7. Alinierea robustă a densităţilor de puncte 3D

Măsurarea distanţei dintre diferite forme geometrice 3D
Estimarea rotaţiei şi a translaţiei optime între nori de puncte
Prezentarea generală a algoritmului ICP
Alinierea parţială şi globală a formelor utilizându-se ICP

În lucrarea de față se vor studia metode de măsurare a distanțelor în interiorul norilor de puncte 3D, estimarea rotației și translației dintre diferite densități de puncte 3D, cât și potrivirea a două densități de puncte utilizându-se algoritmul *Iterative Closest Point* (ICP).

7.1 Baze teoretice

Cunoscându-se doi nori de puncte, M și P, fiecare reprezentat în sisteme de coordonate proprii, se pune problema identificării transformatei de coordonate optime (rotație și translație) care să permită alinierea lui M relativ la P. Metoda ICP [**?** **?**] prezintă o soluţie eficientă şi rapidă pentru o astfel de problemă. Aceasta, utilizează un simplu proces de minimizare a distantei Euclidiene dintre punctele norului, cu scopul de a se aduce modelele considerate într-un sistem de coordonate comun. Această abordarea nu este dependentă de modul de reprezentare al formelor, permiţând astfel alinierea unui număr mare de tipuri de date geometrice [**?**]. Dintre acestea se pot menţiona:

- seturi de puncte;
- segmente de dreaptă;
- curbe implicite: $\vec{g}(x, y, z) = 0$;
- curbe parametrizate: $(x(u), y(u), z(u))$;
- seturi de triunghiuri (suprafeţe reprezentate prin intermediul unui set de triunghiuri($mesh$));
- suprafeţe implicite: $g(x, y, z) = 0$;
- suprafeţe parametrizate: $g(x(u, v), y(u, v), z(u, v))$.

Din punct de vedere al aplicabilităţii, algoritmul poate fi utilizat cu succes la fuzionarea diferitelor perspective ale unui obiect. Scopul unui astfel de proces de fuzionare poate fi fie de a se obţine un volum rigid, fie de a se compara două sau mai multe scene între ele (pentru determinarea similitudinilor sau a diferenţelor dintre scene), fie de stabilire a congruenței dintre două sau mai multe forme (identificarea formelor echivalente), sau de a se estima mişcarea în spațiu a senzorului video.

Considerându-se un caz concret, metoda ICP poate furniza o măsură de *confidență* a apartenenţei unei suprafeţe percepute (de exemplu, o parte a corpului uman) faţă de o suprafaţă de referinţă (de exemplu, întreg corpul uman). Acest proces poartă numele de *potrivire*. În exemplul prezentat se recurge la o potrivire *locală*, deoarece suprafața percepută corespunde doar unei anumite regiuni din suprafața de referință. Similar, poate exista și o

potrivire *globală*, în care alinierea se produce între două suprafețe cu structură constructivă similară, ca și dimensiune (de exemplu, două siluete ale aceleiași căni). Principial, cele 6 grade de libertate necesare unui proces de potrivire, 3 pentru rotație și 3 pentru translație, sunt determinate utilizându-se corespondențe de tipul *celui mai apropiat vecin.*

7.1.1 Măsurarea distanţei dintre diferite date geometrice

Identificarea celui mai apropiat vecin se realizează diferit pentru fiecare tip de dată, de natură geometrică. Se consideră un nor P format din N_p puncte 3D, notate cu p_i : $P = \{p_i\}$, unde $i = 1, \dots N_p$, şi un nor M format din N_m puncte 3D, notate m_j : $M = \{m_j\}$, unde $j = 1, \dots N_m$. Distanţa Euclidiană dintre punctele celor doi nori se determină după cum urmează:

$$d(r_1, r_2) = ||r_1 - r_2||^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}, \tag{7.1}$$

unde, $r_1(x, y, z)$ și $r_2(x, y, z)$ sunt două puncte oarecare aparţinând norilor M și P, iar d este distanța Euclidiană dintre puncte. Distanța dintre punctul $m \in M$ și cel mai apropiat punct de pe suprafața P se calculează folosindu-se următoarea ecuație:

$$d(m, P) = \min_{i \in 1, \dots N_p} d(m, p_i). \tag{7.2}$$

Cel mai apropiat punct $p_i \in P$ satisface egalitatea $d(m, p_i) = d(m, P)$.

7.1.2 Estimarea rotaţiei şi a translaţiei optime între nori de puncte

Pe lângă procedura prezentată anterior, pentru estimare a celui mai apropiat vecin, metoda ICP mai implică utilizarea unui proces de transformare inițială a datelor geometrice. Considerându-se spațiul tridimensional, identificarea rotației și a translației se poate realiza eficient utilizându-se vectorii *quaternion* ($\vec{q}$). În spațiul cu 2 sau 3 dimensiuni vectorii quaternion reprezintă o alternativă eficientă a unghiurilor Euler. Principalul lor avantaj este reprezentat de simplitatea reprezentării lor, cât și timpul de calcul redus [**?**]. Utilizându-se unghiurile Euler, rotația unui corp poate fi obținută prin multiplicarea a trei rotații, câte una pentru fiecare unghi Euler. Utilizându-se quaternionul, rotația se obține într-o singură etapă, care înglobează deja funcțiile trigonometrice sin și cos. Totodată, conversia din quaternion într-o matrice de rotație este eficientă. În această lucrare, va fi utilizată o soluție simplă pentru determinarea rotației și translației, descrisă de Horn în [**?**] .

Algoritmul iterativ de aliniere ICP urmărește mutarea (*înregistrarea*) unei suprafețe M (*suprafață transformată*) cu scopul de a o suprapune, într-o manieră ideală, peste o suprafață de referință P. N_m și N_p reprezintă numărul de puncte ce descriu suprafețele considerate. Distanța metrică d, măsurată între punctele individuale m_i și suprafața P, se poate descrie astfel:

$$d(m, P) = \min_{p \in P} ||p - m||. \tag{7.3}$$

Punctul din P care satisface condiția de cel mai apropiat vecin se notează cu y. Acesta trebuie să respecte condiţia $d(m, y) = d(m, P)$, unde $y \in P$.

Din punctul de vedere al intervalului de calcul, identificarea lui y se realizează în maxim $O(N_p)$, iar timpul total (a tuturor punctelor din M) este reprezentat de $O(N_m N_p)$. Cu toate acestea, timpul de calcul estimat este $O(\log(N_p) \cdot log(N_p))$. Fie Y un vector care conţine cele

mai apropiate puncte și $\mathcal{C}$ operatorul de cel mai apropiat punct, atunci se poate considera:

$$Y = \mathcal{C}(M, P). \tag{7.4}$$

Cunoscând vectorul celor mai apropiate puncte Y, alinierea celor două suprafețe, utilizând metoda celor mai mici pătrate, se calculează după cum urmează:

$$(\vec{q}, d) = \mathcal{Q}(M, Y). \tag{7.5}$$

În continuare, setul de puncte M este actualizat utilizând afirmația $M = \vec{q}(M)$.

7.1.3 Prezentarea generală a algoritmului ICP

Fiind cunoscute aspectele descrise anterior, algoritmul de aliniere ICP poate fi descris, pe scurt, după cum urmează:

- se definesc datele de intrare sub forma a doi nori de puncte. Se notează cu M norul translatat și cu P norul de referință. Dimensiunile celor doi nori sunt N_m și N_p;
- se realizează inițializarea iteraţiilor din algoritm, prin setarea $M_0 = M$, $\vec{q}_0 = [1.0,\ 0.0,\ 0.0,\ 0.0]^t$, și $k = 0$. Vectorii de aliniere sunt definiți relativ la setul translatat M_0, astfel încât alinierea finală să reprezinte transformarea completă. Sunt aplicate etapele (a), (b), (c) și (d) până când algoritmul converge către un minim, cu o toleranţă τ:
  a. se calculează cel mai apropiat punct: $Y_k = \mathcal{C}(M_k, P)$ (timp maxim de calcul: $O(N_m N_p)$, timp mediu de calcul: $O(N_m \log N_p\ )$;
  b. se calculează matricea de transformare utilizată în procesul de aliniere: $(\vec{q}, d_k) = \mathcal{Q}(M_0, Y_k)$ (timp de calcul: $O(N_m)$);
  c. se realizează alinierea: $P_{k+1} = \vec{q}_k(P_0)$ (timp de calcul: $O(N_m)$);
  d. dacă eroarea medie pătratică dintre punctele considerate şi cele de referinţă scade sub o valoare de prag $\tau > 0$ algoritmul se încheie. Precizia procesului de aliniere se determină ţinându-se cont de: $d_k - d_{k+1} < \tau$.

7.1.4 Alinierea parţială şi globală a formelor utilizându-se ICP

Acurateţea şi rapiditatea procesului de aliniere sunt dependente de structura formelor implicate. Contextual, pot exista două situaţii distincte:

- norul translatat descrie o formă globală similară cu acea a norului de referinţă;
- norul translatat descrie o anumită regiune din norul de referinţă.

Aplicarea algoritmului în primul caz presupune, pe de o parte, translatarea centrului de masă μ_m al formei M peste centrul de masă μ_p al formei P, iar, pe de altă parte, identificarea setului de rotaţii optime care produc o eroare de potrivire (convergenţă) minimă. Cu toate acestea, suprapunerea centrelor de masă nu este obligatorie. Se poate recurge la o pretranslație, utilizându-se un set de rotaţii iniţiale. Un exemplu de aliniere a doi nori de puncte care înfăţişează silueta a două căni diferite este prezentat în Fig. 7.1. Cănile sunt vizualizate din aceeaşi perspectivă. Eroarea de potrivire pentru cazul considerat este de $3.1866 \cdot e^{-5}$.

O problemă des întâlnită în procesele de aliniere apare în momentul în care doar o mică parte din punctele formei M au corespondent (cel mai apropiat vecin) în forma P. Un caz real este acela în care se încearcă alinierea unei mici porțiuni a unei căni relativ la o formă rigidă (reprezentată printr-o structură globală). Totuși, dacă regiunea potrivită prezintă suficient de multe particularităţi, algoritmul ICP poate converge optim către un minim local. Trebuie remarcat faptul că numărul de translaţii necesar este considerabil mai mare, deoarece cele două centre de masă nu se mai comportă în mod similar. Un exemplu de aliniere, între o

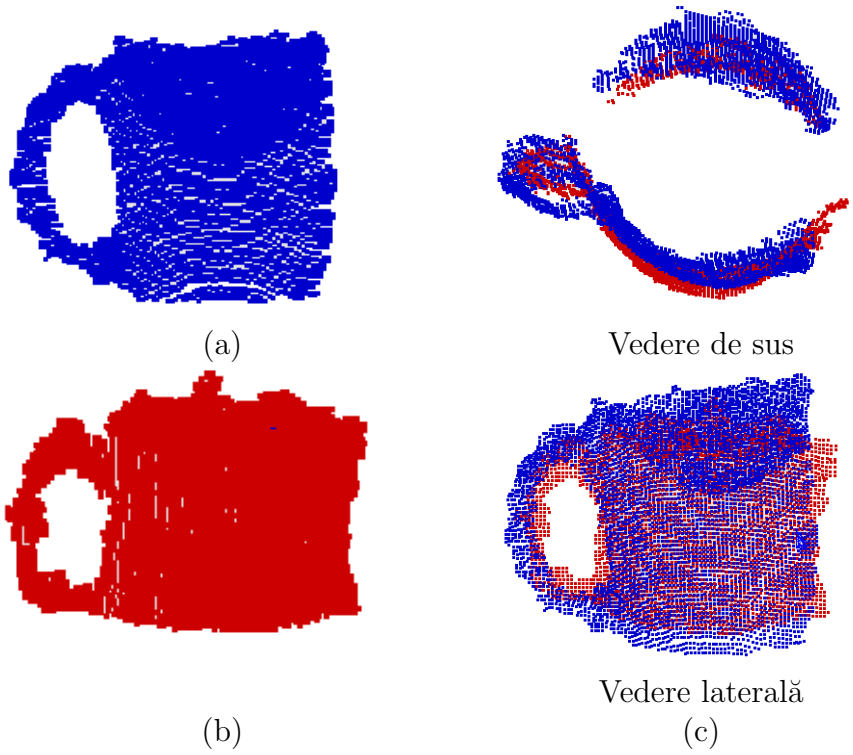

Fig. 7.1 Alinierea a două suprafețe cu siluete similare. (a) Model translatat (M). (b) Model de referință (P). (c) Modele aliniate utilizându-se metoda ICP.

suprafaţă rigidă şi o suprafaţă care descrie doar o anumită regiune a acesteia, este prezentat în figura 7.2. Eroarea de potrivire pentru cazul considerat este egală cu $7.489 \cdot e^{-8}$.

7.2 Cerinţe

1. Să se încarce doi nori de puncte de pe HDD;
2. Să se scrie utilizeze metoda ICP din librăria PCL pentru alinierea celor doi nori de puncte;
3. Să se translateze și să se rotească un nor de puncte, față de celălalt, utilizându-se matricea de transformare returnată de algoritmul ICP.

7.3 Codul sursă al aplicaţiei

```
1  #include <pcl/point_types.h>
2  #include <pcl/point_cloud.h>
3  #include <pcl/io/pcd_io.h>.h>
4  #include <pcl/kdtree/impl/kdtree_flann.hpp>
5  #include <pcl/registration/icp.h>
6
7  using namespace pcl;
8  using namespace std;
9
10 typedef PointXYZRGBA        PclPointType;
11 typedef PointCloud<PclPointType>  PclCloud;
12
13 PclCloud::Ptr cloud_model (new PclCloud);
14 PclCloud::Ptr cloud_aliniat (new PclCloud);
15 PclCloud cloud_rezultat;
```

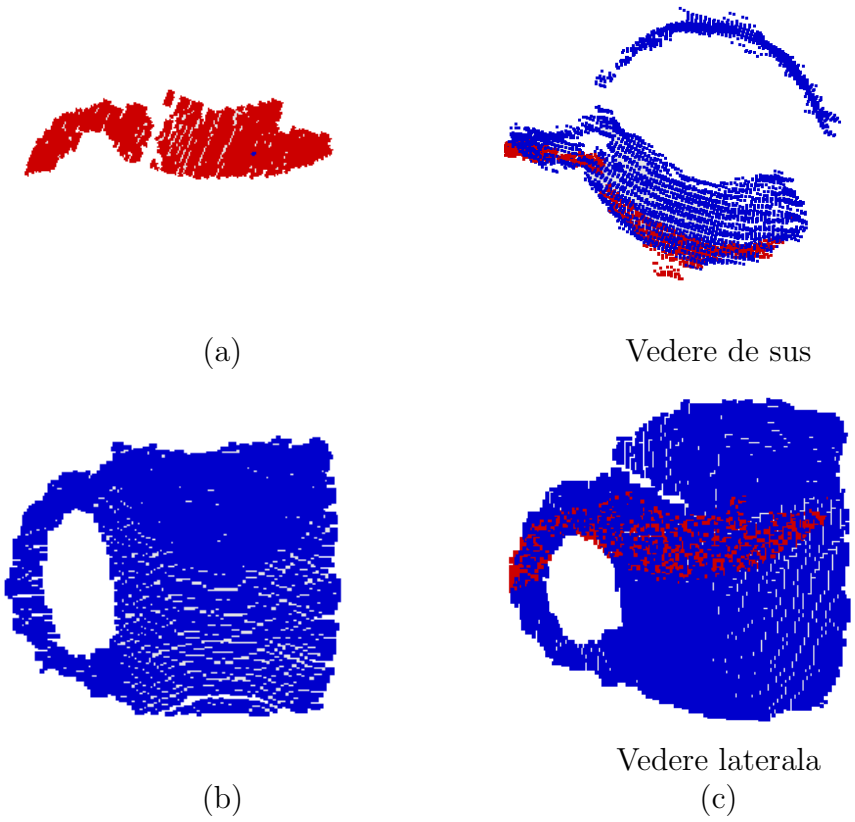

Fig. 7.2 Alinierea unei regiuni la o suprafață rigidă. (a) Model translatat (M). (b) Model de referință (P). (c) Modele aliniate utilizându-se metoda ICP.

```
16 double dConvergenta;
17 Eigen::Matrix4f transformare_ICP;
18 float fDistMaxACorresp = 0.5f;
19 int nMaxIteratii = 1000;
20
21 Eigen::Matrix4f AliniereICP(PclCloud::Ptr cloud_model, PclCloud &↩
      cloud_aliniat, double &dConvergenta, int nMaxIteratii, float ↩
      fDistMaxACorresp)
22 {
23   Eigen::Matrix4f transformare;
24
25   PclCloud::Ptr pCloud_aliniat(new PclCloud);
26   copyPointCloud(cloud_aliniat, *pCloud_aliniat);
27
28   // Clasa algoritmului ICP
29   IterativeClosestPoint<PclPointType, PclPointType> icp;
30
31   // Stabilirea norului de referinta
32   icp.setInputCloud(cloud_model);
33
34   // Stabilirea norului translatat (aliniat)
35   icp.setInputTarget(pCloud_aliniat);
36
37   // Corespondentele cu distanta mai mare decat pragul stabilit vor ↩
       fi rejectate
```

```
38    icp.setMaxCorrespondenceDistance(fDistMaxACorresp);
39
40    // Stabilirea numarului maxim de iteratii
41    icp.setMaximumIterations(nMaxIteratii);
42
43    // Setarea factorul epsilon al tranformarii de similaritate
44    icp.setTransformationEpsilon (1e-8);
45
46    PointCloud<PclPointType> Final;
47
48    // Aplicarea algoritmului ICP
49    icp.align(Final);
50
51    cout << "Convergenta: " << icp.hasConverged() << " avand confidenta↩
          : " << icp.getFitnessScore() << endl;
52
53    // Salvarea valorii convergentei
54    dConvergenta = icp.getFitnessScore();
55
56    // Returnarea matricei de transformare determinata prin ICP
57    return icp.getFinalTransformation();
58  }
59
60  int main (int argc, char ** argv)
61  {
62    cout<<"Laborator 7: Alinierea suprafetelor"<<endl;
63
64    string strCaleCloudA, strCaleCloudB, strCaleSalvareRezultat;
65
66    if(argc !=4)
67    {
68      cout<<"Utilizare: aplicatie <cale_nor_A.pcd> <cale_nor_B.pcd> <↩
            cale_salvare_nori_aliniati.pcd>"<<endl;
69      exit(0);
70    }
71    else
72    {
73      strCaleCloudA = argv[1];
74      strCaleCloudB = argv[2];
75      strCaleSalvareRezultat = argv[3];
76    }
77
78    // Citirea primului nor de puncte
79    if (io::loadPCDFile(strCaleCloudA.c_str(), *cloud_model))
80      cout<<"Norul de puncte nu a putut fi citit."<<endl;
81
82    // Colorare nor de puncte in albastru
83    for (unsigned int a = 0; a < cloud_model->points.size(); a++)
84    {
85      cloud_model->points[a].r = 0;
86      cloud_model->points[a].g = 0;
87      cloud_model->points[a].b = 255;
```

```
88      }
89
90    // Citirea celui de-al doilea nor de puncte
91    if (io::loadPCDFile(strCaleCloudB.c_str(), *cloud_aliniat))
92      cout<<"Norul de puncte nu a putut fi citit."<<endl;
93
94    // Colorare nor de puncte in rosu
95    for (unsigned int b = 0; b < cloud_aliniat->points.size(); b++)
96    {
97      cloud_aliniat->points[b].r = 255;
98      cloud_aliniat->points[b].g = 0;
99      cloud_aliniat->points[b].b = 0;
100   }
101
102   // Aliniere ICP
103   transformare_ICP = AliniereICP(cloud_model, *cloud_aliniat, ↩
        dConvergenta, 300, 0.05);
104
105   // Operatia de transformare ce aliniaza cloud_aliniat la ↩
        cloud_model
106   transformPointCloud(*cloud_aliniat, *cloud_aliniat, ↩
        transformare_ICP);
107
108   // Salvarea norilor cloud_model si cloud_aliniat intr-un singur nor↩
         de puncte
109   for (unsigned int i = 0; i < cloud_model->points.size(); i++)
110     cloud_rezultat.push_back(cloud_model->points[i]);
111
112   for (unsigned int j = 0; j < cloud_aliniat->points.size(); j++)
113     cloud_rezultat.push_back(cloud_aliniat->points[j]);
114
115   io::savePCDFile(strCaleSalvareRezultat.c_str(), cloud_rezultat);
116
117   return 0;
118 }
```

7.4 Descrierea funcţiilor principale

```
21 Eigen::Matrix4f AliniereICP(PclCloud::Ptr cloud_model, PclCloud &↩
     cloud_aliniat, double &dConvergenta, int nMaxIteratii, float ↩
     fDistMaxACorresp)
```

Această funcţie are rolul de a alinia un nor de puncte oarecare (*cloud_aliniat*) la un nor de puncte de referinţă (*cloud_model*). Parametrul *dConvergenta* returnează măsura potrivirii celor doi nori de puncte sub forma unei erori de distanţă. Acest parametru descrie eroare de potrivire ce poate exista înre suprafeţele celor doi nori de puncte, după ce aceştia au fost aliniaţi.

- `cloud_model`: nor de puncte de referință. Asupra acestuia nu trebuie să se aplice nicio transformare de similaritate (coordonate);
- `cloud_aliniat`: nor de puncte ce urmează a fi transformat pentru a se suprapune peste norul de puncte de referință;
- `dConvergenta`: returnează potrivirea dintre punctele celor doi nori;
- `nMaxIteratii`: numărul maxim de iterații a algoritmului ICP;

- `fDistMaxACorresp`: distanța maximă dintre punctele celor doi nori.

```
79 io::loadPCDFile (const string& file_name, PointCloud<PointT> &cloud);
```

Încarcă un fişier tip nor de puncte.
- `file_name`: calea către fişierul sursă;
- `cloud`: structura în care va fi încărcat norul de puncte.

```
106 transformPointCloud(const pcl::PointCloud< PointT > &cloud_in, pcl::↩
    PointCloud< PointT > &cloud_out, Eigen::Matrix4f &transform)
```

Aplică o transformare rigidă (rotație și translație), definită prin intermediul unei matrice de transformare 4×4.
- `cloud_in`: norul sursă (care urmează a fi transformat);
- `cloud_out`: norul transformat;
- `transform`: matricea de transformare.

```
115 io::savePCDFile(const std::string &file_name, const pcl::PointCloud< ↩
    PointT > &cloud)
```

Salvează un nor de puncte într-un fișier cu extensia *.pcd*.
- `file_name`: calea către fișierul destinație;
- `cloud`: norul de puncte ce urmează a fi salvat.