precum *MeshLab*.

- `file_name`: calea către fişierul destinaţie. Aceasta, trebuie să conţină la sfârşitul numelui extensia *.ply*;
- `cloud`: norul de puncte care urmează a fi salvat.