Dzieło edukacyjne

Materiały dydaktyczne - **instrukcja do ćwiczeń** - do przedmiotu

FTP

SEM. III

Ćwiczenia, godz. 15

Autor opracowania:
Dr inż. Jakub Markiewicz

Fundusze Europejskie Wiedza Edukacja Rozwój
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Spis treści

1. Wstęp

Do wykonania ćwiczeń w ramach przedmiotu Fotogrametria Bliskiego Zasięgu niezbędne jest zainstalowanie następujących bibliotek funkcji i interpretera Python:

- Python wersja 3.6, 3.7 lub 3.8
- Open3D - *pip install open3d*
- LasPy - *pip install laspy*
- NumPy - *pip install numpy*

W celu importu i konwersji chmury punktów z formatu LAS do formatu wykorzystywanego przez bibliotekę Open3D należy wykorzystać skrypt (1)

```
# Import potrzebnych bibliotek
import open3d as o3d
import numpy as np
import laspy

#Wczytanie chmury punktów w formacie las
def wczytanie_chmury_punktow_las_konwwersj_do_o3d(plik):
las_pcd = laspy.file.File(plik, mode = 'r')
x = las_pcd.x
y = las_pcd.y
z = las_pcd.z

#Normalizacja koloru
r = las_pcd.red/max(las_pcd.red)
g = las_pcd.green/max(las_pcd.green)
b = las_pcd.blue/max(las_pcd.blue)

#Konwersja do format NumPy do o3d
las_points = np.vstack((x,y,z)).transpose()
las_colors = np.vstack((r,g,b)).transpose()
chmura_punktow = o3d.geometry.PointCloud()
chmura_punktow.points = o3d.utility.Vector3dVector(las_points)
chmura_punktow.colors = o3d.utility.Vector3dVector(las_colors)

return chmura_punktow

def main():
      chmura_punktow = wczytanie_chmury_punktow_las_konwwersj_do_o3d("chmura_dj.las")
      o3d.io.write_point_cloud("chmura_dj_punkty_odstajace.pcd", punkty)
      o3d.io.write_point_cloud("chmura_dj_odfiltrowana.pcd", chmura_punktow_odfiltrowana)

if __name__ == "__main__":
      main()
```

(1)

2. Interaktywna praca z chmurami punktów

Zbiór funkcji zaimplementowanych w bibliotece Open3D umożliwia interaktywną pracę z chmurami punktów. W ramach ćwiczeń wykorzystywane będą funkcje umożliwiające: (1) manualne przycinanie fragmentu chmury punktów w oparciu o zdefiniowany poligon odniesienia, (2) pomiar współrzędnych XYZ punktów oraz (3) obliczanie obszarów opracowania w oparciu o chmurę punktów.

W celu interaktywnego **przycięcia fragmentu** chmury punktów należy wykorzystać funkcję (2):

```
def manualne_przycinanie_chmury_punktów(chmura_punktów):
    print("Manualne przycinanie chmury punktów")
    print("Etapy przetwarzania danych:")
    print(" (0) Manualne zdefiniowanie widoku poprzez obrót myszka lub:")
    print(" (0.1) Podwójne wciśnięcie klawisza X - zdefiniowanie widoku ortogonalnego względem osi X")
    print(" (0.2) Podwójne wciśnięcie klawisza Y - zdefiniowanie widoku ortogonalnego względem osi Y")
    print(" (0.3) Podwójne wciśnięcie klawisza Z - zdefiniowanie widoku ortogonalnego względem osi Z")
    print(" (1) Wciśnięcie klawisza K - zmiana na tryb rysowania")
    print(" (2.1) Wybór zaznaczenia poprzez wciśnięcie lewego przycisku myszy i interaktywnego
narysowania prostokąta lub")
    print(" (2.2) przytrzymanie przycisku ctrl i wybór wierzchołków poligonu lewym przyciskiem myszy")
    print(" (3) Wciśnięcie klawisza C - wybór zaznaczonego fragmentu chmury punktów i zapis do pliku")
    print(" (4) Wciśnięcie klawisza F - powrót do interaktywnego wyświetlania chmury punktów")
    #o3d.visualization.draw_geometries_with_editing(self, window_name='Open3D', width=1920,
height=1080, left=50, top=50, visible=True)
    o3d.visualization.draw_geometries_with_editing([chmura_punktow],window_name='Przycinanie chmury
punktów')
```

(2)

W wyniku przetwarzania danych powstaje nowa chmura punktów, która zapisywana jest do pliku przy wykorzystaniu interfejsu I/O.

Do **pomiaru punktów** na chmurze punktów służy funkcja (3):

```
def pomiar_punktow_na_chmurze(chmura_punktów):
    print("Pomiar punktów na chmurze punktów")
    print("Etapy pomiaru punktów: ")
    print(" (1.1) Pomiar punktu - shift + lewy przycisk myszy")
    print(" (1.2) Cofniecie ostatniego pomiaru - shift + prawy przycisk myszy")
    print(" (2) Koniec pomiaru - wciśnięcie klawisza Q")
    vis = o3d.visualization.VisualizerWithEditing()
    vis.create_window(window_name='Pomiar punktów')
    vis.add_geometry(chmura_punktow)
    vis.run()  # user picks points
    vis.destroy_window()
    print("Koniec pomiaru")
    print(vis.get_picked_points())
    return vis.get_picked_points()
```

(3)

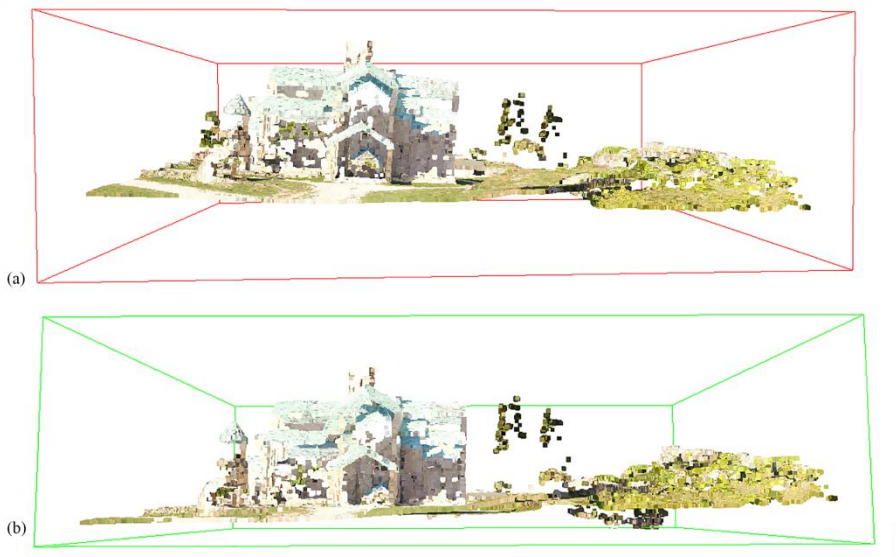
Na rysunku Rys**. 1** przedstawiono przykład wyniku pomiaru punktów za pomocą interaktywnych funkcji biblioteki Open3D.

Rys. 1 Przykład pomierzonych punktów na chmurze punktów

W celu wyznaczenia obszaru opracowania (granic obszaru opracowania) w postaci prostopadłościanu wykorzystuje się funkcję (4)**:**

```
def obliczanie_obszaru_opracowania(chmura_punktów, typ = 'AxisAlignedBoundingBox'):
    if typ == 'AxisAlignedBoundingBox':
        print('Obliczanie obszaru opracowania zorientowanego względem osi XYZ')
        aabb = chmura_punktow.get_axis_aligned_bounding_box()
        aabb.color = (1, 0, 0)
        print(aabb)
        o3d.visualization.draw_geometries([chmura_punktow,aabb],window_name='AxisAlignedBoundingBox')
    else:
        print('Obliczanie obszaru opracowania zorientowanego względem chmury punktów')
        obb = chmura_punktów. get_oriented_bounding_box ()
        obb.color = (0, 1, 0)
        print(obb)
        o3d.visualization.draw_geometries([chmura_punktów,obb], window_name = 'OrientedBoundingBox')
```

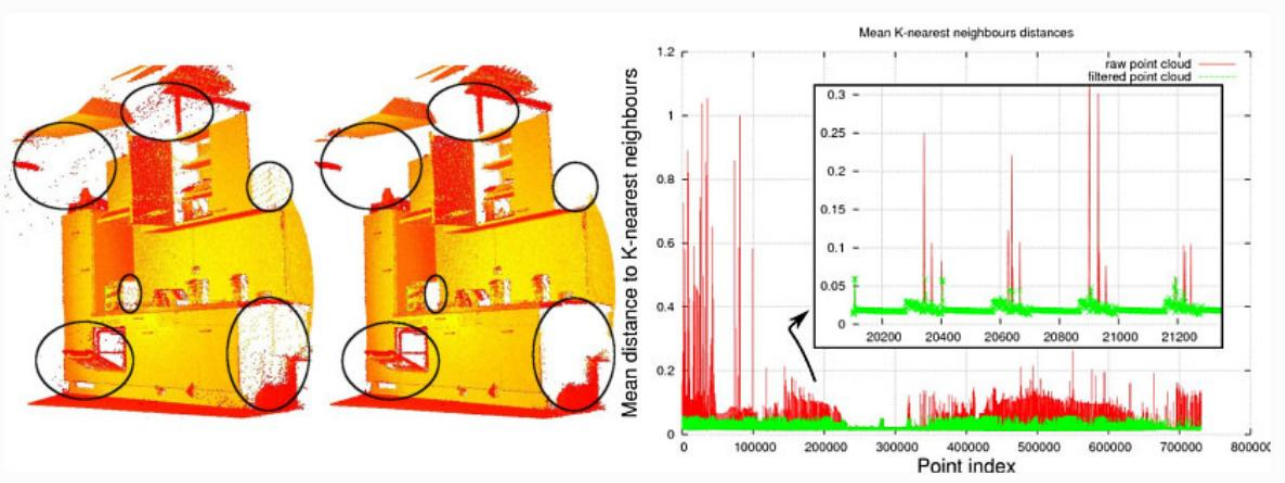
(4)



Rys. 2 Przykład wyznaczenie obszaru opracowania: (a) zorientowanego względem osi XYZ, (b) względem chmury punktów

3. Filtracja chmur punktów

Do wyeliminowania obserwacji odstających (ang. *outliers*) w chmurach punktów stosuje się metody filtracji zależne od wykorzystywanego sensora pomiarowego oraz metody pozyskiwania danych. Występowanie obserwacji odstających znacząco wpływa na poprawne wyznaczenie cech lokalnych chmur punktów (tj. wektorów normalnych, zmiany krzywizn czy lokalną aproksymacje kształtu odniesienia), które mogą prowadzić do błędnego przetwarzania chmur punktów i generowania błędnej dokumentacji na podstawie ww. zbiorów danych. W celu wyeliminowania obserwacji odstających wykorzystuje się metody statystycznej analizy chmur, które są niezależne od wykorzystywanego sensora i sposobu przetwarzania danych. Usuwanie obserwacji odstających polega na analizie sąsiedztwa wokół badanego punktu i sprawdzaniu czy spełnione zostało założone kryterium dokładnościowe. Przykładem filtra statystycznego jest *StatisticalOutlierRemoval,* który bazuje na obliczeniu rozkładu odległości między punktem a jego najbliższymi sąsiadami. Dla każdego punktu obliczana jest średnią odległość od niego do wszystkich jego sąsiadów. Zakładając, że uzyskany rozkład jest rozkładem Gaussa ze średnią i odchyleniem standardowym, wszystkie punkty, których odległości są poza przedziałem zdefiniowanym przez średnią odległości globalną i odchylenie standardowe, można uznać za wartości odstające i usunąć z chmury punktów. Na rysunku 3 przedstawiono efekty działania filtra. Rysunek 3b przedstawia średnie odległości k najbliższego sąsiada w sąsiedztwie punktu przed i po filtrowaniu.



Rys. 3 Przykład działania filtra *StatisticalOutlierRemoval:* (a) chmura punktów przed i po filtracji, (b) przykład wyznaczonego histogramu odległości pomiędzy chmurami punktów przed i po filtracji [1]

W celu implementacji algorytmu *StatisticalOutlierRemoval* przy wykorzystaniu biblioteki Open3D i LasPy, należy wykorzystać funkcje zamieszczone w skrypcie (5).

```
#Filtracja chmur punktów metodą StatisticalOutlierRemoval
def wyznaczanie_obserwacji_odstających(chmura_punktów, liczba_sąsiadów = 30, std_ratio = 2.0):
      chmura_punktów_odfiltrowana, ind = chmura_punktów.remove_statistical_outlier(nb_neighbors
      = liczba_sąsiadów, std_ratio = std_ratio)
      punkty_odstające = chmura_punktów.select_down_sample(ind, invert = True)
      print("Wyświetlanie chmur punktów - punkty odstające (kolor czerwony), chmura i punktów
(kolor RGB): ")
      punkty_odstające.paint_uniform_color([1,0,0])
      o3d.visualization.draw_geometries([chmura_punktów_odfiltrowana, punkty_odstające])
      return chmura_punktów_odfiltrowana, punkty_odstające
```

(5)

W celu usunięcia punktów odstających, w funkcji *StatisticalOutlierRemoval*, należy zdefiniować parametry:

- `liczba_sąsiadow (nb_neighbors)` – liczba najbliższych punktów (sąsiadów) branych pod uwagę w celu obliczenia średniej odległości dla badanego danego punktu,
- `std_ratio` - parametr progu filtrowania związany z krotnością wartości odchylenia standardowego w rozkładzie Gaussa. Im niższa wartość, tym bardziej więcej punktów będzie odfiltrowanych.

Wynik działania skryptu (5) przedstawiono na rysunku Rys**. 4**.

Rys. 4 Przykład działania filtra *StatisticalOutlierRemoval*

W bibliotece Open3D zaimplementowana jest również filtracja punktów, która usuwa punkty posiadające liczbę sąsiadów mniejszych niż przyjęty próg filtracji w danej sferze wokół punktu (radiusoutlierremoval; (6). W celu uśnięcia obserwacji odstających niezbędne jest zdefiniowanie następujących parametrów:

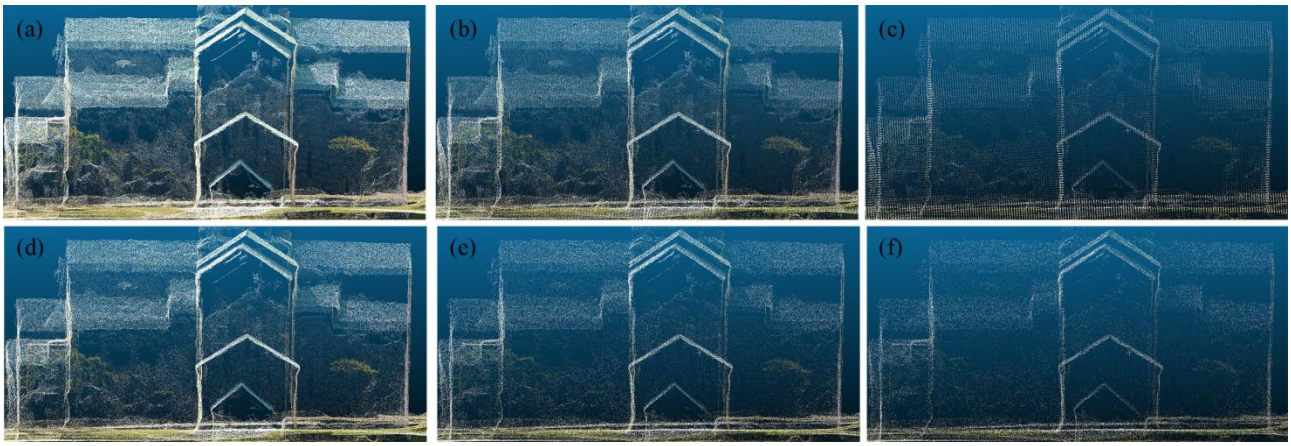
- `min_liczba_punktow (nb_points)` – minimalna liczba punktów znajdujących się w sferze o zdefiniowanym promieniu,
- `promień (radius)` – promień sfery, w której analizowana będzie minimalna liczba punktów.

```
#Filtracja chmur punktów metodą radius_outlier_removal
def wyznaczanie_obserwacji_odstających(chmura_punktów, min_liczba_punktow= 30, promień_sfery =
2.0):
      chmura_punktow_odfiltrowana, ind = chmur_punktow.remove_radius_outlier(nb_points=
      min_liczba_punktow, promien_sfery =0.05)
      punkty_odstające = chmura_punktów.select_down_sample(ind, invert = True)
      print("Wyświetlanie chmur punktów - punkty odstające (kolor czerwony), chmura punktów
(kolor RGB)")
      punkty_odstające.paint_uniform_color([1,0,0])
      o3d.visualization.draw_geometries([chmura_punktów_odfiltrowana, punkty_odstające])
      return chmura_punktów_odfiltrowana, punkty_odstające
```

(6)

4. Generalizacja chmur punktów (*point cloud downsampling*)

Jednym ze sposobów zapisu chmur punktów 3D jest jej konwersja do postaci wokseli (odpowiedniki pikseli w układzie 3D), polegająca na podziale chmury punktów na zbiór prostopadłościanów. W tym podejściu chmura punktów nie jest zapisywana w postaci współrzędnych XYZ, lecz w postaci topologicznych zależności pomiędzy sąsiednimi wokselami i ich odniesienia do początku układu współrzędnych chmury punktów. W zależności od sposobu podziału chmury punktów wyróżniamy 3 typy reprezentacji wokselowej: model regularnej siatki (ang. *Grided Voxel Model*), model niepełnego zbioru (ang. *Sparse Voxel Model*) i w postaci drzewa ósemkowego (ang. *Octree Voxel Model*) [2]. Taki sposób zapisu i przetwarzania danych pozwala na przyspieszenie pracy na chmurze punktów 3D.



Rys. 5 Wyniki działania algorytmów zmniejszających wielkość chmury (downsampling) w oparciu o woksele i odległości między wokselami: (a) 0.1, (b) 0,5 (c) 1 oraz usuwająca co n-ty punkt: (d) 2, (e) 5 i (f) 10.

W celu zmniejszenia wielkości chmur punktów, usunięcia zbędnych punktów lub przekształcenia z postaci nieuporządkowanej do regularnej struktury wokseli (Rys. 5), wykorzystywane są dwa podejścia:

- `voxel_downsample` - bazujące na stworzeniu regularnej siatki wokseli o zadanej odległości pomiędzy nimi. W celu wyznaczenia nowych wartości atrybutów wokseli (tj. kolor, intensywność, etc.) są wyznaczane w oparciu o centroid woksela.
- `uniform_down_sample` - usuwa co n-ty zdefiniowany punkt w chmurze punktów.

Powyższe funkcje zostały przedstawione w kodzie nr (7).

```
#Downsampling chmur punktów
def regularyzacja_chmur_punktów(chmura_punktów, odległość_pomiedzy_wokselami = 0.1):
      chmura_punktów_woksele = chmura_punktów.voxel_down_sample(voxel_size=
      odleglość_pomiedzy_wokselami)
      print("Wyświetlanie chmury punktów w regularnej siatce wokseli - odległość %f: " %
odległość_pomiedzy_wokselami)
      o3d.visualization.draw_geometries([chmura_punktów_woksele])
      return chmura_punktów_woksele
```
(7)

```
def usuwanie_co_n_tego_punktu_z_chmury_punktów(chmura_punktów ,co_n_ty_punkt = 10):
       chmura_punktów_co_n_ty = chmura_punktów.uniform_down_sample(every_k_points=
       co_n_ty_punkt)
       print("Wyświetlanie chmura punktów zredukowanej co %i: " % co_n_ty_punkt)
       o3d.visualization.draw_geometries([chmura_punktów_co_n_ty])
       return chmura_punktów_co_n_ty
```

5. Podział chmur punktów na klasy metodą DBSCAN

Algorytm DBSCAN (*ang. Density-Based Spatial Clustering of Application with Noise*) należy do zbioru algorytmów klasteryzujących, których zadaniem jest eksploracja danych polegająca na podziale zbioru danych na grupy w taki sposób, by elementy w tej samej grupie były do siebie podobne i jednocześnie jednoznacznie rozróżnialne od elementów z pozostałych grup. DBSCAN jest przykładem algorytmu gęstościowego, który oprócz odległości od najbliższych punktów zakłada również zależności wewnętrznej gęstości. Ogólna zasad działania opiera się na zależności, że dopóki liczba obiektów wokół klastra jest duża (z zadanymi parametrami) klaster ten będzie się powiększał. Jest to metoda, która nie wymaga od użytkownika zdefiniowania liczby generowanych klas, nie ogranicza się do kształtów sferycznych/prostokątnych etc. oraz jest odporna na szumy w zbiorze danych.

Sposób działania algorytmu można przedstawić w następujący sposób [3]:

1. Wybierana jest losowa obserwacja x_i.
2. Jeżeli obserwacja x_i ma minimalną liczbę bliskich sąsiadów, wówczas uznawana jest za należącą do klastra.
3. Krok 2 powtarzany jest rekurencyjnie dla wszystkich sąsiadów obserwacji x_i, a następnie sąsiadów tych sąsiadów, itd. Te obserwacje tworzą jądro klastra.
4. Po wyczerpaniu najbliższych obserwacji w kroku 3, następuje losowy wybór nowego punktu i powrót do kroku 1.

Sposób implementacji metody DBSCAN przy wykorzystaniu biblioteki Open3D został przedstawiony w kodzie nr (8). Wyzwalając funkcję `cluster_dbscan` należy zdefiniować następujące parametry:

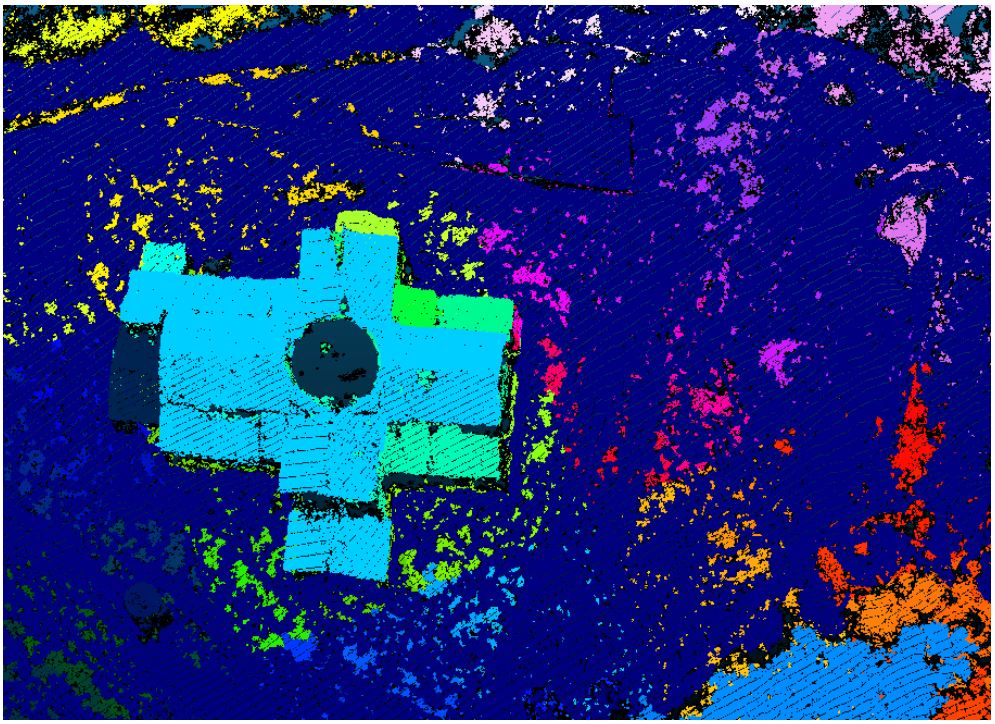
- `odległość_miedzy_punktami(eps)` - minimalna odległość pomiędzy sąsiednimi punktami tworzącymi klaster.
- `min_punktów(min_points)` - minimalna liczba punktów wchodzących w skład klastra.
- `print_progress` - parametr odpowiadający za wyświetlanie informacji o etapach przetwarzania (wartość `True`).

Wynikiem działania algorytmu są etykiety z numerem klasy. Jeżeli wartość etykiety wynosi -1 to algorytm dany punkt zaklasyfikował jako szum.

```
#Klasteryzacja punktów algorytmem DBSCAN

import matplotlib.pyplot as plt

def klasteryzacja_DBSCAN(chmura_punktów,odległość_miedzy_punktami, min_punktow, progress =
True):
      with o3d.utility.VerbosityContextManager(o3d.utility.VerbosityLevel.Debug) as cm:
            klasy = np.array(chmura_punktów.cluster_dbscan(eps= odległość_miedzy_punktami,
      min_points= min_punktów, print_progress= progress))
      liczba_klas = klasy.max()+1
      print("Algorytm DBSCAN wykrył %i klas" % liczba_klas)
      paleta_barwna_kolorów = plt.get_cmap("tab20")( klasy / (klasy.max() if klasy.max() > 0
      else 1)
      colors[klasy < 0] = 0
      chmura_punktów.colors = o3d.utility.Vector3dVector(colors[:, :3])
      o3d.visualization.draw_geometries([chmura_punktów])
```

(8)

Na rysunku Rys**. 6** Przykład działania algorytmu DBSCAN z parametrami: `odległość_miedzy_punktami(eps)` 0.5; `min_punktów (min_points)` - 10. przedstawiono przykład działania algorytmu DBSCAN.



Rys. 6 Przykład działania algorytmu DBSCAN z parametrami: `odległość_miedzy_punktami(eps)` 0.5; `min_punktów (min_points)` - 10.

6. Generowanie modeli 3D w strukturze nieregularnej siatki trójkątów

Chmury punktów powstałe w wyniku przetwarzania zdjęć metodą MVS lub pozyskane z sensorów aktywnych opisują opracowywany obiekt w postaci dyskretnego zbioru punktów. W celu wygenerowania ciągłego model w postaci nieregularnej siatki trójkątów można wykorzystać jedną z trzech metod zaimplementowanych w bibliotece Open3D.

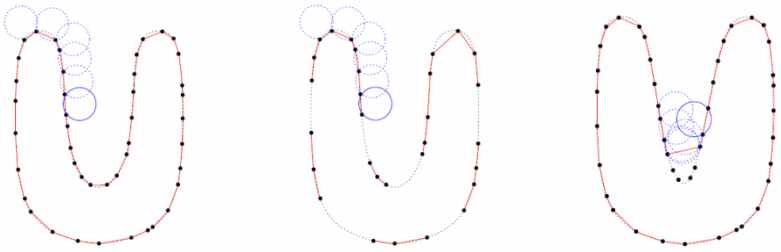
6.1. Alpha shapes

Kształt alfa jest uogólnieniem wypukłego kształtu, który można interpretować w następujący sposób: Wyobraźmy sobie ogromną masę lodów zawierających punkty S jako kawałki twardej czekolady. Za pomocą jednej z tych łyżek do lodów w kształcie kuli wycinamy wszystkie części bloku lodów, do których możemy sięgnąć bez uderzania o kawałki czekolady, tym samym wycinając w ten sposób nawet dziury w środku (np. Części niedostępne po prostu przesuwając łyżkę z na zewnątrz). Ostatecznie otrzymamy (niekoniecznie wypukły) obiekt ograniczony czapkami, łukami i punktami. Jeśli teraz wyprostujemy wszystkie okrągłe ściany do trójkątów i odcinków linii, otrzymamy intuicyjny opis tego, co nazywa się kształtem alfa litery S [4].

```
#Generowanie modelu metodą Alpha Shape

def tin_alpha_shape(chmur_punktow,alfa):
      print("Przetwarzanie danych algorytmem Alpha Shape z parametrem alfa %0.3f " % alfa)
      tin = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(chmur_punktow, alfa)
      tin.compute_vertex_normals()
      o3d.visualization.draw_geometries([tin], mesh_show_back_face=True)
```
(9)

6.2. Ball pivoting [5]

Algorytm Ball-Pivoting [6] łączy sąsiednie punkty powierzchni, jeżeli kolejne punkty znajdują się na okręgu o zadanym promieniu. Zastosowanie stałego promienia skutkuje nastawieniem działania algorytmu głównie na odtwarzanie modeli pokrytych regularną siatką punktów. W przypadku dobrania zbyt dużego promienia model zostanie odwzorowany z pominięciem detali. Zastosowanie zbyt małego promienia prowadzi do powstania dziur (**Rys. 7).**



Rys. 7 Algorytm Ball – Pivoting: kolejne punkty powierzchni łączone są okręgiem o zadanej średnicy (a). Jeżeli promień jest zbyt mały, to metoda tworzy dziury w modelu (b). Jeżeli promień jest zbyt duży, to model zostanie zniekształcony [6]

```
  #Generowanie modelu metodą Ball Pivoting
  def wyznaczanie_normalnych(chmura_punktów):
        print("Wyznaczanie normalnych dla punktów w chmurze punktów")
        chmura_punktów.normals = o3d.utility.Vector3dVector(np.zeros((1, 3))) # Jeżeli
        istnieją normalne to są zerowane
        return chmura_punktów.estimate_normals()

def ball_pivoting(chmur_punktow, promienie_kul = [0.005, 0.01, 0.02, 0.04]):
        chmura_punktow_z_normalnymi = wyznaczanie_normalnych(chmura_punktow):
        print("Przetwarzanie danych algorytmem Ball Pivoting”)
        tin =
        o3d.geometry.TriangleMesh.create_from_point_cloud_ball_pivoting(chmura_punktow_z_norm
        alnymi, o3d.utility.DoubleVector(promienie_kul))
        o3d.visualization.draw_geometries([chmura_punktow_z_normalnymi, tin])
```

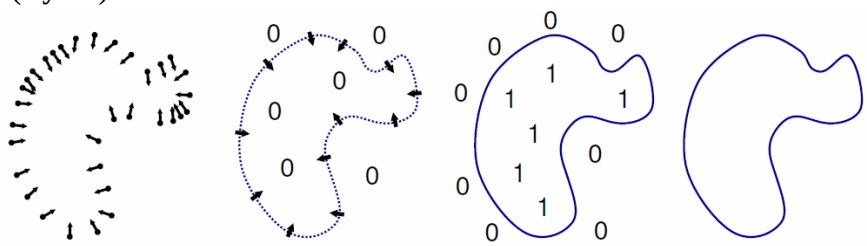
(6)

Rys. 8 Przykład modelu 3D wygenerowanego przy pomocy algorytmu Ball – Pivoting:

6.3. Metoda Poissona [5]

Algorytm wykorzystuje pojęcie funkcji charakterystycznej zbioru χ, określonej przez 1 w przypadku punktu leżącego wewnątrz modelu i 0 dla punktu leżącego na zewnątrz.

Zakłada się, że istnieje związek między wektorami normalnymi punktów tworzących powierzchnię modelu, a funkcją charakterystyczną modelu. W szczególności gradient funkcji charakterystycznej modelu jest polem wektorowym wynoszącym zero prawie wszędzie poza najbliższym otoczeniem powierzchni, gdzie przyjmuje wartości równe normalnym powierzchni. Zatem wektory normalne zbioru punktów mogą być traktowane jako przykłady gradientów indykatora modelu (**Rys. 9**).



Rys. 9. Proces odtwarzania powierzchni. Od lewej: zbiór zaczepionych wektorów normalnych $\bar{V}$, gradient charakterystyczny $\nabla\chi_M$, funkcja charakterystyczna χ_M, powierzchnia ∂M [7].

W ten sposób sprowadza się problem wyznaczenia funkcji charakterystycznej do wyznaczenia funkcji skalarnej χ której gradient najlepiej przybliża pole wektorowe $\overline{\mathrm{V}}$, określone przez wektory normalne modelowanych punktów (wzór (10).

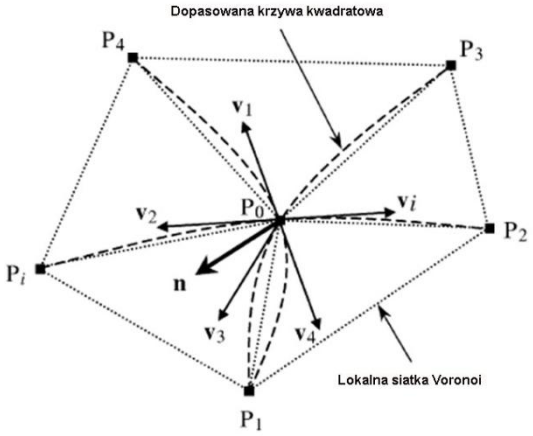
$$\min_{\chi}\left\|\nabla\chi - \vec{\mathrm{V}}\right\| \tag{10}$$

Po zastosowaniu operatora dywergencji problem przekształci się w problem wyrażony równaniem różniczkowym Poissona. Należy znaleźć taką skalarną funkcję χ, której Laplacian (dywergencja gradientu) jest równy dywergencji pola wektorowego $\vec{\mathrm{V}}$

$$\Delta\chi \stackrel{\mathrm{def}}{=} \nabla\nabla\chi = \nabla\vec{\mathrm{V}} \tag{11}$$

Charakterystyka takiego podejścia w odtwarzaniu powierzchni jest następująca:

- rozwiązanie jest globalne, wyliczane na podstawie wszystkich punktów, bez udziału kroków pośrednich, takich jak, wyliczanie rozwiązań lokalnych i budowanie z nich jednolitego modelu,
- uzyskiwanie gładkich powierzchni nawet dla zakłóconych danych; nie powstają lokalne piki wytworzone przez pojedynczy punkt odstający od powierzchni,
- algorytm jest odporny na lokalne nieciągłości w analizowanym polu,
- istnieją efektywne algorytmy rozwiązywania równania Poissona.



Rys. 10. Wyliczanie wartości normalnych dla chmury punktów [8]

Wyliczenie wartości normalnych dla poszczególnych punktów odbywa się na podstawie lokalnego przybliżenia powierzchni. Dla każdego punktu znajduje się n najbliższych punktów, które będą tworzyły podstawy trójkątów stykających się wierzchołkami w rozpatrywanym punkcie. Na bazie tej powierzchni wylicza się wektor normalny (Rys. 10).

Metoda jest bardzo wrażliwa na zmiany w wyliczanych wektorach normalnych. Przy złym doborze parametrów generuje narośle na modelu. Jednakże przy właściwym wyznaczeniu wektorów normalnych metoda daje dobre wyniki.

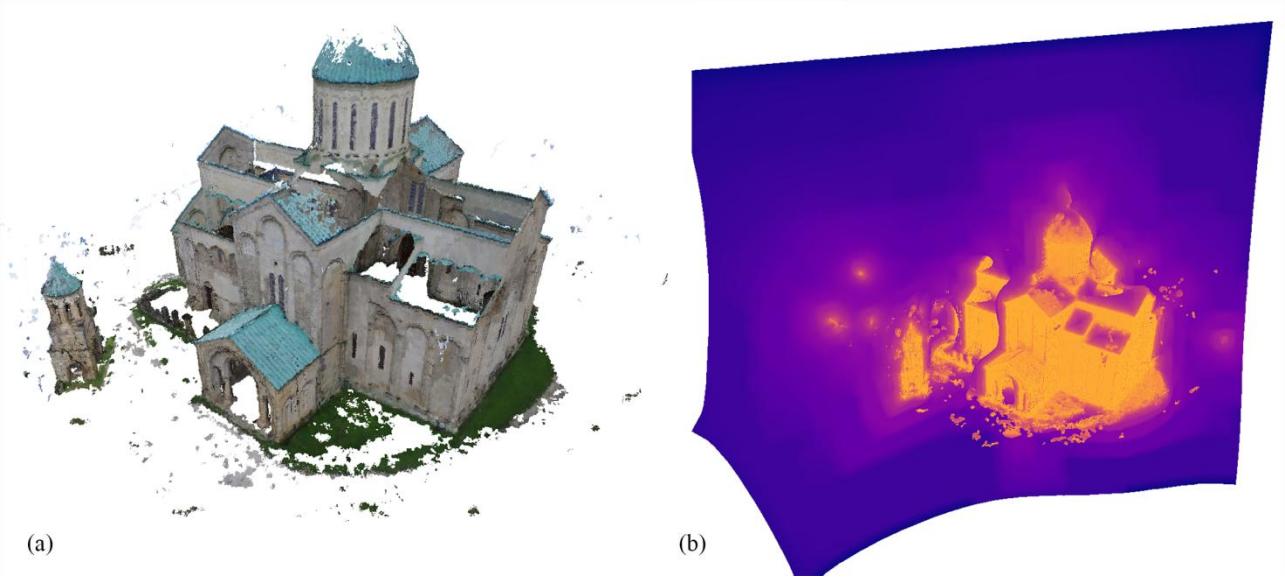
W celu wygenerowanie modelu 3D przy wykorzystaniu funkcji biblioteki Open3D należy uruchomić skrypt (12) i zdefiniować wartość głębi (*ang. depth*), który definiuje poziom octree co

przekłada się na szczegółowość odwzorowywanego modelu. Im ta wartość jest większa, tym uzyskuje się model o wyższej rozdzielczości.

```
#Generowanie modelu metodą Poissona
def wyznaczanie_normalnych(chmura_punktów):
    print("Wyznaczanie normalnych dla piubktow w chmurze punktów")
    chmura_punktów.normals = o3d.utility.Vector3dVector(np.zeros((1, 3))) # Jeżeli istnieją
normalne to są zerowane
    chmura_punktów.estimate_normals()

def poisson(chmura_punktów):
    print("Przetwarzanie danych algorytmem Poissona")
    wyznaczanie_normalnych(chmura_punktów)
    with o3d.utility.VerbosityContextManager(o3d.utility.VerbosityLevel.Debug) as cm:
        tin, gęstość =
o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(chmura_punktow, depth=15)
    print(tin)
    o3d.visualization.draw_geometries([tin])
    return tin, gęstość

#Wyświetlanie gęstości chmur punktów
def wyświetlanie_gęstości(gęstość, tin):
    gęstość = np.asarray(gęstość)
    gęstość _colors = plt.get_cmap('plasma')(
        (gęstość - gęstość.min()) / (gęstość.max() - gęstość.min()))
    gęstość_colors = gęstość _colors[:, :3]
    gęstość_mesh = o3d.geometry.TriangleMesh()
    gęstość_mesh.vertices = tin.vertices
    gęstość_mesh.triangles = tin.triangles
    gęstość_mesh.triangle_normals = tin.triangle_normals
    gęstość_mesh.vertex_colors = o3d.utility.Vector3dVector(gęstość_colors)
    o3d.visualization.draw_geometries([gęstość _mesh])

#Usunięcie punktów w oparciu o wyliczona gęstość
def filtracja_modelu_w_oparciu_o_gęstość(tin, gęstość, kwantyl = 0.01):
      print("Usuniecie trójkątów powstałych w oparciu o małą liczbę punktów")
      vertices_to_remove = gestosc < np.quantile(gestosc, kwantyl)
      tin.remove_vertices_by_mask(vertices_to_remove)
      print(tin)
      o3d.visualization.draw_geometries([tin])

def poisson_filtracja(chmura_punktów):
    tin, gęstość = poisson(chmura_punktów)
    wyświetlanie_gęstości(gęstość, tin)
    filtracja_modelu_w_oparciu_o_gęstość(tin, gęstość, kwantyl = 0.05)
    o3d.io.write_triangle_mesh("model_3d.ply", tin)
```

(12)

Na rysunkuRys. **11**. Przykład modelu wygenerowanego metoda Poissona: (a) ostateczny model w strukturze nieregularnej siatki trójkątów, (b) mapa gęstości modelu przed ostateczną filtracją przedstawiono przykład modelu 3D w strukturze nieregularnej sitaki trójkątów wygenerowanego przy wykorzystaniu algorytmu Poissona oraz wizualizację mapy gęstości trójkątów przed ostateczną filtracją danych.



Rys. 11. Przykład modelu wygenerowanego metoda Poissona: (a) ostateczny model w strukturze nieregularnej siatki trójkątów, (b) mapa gęstości modelu przed ostateczną filtracją

7. Wyznaczanie i opisywanie punktów charakterystycznych (keypoints)

Współczesne oprogramowanie fotogrametryczne służące do przetwarzania zdjęć i chmur punktów opiera się na algorytmach bazujących na połączeniu metod powszechnie stosowanych w widzeniu maszynowym oraz algorytmów wykorzystywanych w klasycznych opracowaniach fotogrametrycznych. Połączenie obu podejść pozwala na orientowanie i przetwarzanie danych w sposób w pełni automatyczny [9].

Do wyszukiwania punktów wiążących na zdjęciach w fotogrametrii stosowna jest metoda ABM działająca powierzchniowo (ang. Area – Based Matching), oraz FBM (ang. Feature – Based Matching), bazująca na wykrytych punktach charakterystycznych (cechach). Tabela 1 przedstawia zestawienie porównawcze metod dopasowywania zdjęć ABM i FBM [10]

Tab. 1 Porównanie metod dopasowywania ABM i FBM [10]

Cecha	Area – Based Matching (ABM)	Feature – Based Matching (FBM)
Strategia dopasowania obrazów	**każdy z dopasowywanych punktów stanowi środek małego okna pikseli, będącego oknem wzorcowym, do którego dopasowywane jest okno poszukiwawcze (o wymiarach równych oknu wzorcowemu)**	**analiza całości zdjęcia, z którego wybierane są jedynie punkty charakterystyczne**
Szybkość	**porównywalna**	**porównywalna**
Uniwersalność	**porównywalna**	**porównywalna**
Dokładność	**niższa**	**większa**
Zbieżność dopasowania	**nie zawsze osiąga wysokie wartości nawet przy zastosowaniu metody *Cross – Corellation***	**wysoka**
Parametry wejściowe	**niezbędny dokładny dobór pierwszych przybliżeń parametrów wejściowych**	**brak konieczności wyznaczenia parametrów wejściowych**
Czułość na błędy	**większa**	**mniejsza, wynikająca z zaproponowanego modelu matematycznego**

Większość obecnie stosowanych aplikacji do orientacji zdjęć naziemnych i chmur punktów wykorzystuje algorytmy FBM zaimplementowane w podejściu SfM, bazujące na wykrywaniu cech charakterystycznych. Obecnie algorytmy ABM stosowane są głównie do generowania gęstych chmur punktów na podstawie sieci zdjęć. W odróżnieniu do metody ABM, gdzie wykorzystywane jest okno poszukiwawcze, w metodzie FBM analizowany jest obszar całego zdjęcia, na którym to dopasowywane są do siebie jedynie wybrane fragmenty, charakteryzujące się zbliżonymi cechami. Przyjmuje się zatem pewne założenia definiujące dany punkt jako punkt charakterystyczny [9]:

- jednoznaczność punktów wykrywanych przez algorytmy FBM – wykrywane punkty muszą odróżniać się od najbliższych sąsiadów. Wskazane jest, aby wykrywane punkty były wysoce kontrastowe, co wyróżniałoby je w lokalnym otoczeniu,
- niezmienność – punkt powinien być wykryty niezależnie od zmian w dystorsji radiometrycznej czy geometrycznej. Warunek ten w największym stopniu wpływa na dokładność, poprawność i niezawodność procesu dopasowywania punktów charakterystycznych,
- stabilność – punkty powinny być wykrywane zawsze jednakowo niezależnie od warunków otoczenia (np. zmian oświetlenia czy skali),
- interpretowalność – podział wykrywanych cech według typu (np. krawędź czy narożnik),
- unikalność – elementy wykrywane na zdjęciach powinny charakteryzować się unikalnością, umożliwiającą ich jednoznaczne wyodrębnienie na podstawie pewnych cech charakterystycznych. Właściwość ta jest szczególnie istotna dla obrazów zawierających powtarzalną teksturę.

W metodzie SfM cechy charakterystyczne wykrywane są pojedynczo dla każdego opracowywanego zdjęcia, a ich wzajemne dopasowanie odbywa się w kolejnym etapie. Mogą one obejmować punkty, krawędzie, linie lub nawet całe regiony – w ten sposób tworzą grupę tzw. cech lokalnych, a kilka lub kilkanaście cech lokalnych stanowi tzw. strukturę globalną.

Wyszukiwanie punktów wiążących jest procesem trzyetapowym. Początkowo, przy użyciu detektorów, wykrywane są obszary odniesienia według zadanego klucza. Następnie za pomocą deskryptorów opisywane są ich cechy charakterystyczne. Finalne dopasowanie punktu dokonywane jest w oparciu o metody statystycznego dopasowania cech (Moussa, 2006; Urban i Weinmann, 2015; Markiewicz, 2016).

7.1. Detekcja punktów charakterystycznych (keypoints) przy wykorzystaniu metody ISS (Intrinsic Shape Signatures)

W celu wykrycia punktów charakterystycznych w chmurach punktów tzw. 3D deskryptorów, analizowana jest m.in. zmiana gradientów wektorów normalnych (zmiany kształtu). W bibliotece Open3D zaimplementowany jest algorytm ISS Keypoint [14]:

```
#Wykrywanie punktów metoda ISS
def detektor_ISS(chmura_punktów):
    print('Detekcja punktów charakterystycznych')
    keypoints = o3d.geometry.keypoint.compute_iss_keypoints(chmura_punktów)
    keypoints.paint_uniform_color([1.0, 0.0, 0.0])
    o3d.visualization.draw_geometries([keypoints])
```
(13)

W celu poprawy wizualizacji wykrytych punktów charakterystycznych należy manualnie przedstawić punkty w postaci sfer (14).

```
#Poprawa wizualizacji wykrytych punktów charakterystycznych
def wizualizacja_punktow_charakterystycznych_za_pomoca_sfer(keypoints):
    sfery = o3d.geometry.TriangleMesh()
    for keypoint in keypoints.points:
        sfery = o3d.geometry.TriangleMesh.create_sphere(radius=0.01)
        sfery.translate(keypoint)
        sfery += sfery
    sfery.paint_uniform_color([1.0, 0, 0.0])
    return sfery
```
(14)

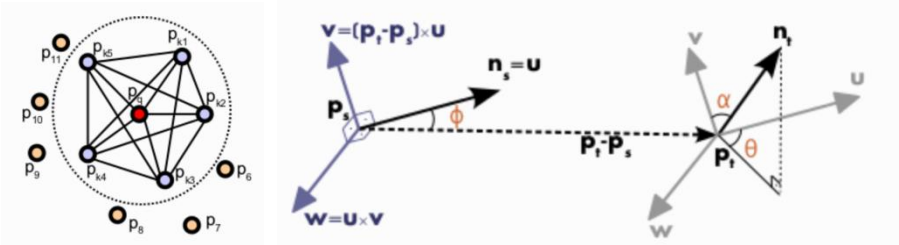
7.2. Opis punktów charakterystycznych za pomocą deskryptora FPFH (Fast Point Feature Histograms)

Do opisu punktów kluczowych wykrywanych na zdjęciach lub danych rastrowych analizowana jest zmiana gradientów stopnia szarości. W przypadku danych 3D (chmur punktów) możliwe jest wykorzystanie zmian gradientów kolorów/intensywności (jeśli zapisana) lub zmiany gradientów kształtu. W przypadku analizy intensywności pozyskanej z naziemnego skaningu laserowego należy mieć na uwadze zależność między jej wartościami a dokładnością odwzorowania kształtu. Dlatego też zasadne jest wykorzystywanie zmian gradientów kształtu (reprezentowanego za pomocą normalnych punktu do powierzchni aproksymowanej na podstawie najbliższych punktów), które pozwala na przetwarzanie chmur punktów pozyskanych z dowolnego sensora. Jedną z metod wykorzystywanych do wyznaczania zależności pomiędzy wektorami normalnymi punktów charakterystycznych jest deskryptor Fast Point Feature Histogram, który jest uogólnieniem deskryptora Point Feature Histogram [1].

Deskryptor PFH bazuje na właściwości geometrycznych k - sąsiedztwa punktu poprzez uogólnienie średniej krzywizny wokół punktu przy użyciu wielowymiarowego histogramu wartości. Ta wielowymiarowa hiperprzestrzeń zapewnia sygnaturę informacyjną dla reprezentacji cech, jest niezmienna w przestrzeni i dobrze radzi sobie z różnymi gęstościami próbkowania lub poziomami szumu obecnymi w sąsiedztwie.

Reprezentacja histogramu elementu punktowego jest oparta na związkach między punktami w sąsiedztwie k i ich oszacowanymi normalnymi powierzchni. Mówiąc najprościej, stara się jak najlepiej uchwycić próbkowane zmiany powierzchni, biorąc pod uwagę wszystkie interakcje między kierunkami oszacowanych normalnych. Wynikowa hiperprzestrzeń jest zatem zależna od jakości estymacji normalnych powierzchni w każdym punkcie.

Rysunek Rys**. 12**a przedstawia diagram obszaru wpływu obliczenia PFH dla punktu zapytania ($\boldsymbol{p_q}$), zaznaczonego na czerwono i umieszczonego w środku koła (sfera w 3D) o promieniu $\boldsymbol{r}$ i wszystkich jego $\boldsymbol{k}$ sąsiadów (punktów o odległościach mniejszych niż promień r) są w pełni połączone w siatkę. Ostateczny deskryptor PFH jest obliczany jako histogram relacji między wszystkimi parami punktów w sąsiedztwie, a zatem ma złożoność obliczeniową O ($\boldsymbol{k^2}$).



Rys. 12. Sposób wyznaczania (a) wektora normalnego w promieniu wokół badanego punktu, (b) sposób wyznaczania zależności między wektorami normalnymi - wektorem normalnym w badanym punkcie a najbliższymi punktami [1].

W celu obliczenia względnej różnicy między dwoma punktami $\boldsymbol{p_i}$ i $\boldsymbol{p_j}$ oraz powiązanymi z nimi normalnymi $\boldsymbol{n_i}$ i $\boldsymbol{n_j}$, definiujemy układ współrzędnych w jednym z punktów (Rys. 12b).

$$\begin{aligned} u &= n_s \\ v &= u \; x \frac{(p_t - p_s)}{\|p_t - p_s\|_2} \\ w &= u \; x \; v \end{aligned} \tag{15}$$

$$\alpha = v * n_t$$
$$\emptyset = u * \frac{(p_t - p_s)}{d}$$
$$\theta = \arctan(w * n_t, u * n_t)$$

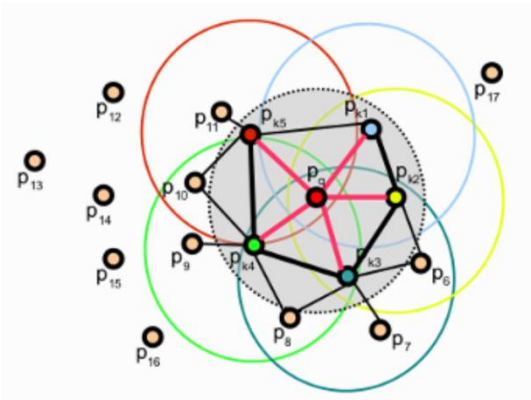
Różnicę między dwoma normalnymi $\boldsymbol{n_s}$ i $\boldsymbol{n_t}$ można wyrazić jako zbiór cech kątowych za pomocą wzoru (15), gdzie $\boldsymbol{d}$ jest odległością euklidesową między dwoma punktami. Kwadruplet $(\alpha, \emptyset, \theta, d)$ jest obliczany dla każdej pary dwóch punktów w sąsiedztwie k, zmniejszając w ten sposób 12 wartości (XYZ i normalne) z dwóch punktów i ich normalnych do czterowymiarowego wektora.

FPFH deskryptor jest uproszczoną wersją deskryptora PFH i bazuje na następujących zależnościach:

- w pierwszym kroku dla każdego przetwarzanego punktu $\boldsymbol{p_q}$ obliczany jest zestaw parametrów $\alpha, \emptyset, \theta$ między punktem a jego sąsiadami, zgodnie z opisem deskryptor PFH - histogram uproszczonego elementu punktu (Simplified Point Feature Histogram, SPFH),
- w drugim kroku dla każdego punktu wyznacza się ponownie k sąsiadów, a sąsiednie wartości SPFH są używane do ważenia końcowego histogramu p_q (zwanego FPFH) w następujący sposób (16):

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{w_i} * SPFH(p_i) \qquad (16)$$

gdzie waga w_i to odległość między punktem $\boldsymbol{p_q}$ a sąsiednim punktem $\boldsymbol{p_i}$ w zadanej przestrzeni metrycznej.



Rys. 13. Schemat doboru wag w algorytmie FPFH [1]

W celu zrozumienia sposobu ważenia obserwacji, na rysunku Rys**. 13** przedstawiono diagram regionu wpływu dla zbioru k - sąsiedztwa dla punktu $\boldsymbol{p_q}$. Dla danego punktu $\boldsymbol{p_q}$ algorytm w pierwszym etapie wyznacza wartości SPFH, tworząc pary między sobą a jego sąsiadami (zilustrowane czerwonymi liniami). Jest to powtarzane dla wszystkich punktów w zbiorze danych, po czym następuje ponowne ważenie wartości SPFH $\boldsymbol{p_q}$ przy użyciu wartości SPFH k sąsiadów, tworząc w ten sposób FPFH dla $\boldsymbol{p_q}$. Dodatkowe połączenia FPFH, wynikające z dodatkowego schematu ważenia, są pokazane czarnymi liniami. Jak pokazuje diagram, niektóre pary wartości zostaną policzone dwukrotnie (zaznaczone grubszymi liniami na rysunku).

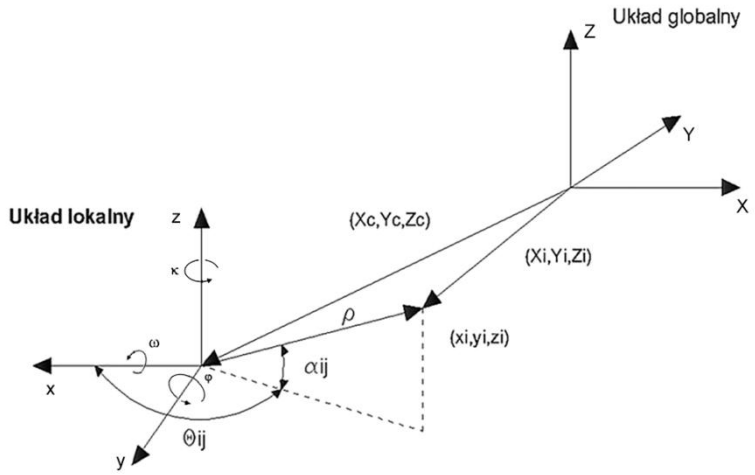
W celu opisania punktów za pomocą deskryptora FPFH przy wykorzystaniu funkcji biblioteki Open3D należy uruchomić skrypt (17) i zdefiniować maksymalną liczbę najbliższych sąsiadów (`max_nn=liczba_n_sąsiadów`) oraz wielkość promienia sfery (`radius= odległość_miedzy_punktami *5`), w której wyznaczanie będą wartości wektorów normalnych i kątów między wektorami.

```
#Wyznaczenie deskryptora metoda FPFH
def deskryptor_3D_FPFH(chmura_punktów, odległość_miedzy_punktami, liczba_n_sąsiadów = 50):
    promień = odległość_miedzy_punktami *5
    print('Wyliczanie deskryptora FPFH w promieniu %.3f i %i najbliższych sąsiadów' %
(promień, liczba_n_sąsiadów))
    deskrypotr_fpfh = o3d.pipelines.registration.compute_fpfh_feature(chmura_punktów,
    o3d.geometry.KDTreeSearchParamHybrid(radius= promień, max_nn= liczba_n_sąsiadów))
    return deskrypotr_fpfh
```

(17

8. Orientacja chmur punktów [9]

Etapem kluczowym w przetwarzaniu danych TLS jest proces orientacji danych, polegający na łączeniu chmur punktów w przyjętym układzie odniesienia, którym może być państwowy układ współrzędnych, układ lokalny lub układ wewnętrzny, zawiązany z jednym ze skanów, tzw. skanem referencyjnym. Wskutek przeprowadzonego procesu orientacji danych, dla każdego ze skanów uzyskujemy elementy orientacji zewnętrznej, czyli wyznaczenia położenia środka układu skanera w przyjętym układzie odniesienia oraz kątów obrotu, wykorzystywanych następnie do transformacji chmury punktów (Rys. 14).



Rys. 14. Zależność pomiędzy układem skanera a układem globalnym. ρ – odległość od punktu, θ_{ij} – kąt poziomy i α_{ij} – kąt pionowy mierzony przez skaner, (x_i, y_i, z_i) – współrzędne mierzonego punktu, κ – kąt obrotu względem osi z, φ – kąt obrotu względem osi y, ω –kąt obrotu względem osi x, (X_c, Y_c, Z_c) – wektor translacji [15]

W tym celu najczęściej wykorzystywana jest transformacja 3D afiniczna. W przypadku występowania różnic w skali pomiędzy układem skanera a zewnętrznym układem odniesienia zalecane jest stosowanie transformacji 3D przez podobieństwo, która wykonywana jest w oparciu o punkty dostosowania (minimum 3) równomiernie rozmieszczone na całym obszarze opracowania. Zwiększenie liczby wykorzystywanych punktów dostarcza obserwacji nadliczbowych, co podnosi dokładność orientacji danych i pozwala na wyeliminowanie błędów grubych.

Zależność pomiędzy lokalnym układem instrumentu a globalnym układem odniesienia wyrażona jest wzorami (18) i (19):

$$M_{zew} = R_{\omega\varphi\kappa} * M_{instr} + T \tag{18}$$

$$R_{\omega\varphi\kappa} = R_1(\omega)R_2(\varphi)R_1(\kappa) \tag{19}$$

gdzie: M_{zew} – wektor współrzędnych punktów w układzie globalnym, M_{instr} – wektor współrzędnych punktów w układzie lokalnym (skanera), T – macierz translacji, $R_{\omega\varphi\kappa}$ – macierz obrotów, $R_3(\kappa)$ – macierz obrotów względem osi z o kąt κ, $R_2(\varphi)$ – macierz obrotów względem osi y o kąt φ i $R_1(\omega)$ – macierz obrotów względem osi x o kąt ω.

Macierze obrotu trójwymiarowego są bezpośrednio związane z opracowywaniem danych ze skaningu laserowego i mogą być definiowane za pomocą różnych parametrów i formuł matematycznych. Wyróżnia się następujące sposoby ich przedstawienia:

- macierze obrotów względem kartezjańskiego układu współrzędnych, gdzie macierz obrotu definiowana jest poprzez skrętność względem osi *x,y,z* oraz kąt obrotu *ω, φ, κ* (Rys. 14).

Macierz transformacyjna dla każdej ze współrzędnych przedstawiana jest za pomocą wzorów (20) - (22):

$$R_1(\omega) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\omega & -sin\omega \\ 0 & sin\omega & cos\omega \end{pmatrix} \quad (20)$$

$$R_2(\varphi) = \begin{pmatrix} cos\varphi & 0 & sin\varphi \\ 0 & 1 & 0 \\ -sin\varphi & 0 & cos\varphi \end{pmatrix} \quad (21)$$

$$R_3(\kappa) = \begin{pmatrix} cos\kappa & -sin\kappa & 0 \\ sin\kappa & cos\kappa & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (22)$$

Macierz wektorów obrotu może również zostać przedstawiona przy pomocy 3 oddzielnych kątów. Parametry transformacyjne pomiędzy układem lokalnym i globalnym odniesione są do dodatniej wartości współrzędnej X.

- w formie macierzy obrotu Θ mierzonego względem osi n reprezentowanej przez wektor własny macierzy obrotów (wzór (23))

$$n = (n_1 \quad n_2 \quad n_3)^T \quad (23)$$

Macierz transformacji wykorzystująca parametr *n* została przedstawiona we wzorze (24):

$$R = \begin{pmatrix} cos\theta + n_1^2(1-cos\theta) & n_1n_2(1-cos\theta) + n_3sin\theta & n_1n_3(1-cos\theta) - n_2sin\theta \\ n_1n_2(1-cos\theta) - n_3sin\theta & cos\theta + n_2^2(1-cos\theta) & n_2n_3(1-cos\theta) + n_1sin\theta \\ n_1n_3(1-cos\theta) + n_2sin\theta & cos\theta + n_2^2(1-cos\theta)\theta - n_1sin\theta & cos\theta + n_3^2(1-cos\theta) \end{pmatrix} \quad (24)$$

w formie macierzy obrotów jako macierzy transformacji opartej na jednostkach kwaternionów (wzór (25)) [16].

$$R = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_3) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \quad (25)$$

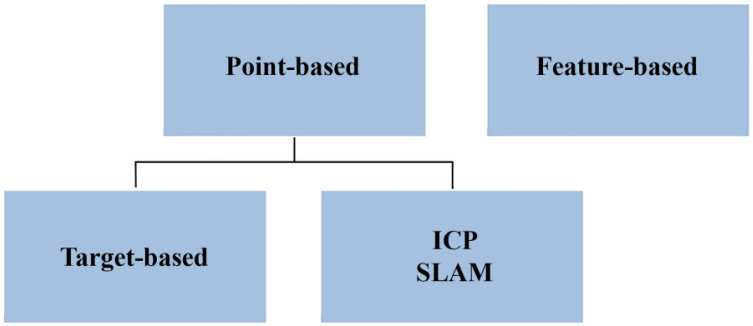
gdzie: q_0, q_1, q_2, q_3 – kolejne kwaterniony

Wartości kwaternionów można przedstawić w funkcji kąta θ i parametru określonej osi w postaci wzoru 4.9:

$$\dot{q} = \cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right) n_1 i + sin\left(\frac{\theta}{2}\right) n_2 j + sin\left(\frac{\theta}{2}\right) n_3 k \tag{4.9}$$

Przedstawienie macierzy z wykorzystaniem kątów i półosi oraz za pomocą kwaternionów znalazło zastosowanie w procesach łączenia i orientacji skanów w postaci chmur punktów.

W celu wyznaczenia elementów orientacji chmur punktów stosuje się dwie główne metody: bazujące na punktach (ang. *point – based*) lub oparte na cechach wykrywanych w chmurze punktów (ang. *feature – based*). Podział metod orientacji danych TLS zaprezentowany jest na rysunku Rys**. 15**.



Rys. 15. Schemat podziału metod orientacji danych z naziemnego skaningu laserowego ze względu na rodzaj wykorzystywanych punktów wiążących

Metody *point – based* możemy podzielić na dwie główne grupy: metody bazujące na dopasowaniu chmur punktów na podstawie sygnalizowanych punktów osnowy (ang. *target – based*) oraz metody bazujące na dopasowaniu grup punktów do płaszczyzn odniesienia, chmur punktów lub kształtów (metody ICP – ang. *Iterative Closest Point,* oraz SLAM – ang. *Simultaneous Localization and Mapping*).

Orientacja metodą „chmura do chmury", czyli iteracyjnego najbliższego punktu (ICP), umożliwia jedynie wzajemną orientację chmur punktów, przy czym orientowane chmury punktów muszą posiadać duże wspólne pokrycie, w którym wyszukiwane są punkty wiążące. Współczynnikiem definiującym dokładność i poprawność orientacji metodą ICP jest odległość pomiędzy orientowanymi chmurami punktów. W przypadku metod iteracyjnych początkowo orientowane są wzajemnie ze sobą dwie chmury punktów, a następnie dopasowywane są do nich kolejne chmury punktów.

Z kolei proces dopasowania chmur punktów, określany mianem *feature – based,* wykorzystuje cechy wykryte w chmurach punktów, takie jak: krzywizny, krawędzie, płaszczyzny, itp.

W celu poprawnej orientacji dużych zbiorów danych pozyskanych z pomiarów znacznych obszarów wskazane jest podzielenie chmur punktów w podobszary robocze, co przyczynia się do wyeliminowania błędów grubych i zminimalizowania przenoszenia się błędów zgodnie z prawem Gaussa. Jako ostatni krok zaleca się całościowe wyrównanie wszystkich obserwacji na podstawie różnych punktów (punktów wiążących, punktów osnowy i wspólnych obszarów). Uwzględnienie dokładności wyznaczenia różnych typów obserwacji (np. wyznaczenia punktów wiążących, pomiaru punktów kontrolnych czy dopasowania metodą ICP), pozwala wykorzystać je jako

wartości współczynników wagowania w ostatecznym procesie wyrównawczym celem poprawy jakości całego procesu wyrównania.

8.1. Orientacja chmur punktów metodą target-based

Orientacja chmur punktów metodą *target – based* opiera się o matematycznym równaniu umożliwiającym wyznaczenie współrzędnych sferycznych. W celu otrzymania jednoznacznego rozwiązania równania, należy wykorzystać 6 obserwacji dla minimum 3 niewspółliniowych punktów. Im większa liczba punktów, tym więcej jest obserwacji nadliczbowych, co wpływa na zwiększenie dokładności wyznaczenia parametrów transformacyjnych. W celu wyznaczenia elementów orientacji zewnętrznej stosuje się metodę najmniejszych kwadratów Gaussa – Markova. Zlinearyzowany układ równań poprawek przedstawia wzór (26):

$$\begin{bmatrix} A_{\rho e} \\ A_{\theta e} \\ A_{\alpha e} \end{bmatrix} \sigma_e + \begin{bmatrix} w_\rho \\ w_\theta \\ w_\alpha \end{bmatrix} = \begin{bmatrix} v_\rho \\ v_\theta \\ v_\alpha \end{bmatrix}, \tag{26}$$

gdzie: $A_{\rho e}, A_{\theta e}, A_{\alpha e}$ – elementy Jakobianu (pochodne względem parametrów ρ, θ, α odniesionych do parametrów orientacji zewnętrznej e), w – wektor przesunięcia, v – wektor poprawek i σ_e – wektor współrzędnych punktów w lokalnym układzie odniesienia.

8.1.1. Orientacja bazująca na manualnym pomiarze punktów wiążących

Do przeprowadzenia orientacji metodą *target-based*, przy wykorzystaniu biblioteki Open3D, służy funkcja (27):

```
#Orientacja metodą Target-based

def wyświetalnie_par_chmur_punktów(chmura_ referencyjna,chmura_orientowana,transformacja):
    ori_temp = copy.deepcopy(chmura_orientowana)
    ref_temp = copy.deepcopy(chmura_referencyjna)
    ori_temp.paint_uniform_color([1, 0, 0])
    ref_temp.paint_uniform_color([0, 1, 0])
    ori_temp.transform(transformacja)
    o3d.visualization.draw_geometries([ori_temp, ref_temp])

def orientacja_target_based(chmura_referencyjna, chmura_orientowana, typ = 'Pomiar',
Debug = 'False'):
    print('Orientacja chmur punktów metoda Target based')
    wyświetalnie_par_chmur_punktów (chmura_referencyjna, chmura_orientowana,
np.identity(4))
    if typ != 'File':
        print('Pomierz min. 3 punkty na chmurze referencyjnej: ')
        pkt_ref = pomiar_punktów_na_chmurze(chmura_referencyjna)
        print('Pomierz min. 3 punkty orientowanej ')
        pkt_ori = pomiar_punktów_na_chmurze(chmura_orientowana)
    elif typ == 'Plik':
        print('Wyznaczenia parametrów transformacji na podstawie punktów pozyskanych z
plików tekstowych')
        #Wczytanie chmur punktów w postaci plików tekstowych
        #Przygotowanie plików ref i ori
    else: #Inna metoda
        print('Wyznaczenie parametrów na podstawie analizy deskryptorów')
        #Analiza deskryptorów
    assert (len(pkt_ref) >= 3 and len(pkt_ori) >= 3)
    assert (len(pkt_ref) == len(pkt_ori))
```

(27)

```
    corr = np.zeros((len(pkt_ori), 2))
    corr[:, 0] = pkt_ori
    corr[:, 1] = pkt_ref
    print(corr)
    p2p = o3d.pipelines.registration.TransformationEstimationPointToPoint()
    trans = p2p.compute_transformation(chmura_referencyjna,
chmura_orientowana,o3d.utility.Vector2iVector(corr))
    if Debug == 'True':
        print(trans)
        wyswetalnie_par_chmur_punktow(chmura_orientowana,chmura_referencyjna,trans)
    analiza_statystyczna(chmura_referencyjna, chmura_orientowana,trans)
    return(trans)
```

8.1.2. Orientacja bazująca na dopasowaniu deskryptorów 3D

W bibliotece Open3D są zaimplementowane dwie metody dopasowania deskryptorów tj. przybliżona metoda porównująca wartości deskryptorów (28) oraz iteracyjna robustowa metoda (uwzględniająca statystyczne podejście doboru punktów) oparta na algorytmie RANSAC (29).

```
#Dopasowanie deskryptorów
def dopasowanie_deskryptorów(chmura_orientowana, chmura_referencyjna, orientowany_fpfh,
referencyjny_fpfh,max_odległość):
    print('Dopasowanie deskryptorów')
    param_orientacji = o3d.pipelines.registration.registration_fast_based_on_feature_matching(
        chmura_orientowana, chmura_referencyjna, orientowany_fpfh, orientowany_fpfh,
        o3d.pipelines.registration.FastGlobalRegistrationOption(liczba_iteracji = 100
            maximum_correspondence_distance= max_odległość))
      return param_orientacji
```

(28)

Przy przybliżoną metodą należy zdefiniować następujące parametry:

- dwie chmury (orientowaną i referencyjną) z wyznaczonymi deskryptorami,
- wartość progu - maksymalną odległość między dopasowywanymi deskryptorami,
- maksymalną liczbę iteracji.

```
#Dopasowanie deskryptorów metoda RANSAC
def dopasowanie_deskryptorów_RANSAC(chmura_orientowana, chmura_referencyjna,
orientowany_fpfh, referencyjny_fpfh, max_odległość):
    param_orientacji =
o3d.pipelines.registration.registration_ransac_based_on_feature_matching(chmura_orientowa
na, chmura_referencyjna, orientowany_fpfh, referencyjny_fpfh, True, max_odległość,
        o3d.pipelines.registration.TransformationEstimationPointToPoint(False),
        3, [o3d.pipelines.registration.CorrespondenceCheckerBasedOnEdgeLength(0.9),
o3d.pipelines.registration.CorrespondenceCheckerBasedOnDistance(distance_threshold)],o3d.
pipelines.registration.RANSACConvergenceCriteria(100000, 0.999))
    return param_orientacji
```

(29)

W przypadku podejścia bazującego na metodzie RANSAC niezbędne jest zdefiniowanie dodatkowych parametrów:

- `CorrespondenceCheckerBasedOnDistance` - parametr pozwalający na sprawdzenie czy odległość między zorientowanymi chmurami punktów spełnia założone parametry dokładności (odległości pomiędzy chmurami punktów),
- `CorrespondenceCheckerBasedOnEdgeLength` - parametr umożliwiający sprawdzenie czy długości dowolnych dwóch krawędzi (linii utworzonej na podstawie dwóch wierzchołków) utworzonych na podstawie chmury punktów referencyjnej i orientowanej są podobne,
- `CorrespondenceCheckerBasedOnNormal` - parametr sprawdzający kąt między normalnymi punktów, jak parametr przyjmuje się wartość kątową w radianach,]
- `RANSACConvergenceCriteria` - liczba iteracji oraz współczynnik ufności.

8.2. Orientacja chmur punktów metodą ICP

Istnieje wiele wariantów algorytmu poszukiwawczego ICP [16–21]. W niniejszym podrozdziale zostaną szerzej opisane najważniejsze z nich.

Rozważając dwa zbiory danych, możliwe jest wyznaczenie między nimi wzajemnych zależności wyrażonych funkcją (30):

$$y_i = Rx_i + y_0, \tag{30}$$

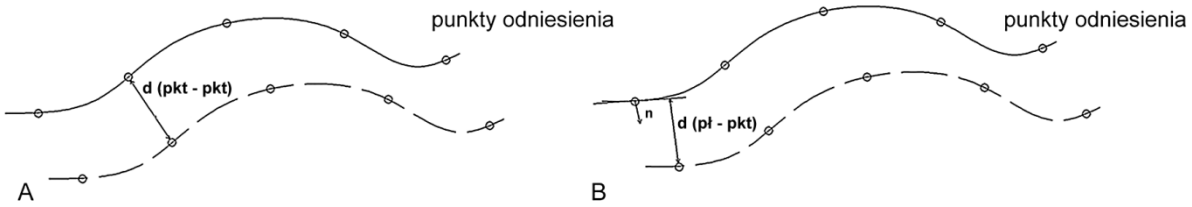
gdzie: R – macierz obrotu, x_i – współrzędne punktu w układzie lokalnym i y_0 – wektor translacji.

Głównym celem metody ICP jest dopasowanie dwóch chmur punktów lub zdefiniowanych kształtów czy modeli do chmury punktów poprzez wykorzystanie zależności odległości euklidesowej, określonej pomiędzy najbliższym punktem ze zbioru punktów opracowywanych a punktem odniesienia. Następnie przy wykorzystaniu funkcji minimalizacji kwadratu odległości (31) obliczane są parametry transformacyjne dla punktów wchodzących w skład obszarów, dla których występuje wspólne pokrycie.

$$e^2 = \sum_i ||Rx_i + y_0 - y_i||^2 \Longrightarrow min \tag{31}$$

W każdej z iteracji algorytmu ICP transformacja może być wyznaczana przy użyciu 4 głównych metod: dekompozycji SVD [22], kwaternionów Horna[23], macierzy ortogonalnych Horna [24] i w oparciu o podwójne kwaterniony Walkera [25]. Algorytmy te charakteryzuje zbliżona skuteczność i stabilność działania w przypadku zaszumionych chmur punktów [26].

W literaturze odnajdujemy również odmienny sposób parametryzacji macierzy obrotów różniący się definicją kątów obrotów. Metoda opracowana przez Chena i Medioniego [27] polega na minimalizowaniu wartości odległości pomiędzy punktami orientowanymi a powierzchniami utworzonymi przez chmury punktów odniesienia. Powierzchnia odniesienia aproksymowana jest lokalnie przez płaszczyznę styczną, zaś mierzona odległość odniesiona jest do płaszczyzny normalnej najbliższego punktu opracowywanego skanu. Zaletą powyższej wersji metody ICP jest możliwość szybszego osiągnięcia zbieżności algorytmu. Schematyczne przedstawienie różnic pomiędzy metodą punktu do punktu i punktu do płaszczyzny zostało przedstawione na rysunku (Rys. 16).



Rys. 16.Metody ICP wykorzystujące odległość: A) punktu do punktu, B) punktu do płaszczyzny odniesienia [15]

Inna koncepcja metody ICP zaproponowana została przez Williamsa i Bennamouna [28]. Zakłada ona jednoczesne wyrównanie większej liczby zbiorów danych oraz wyznaczenie elementów orientacji wzajemnej. Zaproponowany sposób rozwiązania problemu opiera się na

globalnym i odpornym na propagację błędów systematycznych orientowaniu skanów, gdzie chmury punktów łączone są parami.

Metoda wykorzystująca algorytm *least – squares 3D surface matching* [29] polega na dopasowaniu powierzchni odniesienia do powierzchni badanej. Funkcja celu związana jest z szukaniem wartości minimalnej sumy kwadratu odległości, jednakże samo zastosowanie algorytmu *least – squares 3D surface matching* pozwala również na wprowadzenie do wyrównania wag obserwacji ścisłych poprawek ze względu na błędy systematyczne pomiaru TLS.

```
9. #Orientacja chmur punktów metodami ICP
def ICP_registration(source, target, threshold = 1.0, trans_init = np.identity(4), metoda
= 'p2p'):
    print('Analiza dokładności wstępnej orientacji')
10.    evaluation = o3d.pipelines.registration.evaluate_registration(source, target,
threshold, trans_init)
11.    print(evaluation)
12.    if metoda == 'p2p':
        print("Orientacja ICP <Punkt do punktu>")
        reg_p2p = o3d.pipelines.registration.registration_icp(source, target, threshold,
trans_init, o3d.pipelines.registration.TransformationEstimationPointToPoint())
13.        print(reg_p2p)
14.        print("Macierz transformacji:")
15.        print(reg_p2p.transformation)
16.        wyświetalnie_par_chmur_punktów (source, target, reg_p2p.transformation)
               information_reg_p2p =
o3d.pipelines.registration.get_information_matrix_from_point_clouds(source, target,
threshold,reg_p2p.transformation)
17.        return reg_p2p.transformation, information_reg_p2p
    elif metoda == 'p2pl':
        print('Wyznaczanie normalnych')
        source.normals = o3d.utility.Vector3dVector(np.zeros((1, 3))) # Jeżeli istnieją
normalne to są zerowane
        source.estimate_normals()
        target.normals = o3d.utility.Vector3dVector(np.zeros((1, 3))) # Jeżeli istnieją
normalne to są zerowane
        target.estimate_normals()                                                    (32)
        print("Orientacja ICP <Punkt do płaszczyzny>")
        reg_p2pl = o3d.pipelines.registration.registration_icp(
18.            source, target, threshold, trans_init,
19.            o3d.pipelines.registration.TransformationEstimationPointToPlane())
        print(reg_p2pl)
        print("Macierz transformacji:")
        print(reg_p2l.transformation)
        wyświetalnie_par_chmur_punktów (source, target, reg_p2l.transformation)
              information_reg_p2pl =
o3d.pipelines.registration.get_information_matrix_from_point_clouds(
        source, target, threshold,reg_p2pl.transformation)
        return reg_p2l.transformation,information_reg_p2pl
    elif metoda == 'cicp':
        reg_cicp = o3d.pipelines.registration.registration_colored_icp(source, target,
threshold, trans_init)
        print(reg_cicp)
        print("Macierz transformacji:")
        print(reg_cicp.transformation)
        wyświetalnie_par_chmur_punktów (source, target, reg_cicp.transformation)
        information_reg_cicp =
o3d.pipelines.registration.get_information_matrix_from_point_clouds(
        source, target, threshold,reg_cicp.transformation)
        return reg_cicp.transformation, information_reg_cicp
    else:
        print('Nie wybrano odpowiedniego sposobu transformacji')
```

20. Bibliografia

1. PCL PCL Available online: http://www.pointclouds.org/downloads/.
2. Gebhardt, S.; Payzer, E.; Salemann, L.; Fettinger, A.; Rotenberg, E.; Seher, C. Polygons , Point-Clouds , and Voxels , a Comparison of High-Fidelity Terrain Representations. *Fall Simul. Interoperability Work.* **2009**, 1–9.
3. Albon, C. *Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning*; Helion, 2019; ISBN 9781787284395.
4. Kirkpatrick, D.G.; Seidel, R. On the Shape of a Set of Points in the Plane. *IEEE Trans. Inf. Theory* **1983**.
5. Zawieska, D. Wieloobrazowe dopasowanie zdjęć bliskiego zasięgo do automatycznej rekonstrukcji fotorealistycznych modeli 3D. **2013**.
6. Bernardini, F.; Mittleman, J.; Rushmeier, H.; Silva, C.; Taubin, G. The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. Vis. Comput. Graph.* **1999**.
7. Kazhdan, M.; Bolitho, M.; Hoppe, H. Poisson Surface Reconstruction. In Proceedings of the Eurographics Symposium on Geometry Processing (2006); 2006.
8. Ouyang, D.; Feng, H.Y. On the normal vector estimation for point cloud data from smooth surfaces. *CAD Comput. Aided Des.* **2005**.
9. Markiewicz, J.S. Badanie możliwości wykorzystania rastrów intensywności oraz metod dopasowania obrazów w automatycznej orientacji danych z naziemnego skaningu laserowego, Politechnika Warszawska, 2018.
10. Babbar, G.; Bajaj, P.; Chawla, A.; Gogna, M. Comparative study of image matching algorithms. *Int. J. Inf. Technol. Knowl. Manag.* **2010**, *2*, 337–339.
11. Urban, S.; Weinmann, M. Finding a Good Feature Detector-Descriptor Combination for the 2D Keypoint-Based Registration of Tls Point Clouds. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.* **2015**, *II-3/W5*, 121–128.
12. Moussa, W. *Integration of Digital Photogrammetry and Laser Scanning for*; 2006; Vol. 35; ISBN 9783769651379.
13. Markiewicz, J. The use of computer vision algorithms for automatic orientation of terrestrial laser scanning data. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.* **2016**, *41*, 315–322.
14. Yu, Z. Intrinsic shape signatures: A shape descriptor for 3D object recognition. In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009; 2009.
15. Lichti, D.; Pfeifer, N. Introduction to Terrestrial Laser Scanning. **2008**, *12219*.
16. Vosselman, G.; Maas, H.-G. Airborne and Terrestrial Laser Scanning. *Current* 2010, *XXXVI*, 318.
17. Nuchter, A.; Elseberg, J.; Schneider, P.; Paulus, D. Study of parameterizations for the rigid body transformations of the scan registration problem. *Comput. Vis. Image Underst.* **2010**, *114*, 963–980.
18. Bae, K.H.; Lichti, D.D. A method for automated registration of unorganised point clouds.

ISPRS J. Photogramm. Remote Sens. **2008**, *63*, 36–54.

19. Liu, Y. Automatic registration of overlapping 3D point clouds using closest points. *Image Vis. Comput.* **2006**, *24*, 762–781.
20. Theiler, P.W.; Schindler, K. Automatic Registration of Terrestrial Laser Scanner Point Clouds Using Natural Planar Surfaces. *ISPRS Tech. Comm. III, WG III/2* **2005**.
21. Besl, P.; McKay, N. A Method for Registration of 3-D Shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* 1992, *14*, 239–256.
22. Arun, K.S.; Huang, T.S.; Blostein, S.D. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Anal. Mach. Intell.* **1987**, *9*, 698–700.
23. Horn, B.K.P. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A* **1987**, *4*, 629.
24. Horn, B.; Hilden, P., K.; Hugh, M.; Shahriar, N. Closed-form solution of absolute orientation using orthonormal matrices. *J. Opt. Soc. Am. A* **1988**, *5*, 1127.
25. Walker, M.W.; Shao, L.; Volz, R.A. Estimating 3-D location parameters using dual number quaternions. *CVGIP Image Underst.* **1991**, *54*, 358–367.
26. Eggert, D.W.; Lorusso, A.; Fisher, R.B. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Mach. Vis. Appl.* **1997**, *9*, 272–290.
27. Vosselman, G.; Maas, H.-G. Airborne and Terrestrial Laser Scanning. *Current* 2010, *XXXVI*, 318.
28. Williams, J.; Bennamoun, M. Simultaneous Registration of Multiple Corresponding Point Sets. *Comput. Vis. Image Underst.* **2001**, *81*, 117–142.
29. Gruen, A.; Akca, D. Least squares 3D surface and curve matching. *ISPRS J. Photogramm. Remote Sens.* **2005**, *59*, 151–174.