МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В.Г.Шухова»
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Курсовая работа

по дисциплине: «Основы программирования»
по теме: «Поиск затонувшего корабля среди облака точек. Кластеризация облака точек»

Выполнил: студент группы КБ-231

Ушаков Вадим Сергеевич _____
(Подпись)

Проверил:

Лукьянов Александр Михайлович _____
(Подпись)

Белгород 2024 г.

Содержание

1 Введение

В изучении и исследовании исторических артефактов, затонувшие корабли занимают особое место, представляя собой богатый источник информации о прошлом. Поиск таких объектов на морском дне требует применения современных технологий, которые позволяют не только находить, но и детально исследовать затопленные объекты. Одним из инновационных подходов в данной области является использование облака точек — метода, позволяющего создавать трехмерные модели подводного ландшафта.

Актуальность темы обусловлена увеличением доступности технологий лазерного и сонарного сканирования, что создаёт новые возможности для исследования морских глубин. Современные подходы к обработке облаков точек позволяют повышать точность и эффективность поиска затонувших объектов, значительно снижая затраты времени и ресурсов.

Облака точек представляют собой множество данных, полученных с помощью технологии лидарного сканирования или гидролокаторов, которые создают цифровую репрезентацию поверхности морского дна. Этот метод предоставляет возможность высокоточного картографирования подводных объектов, обеспечивая детальное изображение, необходимое для идентификации структур, типичных для затонувших кораблей.

Современные вычислительные алгоритмы играют ключевую роль в обработке и анализе полученных данных. Они позволяют выделить аномалии в облаке точек, которые могут указывать на наличие рукотворных объектов среди морского рельефа. Разработка алгоритмов для автоматизированного поиска и идентификации затонувших кораблей в больших массивах данных становится важной задачей, над решением которой работают многие исследовательские группы.

Таким образом, использование облаков точек представляет собой перспективный метод поиска затонувших кораблей, объединяющий передовые технологии сканирования и обработки данных. Это способствует не только обогащению исторических знаний, но и сохранению культурного наследия подводного мира.

2 Постановка задачи

Целью данной работы является разработка метода и алгоритма для автоматизированного поиска затонувшего корабля среди облака точек, полученных с морского дна. Для достижения этой цели необходимо решить следующие основные задачи:

1. Анализ существующих методов и технологий:
- Провести обзор существующих алгоритмов обработки и анализа облака точек, которые могут быть адаптированы для поиска затонувших объектов.

2. Обработка и предварительная фильтрация данных:
- Разработать методы очистки облаков точек от шумов и артефактов, вызванных особенностями подводного сканирования.

3. Разработка алгоритма поиска объекта:
- Создать алгоритм для выделения затонувщего объекта.

3 Описание решения задачи

Необходимо разработать программу для загрузки, обработки и анализа облака точек с морского дна с целью поиска затонувшего объекта. Для этого используем язык программирования Python.

Язык программирования Python выбран для реализации решения задачи, поскольку он обладает высокой гибкостью, легкостью в освоении и богатыми библиотеками для обработки данных и машинного обучения. Python позволяет разрабатывать алгоритмы обработки облаков точек и легко объединять различные инструменты для анализа и визуализации 3D-данных.

Для реализации функций будем использовать библиотеку Open3D и NumPy.

- Библиотека Open3D используется для обработки, анализа и визуализации облаков точек. Она предоставляет удобные и высокоэффективные инструменты для работы с 3D-геометрией, включая фильтрацию, сегментацию, кластеризацию и визуализацию. Open3D является одной из самых популярных и мощных библиотек для работы с облаками точек, что делает её отличным выбором для задач, связанных с обработкой 3D-данных, таких как анализ морского дна и поиск затонувших объектов.

- Библиотека NumPy используется для эффективной работы с массивами данных и выполнения математических операций над ними. NumPy обеспечивает поддержку многомерных массивов и матриц, а также предлагает множество функций для выполнения стандартных операций, таких как сортировка, индексирование, обработка данных и линейная алгебра. В контексте работы с облаками точек NumPy может использоваться для предварительной обработки данных, например, нормализации координат точек или выполнения математических расчетов.

В рамках данной курсовой работы рассмотрены и реализованы следующие алгоритмы:

- Функция, которая загружает координаты точек из текстового файла, используя NumPy для обработки данных и создания объекта PointCloud из библиотеки Open3D, который представляет облако точек для дальнейшей обработки.

- Функция отвечающая за визуализацию облака точек, что позволяет просматривать и анализировать данные, промежуточные результаты.

- Функция сегментации плоскости, которая частично выделяет точки, принадлежащие основной плоскости, и отделяет их от оставшихся точек, тем самым упрощая дальнейшую работу с облаком точек.

- Функция кластеризации точек и выделения самых больших кластеров, которые могут представлять затонувший корабль.

- Функция, записывающая обработанное облако точек в текстовый файл.

4 Ожидаемый результат

В результате выполнения поставленной задачи ожидается разработка программы, принимающей файл с координатами точек местности затонувшего корабля. Программа будет обрабатывать облако точек и на выходе предоставлять текстовый файл с координатами точек, принадлежащих кораблю.

5 Теоретическая информация

Кластеризация облака точек — это процесс группировки множества точек данных в несколькихмерном пространстве в такие группы (кластеры), что точки внутри одной группы более схожи между собой по некоторому критерию (например, расстоянию), чем с точками из других групп. Цель кластеризации состоит в том, чтобы разделить данные так, чтобы схожие объекты были в одном кластере, а различающиеся — в разных.

5.1 Основные виды кластеризации

1. Кластеризация на основе плотности (Density-Based Clustering)
• Кластеры определяются как области с высокой плотностью объектов.
• Такие алгоритмы могут эффективно выявлять кластеры произвольной формы и игнорировать шум.

2. Иерархическая Кластеризация (Hierarchical Clustering)
• Кластеры формируются в виде иерархической структуры (дерева).
• Существует две основные стратегии:
- Агломеративная: начинается с отдельных точек и поэтапно объединяет их в кластеры.
- Дивизивная: начинается с одного кластера, который делится на меньшие кластеры.

3. Разделяющая Кластеризация (Partitioning Clustering)
• Кластеризация осуществляется путем деления набора данных на непересекающиеся группы (кластеры) заданного числа.
• Объекты в одном кластере более схожи между собой, чем с объектами других кластеров.

4. Чёткая (жёсткая) кластеризация (Hard Clustering
• Каждый объект относится строго к одному кластеру.

5. Нечёткая (мягкая) кластеризация (Fuzzy Clustering)
• Объекты могут принадлежать нескольким кластерам с определёнными степенями принадлежности.
• Позволяет более гибко подходить к задаче, что может быть полезно в случаях, когда границы между кластерами размыты.

5.2 Основные алгоритмы кластеризации

1. K-средние (k-means):
• Простой и широко используемый метод.
• Делит данные на k кластеров, минимизируя внутреннюю дисперсию.
• Требует заранее заданного числа кластеров k.

2. Иерархический кластерный анализ (Hierarchical clustering):
• Не требует заранее заданного числа кластеров.
• Создаёт дерево кластеров (дендрограмму).

3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
• Обнаруживает кластеры произвольной формы.
• Выявляет шумы в данных.
• Не требует заранее заданного числа кластеров.

4. Спектральная кластеризация:
• Работает с графами.
• Подходит для сложных структур и нелинейных границ.

5.3 Применение кластеризации для поиска затонувшего корабля

При задаче поиска затонувшего корабля среди облака точек, кластеризация помогает:
• Выделение область интереса: Области с высокой плотностью точек могут свидетельствовать о наличии объектов, таких как корабли.
• Фильтрация шумов: Методы, такие как DBSCAN, могут помочь исключить шумы, идентифицируя их как незначительные группы точек вдали от плотных регионов.
• Анализ структуры: Кластеризация может выявить структурные паттерны, которые помогают предсказать местоположение объектов в сложных подводных ландшафтах.

Таким образом, кластеризация позволяет эффективно обрабатывать и анализировать большие массивы данных, упрощая задачу поиска затонувшего корабля.

6 Основная часть

6.1 Функции для работы с файлом, содержащим облако точек.

В этом разделе подробно рассмотрим функции, предназначенные для работы с файлами, содержащими данные об облаках точек. Такие файлы, содержат координаты точек в трехмерном пространстве, и для эффективного их использования в анализе и визуализации требуется преобразование их в структурированные форматы, такие как массивы и объекты Open3D.

Как и планировалось изначально, для выполнения работы подключим библиотеки NumPy и Open3D.

```
1 import open3d as o3d
2 import numpy as np
```

Далее напишем функцию load_coordinates_from_txt для загрузки координат точек из текстового файла и создания объекта облака точек с использованием библиотеки Open3D.

Основные этапы работы функции включают:

1. **Чтение данных из файла.** Функция принимает в качестве аргумента путь к текстовому файлу, содержащему координаты точек. Для открытия файла используется контекстный менеджер with, обеспечивающий правильное закрытие файла после завершения работы с ним.

2. **Обработка строк файла.** Каждая строка файла очищается от лишних пробелов и символов новой строки с помощью метода strip(). Далее строки разделяются на отдельные координаты с использованием метода split(','), а каждый элемент преобразуется в число с плавающей запятой.

3. **Сохранение координат в список.** Преобразованные координаты добавляются в список, который в конечном итоге используется для создания массива NumPy.

4. **Создание и настройка объекта PointCloud.** На базе массива координат создается объект PointCloud из библиотеки Open3D. Координаты точек сохраняются в объекте посредством структуры Vector3dVector, что обеспечивает удобные методы для последующей обработки и визуализации данных.

5. **Возврат результата.** Функция возвращает объект облака точек, который может быть использован для визуализации, анализа, и других операций, связанных с пространственными данными.

Таким образом, функция будет выглядеть следующим образом:

```
# создает облако точек используя файл txt
def load_coordinates_from_txt(file_path):

    # Создание пустого списка для хранения координат
    points = []

    # Открываем файл для чтения
    with open(file_path, 'r') as file:

        for line in file:

            # Убираем лишние пробелы и символы новой строки
            line = line.strip()

            # Разбиваем строку по запятым
            if line:
                x, y, z = map(float, line.split(','))
                points.append([x, y, z])

    # Преобразуем список точек в массив numpy
    points = np.array(points)

    # Создаем объект PointCloud из библиотеки open3d
    point_cloud = o3d.geometry.PointCloud()
    point_cloud.points = o3d.utility.Vector3dVector(points)

    return point_cloud
```

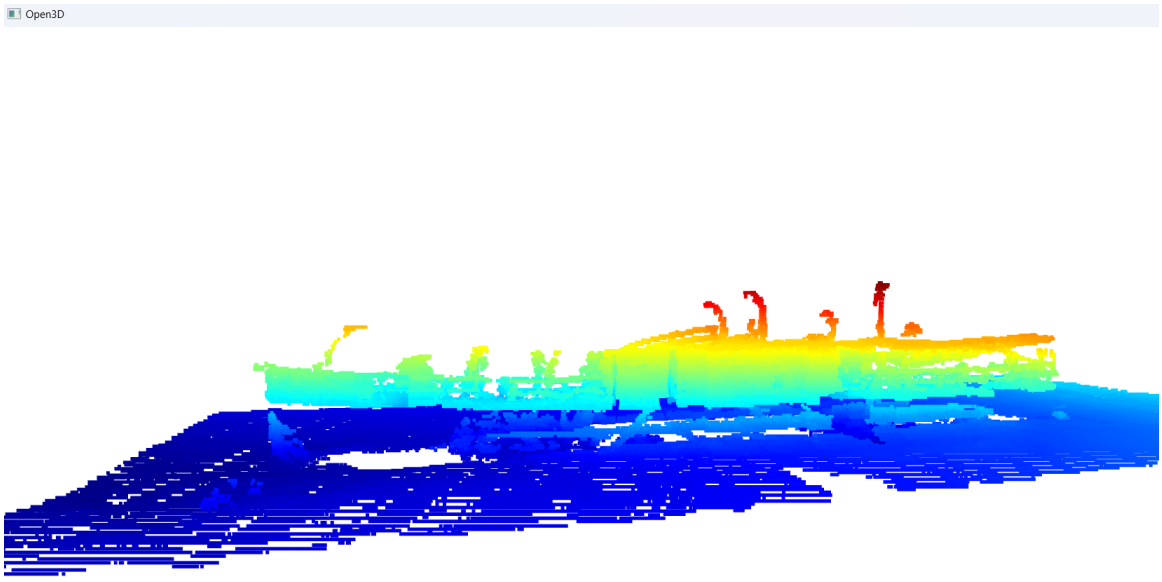
Для проверки написанных функций и анализа результатов напишем простую функцию для визуализации облака точек. Для этого используем уже существующий модуль visualization из библиотеки Open3D.

```
# функция визуализации облака точек
def visualize_point_cloud(point_cloud):
    o3d.visualization.draw_geometries([point_cloud])
```

Проверим написанные функции. Считаем исходный файл с помощью функции load_coordinates_from_txt и выведем содержимое, используя функцию visualize_point_cloud.

```
# Входной файл
file_path = "wreck1.txt"

# Загружаем облако точек из файла
point_cloud = load_coordinates_from_txt(file_path)

# Визуализация облака точек из файла
visualize_point_cloud(point_cloud)
```

Результат работы программы:



Напишем функцию save_point_cloud_to_txt, предназначенную для сохранения облака точек в текстовый файл.

Основные этапы работы функции включают:

1. **Извлечение массива из объекта PointCloud.** Функция принимает в качестве аргумента объект point_cloud, содержащий трёхмерные координаты точек. Используется метод np.asarray() для извлечения массива точек из атрибута points объекта PointCloud.

2. **Открытие файла для записи.** Используется контекстный менеджер with для открытия выходного файла в режиме записи ('w'), что гарантирует автоматическое закрытие файла после завершения блоков записи; это уменьшает вероятность ошибки и упрощает обработку файловых операций.

3. **Запись данных в файл.** В цикле for итерация осуществляется по массиву точек. Для каждой точки производится форматирование координат в строку вида x,y,z, где координаты разделены запятыми. Запись в файл осуществляется с помощью метода write().

4. **Формирование строк с координатами.** В цикле записи каждая координата x, y, z формируется в строки формата 'x,y,z'. Эти строки последовательно записываются в открытый файл, каждая с новой строки.

5. **Закрытие файла и завершение работы.** После завершения всех операций записи файл автоматически закрывается благодаря контекстному менеджеру with, и функция завершает свою работу. Таким образом, создаётся текстовый файл, где каждая строка представляет координаты одной точки, что делает данные удобными для последующего использования в других приложениях.

Таким образом, функция будет выглядеть следующим образом:

```
# Функция для сохранения облака точек в txt файл
def save_point_cloud_to_txt(point_cloud, output_file_path):

    # Получаем массив numpy из объекта PointCloud
    points = np.asarray(point_cloud.points)

    # Открываем файл для записи
    with open(output_file_path, 'w') as file:

        # Записываем каждую точку в строку в формате x, y, z
        for point in points:
            file.write(f"{point[0]},{point[1]},{point[2]}\n")
```

6.2 Функция сегментации плоскости.

Используем сегментацию плоскости перед кластеризацией облака точек затонувшего корабля. Это необходимо для частичного удаления плоскости морского дна, что позволит избавиться от значительной части лишних точек. Благодаря сегментации можно уменьшает объем данных, что влияет на скорость и эффективность алгоритма кластеризации.

Используем метод RANSAC для выделения плоскости из облака точек.

Основные этапы работы функции включают:

1. **Входные параметры:**
   - point_cloud: Исходное облако точек.
   - distance_threshold: Порог расстояния, указывающий максимальное допустимое отклонение точек от предполагаемой плоскости для того, чтобы их рассматривать как принадлежащие плоскости.
   - ransac_n: Количество точек, случайно выбираемых для оценки модели плоскости в каждой итерации.
   - num_iterations: Число итераций алгоритма для нахождения наилучшей модели плоскости.

2. **Выбор точек:** На каждой итерации случайно выбираются точки для оценки модели плоскости.

3. **Оценка модели:** Параметры плоскости вычисляются и проверяются на соответствие: если точки в заданных пределах, они считаются соответствующими.

4. **Оптимизация:** Процесс повторяется заданное кол-во раз для нахождения плоскости с максимальным числом соответствующих точек.

5. **Выходные данные:** Функция возвращает два облака точек:
   - inlier_cloud: точки, принадлежащие плоскости.
   - outlier_cloud: остаток точек.

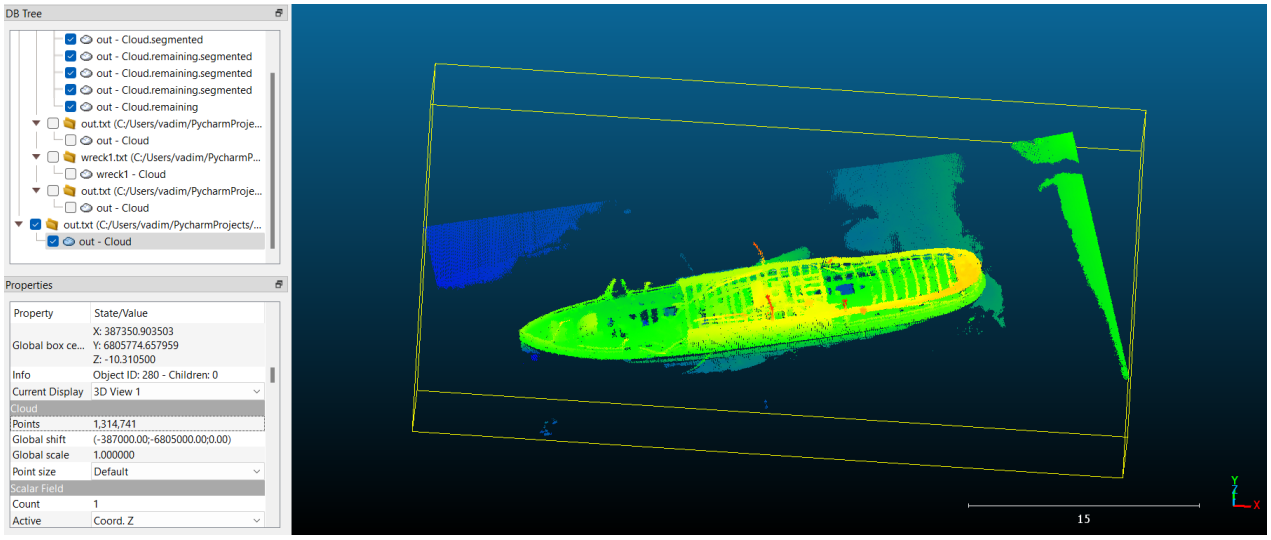
Таким образом, функция примет следующую форму:

```
# Функция для сегментации плоскости из облака точек
def segment_plane(point_cloud, dist_th=0.2, ran_n=10,
    num_iter=100):
    plane_model, inliers = point_cloud.segment_plane(distance
    _threshold=dist_th,ransac_n=ran_n,num_iter=num_iter)

    inlier_cloud = point_cloud.select_by_index(inliers)
    outlier_cloud = point_cloud.select_by_index(inliers,
    invert=True)

    return inlier_cloud, outlier_cloud
```

Проверим написанную функцию.

```
# Выходной файл
output_file = "out.txt"

# Входной файл
file_path = "wreck1.txt"

# Загружаем облако точек из файла
point_cloud = load_coordinates_from_txt(file_path)

# Сегментация плоскости
inlier_cloud, outlier_cloud = segment_plane(point_cloud)

# Сохранени облака в файл
save_point_cloud_to_txt(outlier_cloud, output_file)
```

Откроем записанный выходной файл в программе CloudCompare для более детального анализа.



Можно заметить, что значительная часть точек, относящихся к дну, была удалена. Изначально в файле было 3.5 миллиона точек. После сегментации их количество уменьшилось на 63%, как видно на скриншоте выше, осталось примерно 1.3 миллиона точек. Это значительно ускорит дальнейшую обработку облака.

6.3 Функция кластеризации.

Напишем заключительную функцию кластеризации, которая окончательно выделит затонувший объект. Используем алгоритм DBSCAN.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) — это алгоритм кластеризации, который группирует точки, находящиеся близко друг к другу, и помечает точки, находящиеся в области низкой плотности, как шум.

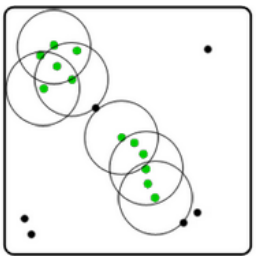
Достоинство данного алгоритма в том, что не нужно указывать количество кластеров, он делает это сам. Кластеры могут иметь произвольную форму, а данные могут поддерживать шум. Нам нужно будет всего лишь подобрать подходящие параметры.

Параметры, необходимые для алгоритма DBSCAN:
1. eps: Определите окрестность вокруг точек данных, то есть если расстояние между двумя точками меньше или равно «eps», то они считаются соседними. Если значение eps выбрано слишком маленьким, большая часть данных будет считаться выбросами. Если он выбран слишком большим, то кластеры сольются, и большинство точек данных будут находиться в одних и тех же кластерах. Один из способов найти значение eps основан награфе к-расстояний.

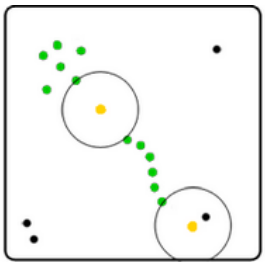
2. МинПтс: минимальное количество соседей (точка данных) в пределах радиуса eps. Чем больше набор данных, тем большее значение необходимо выбрать MinPts. Как правило, минимальное значение MinPts можно вычислить по количеству измерений D в наборе данных как MinPts >= D+1. МинИмальное значение MinPts должно быть не менее 3.

Наглядный пример работы алгоритма.

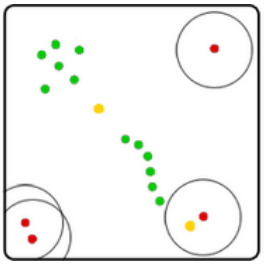
1. Алгоритм берет точку и строит от нее буфер указанного радиуса. Если в буфер попадает количество точек больше, чем минимальное количество точек в радиусе, то эта точка становится корневой и от нее строится новый кластер. Так выбираются все корневые точки.



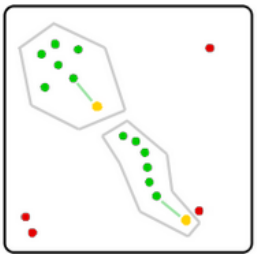
2. Далее алгоритм находит точки, у которых в буфере меньше заданного количества соседей, но есть хотя бы одна корневая точка. Эти точки становятся пограничными.



3. Остались точки, в буфере от которых меньше указанного числа соседей и нет корневых элементов. Эти точки будут считаться выбросами.



4. Если два корневых элемента находятся рядом, то они объединяются в один кластер. Пограничные элементы будут отнесены к группе корневого элемента из своей окрестности.



5. Процесс завершается, когда ни к одному кластеру не может быть добавлено ни одного нового объекта.

Основные этапы работы функции включают:

1. **Применение алгоритма DBSCAN:**
   - Функция получает облако точек (point_cloud), и применяет к нему алгоритм кластеризации DBSCAN.
   - Параметры eps и min_points задают радиус окрестности точки и минимальное количество точек в окрестности соответственно.
   - Результатом является массив меток labels, который содержит идентификаторы кластеров для каждой точки. Точки, не попавшие ни в один кластер, помечаются как шум.

2. **Определение количества кластеров:**
   - Определяется максимальный идентификатор метки кластера (max_label), который показывает общее количество кластеров.
   - Выводится на экран количество кластеров, используя max_label + 1 (поскольку метки начинаются с 0).

3. **Инициализация для поиска самого большого кластера:**
   - Переменные largest_cluster_size и largest_cluster_cloud инициируются для хранения информации о размере самого большого кластера и его данных.

4. **Обработка каждого кластера:**
   - Проходится цикл по каждому кластеру (от 0 до max_label).
   - Определяются индексы точек, принадлежащие текущему кластеру, с помощью np.where(labels == i).
   - Эти точки извлекаются из облака точек с помощью select_by_index и называются cluster_cloud.
   - Рассчитывается размер текущего кластера (cluster_size), и выводится сообщение с количеством точек в этом кластере.

5. **Обновление информации о самом большом кластере:**
   - Если текущий кластер больше по размеру, чем предыдущий самый большой кластер, информация о нем обновляется.

6. **Проверка и вывод информации о самом большом кластере:**
   - Если был найден хотя бы один кластер, то выводится информация о самом большом кластере.
   - Если кластеры не найдены, то выводится соответствующее сообщение.

7. **Возврат результатов:**
   - Функция возвращает облако точек самого большого кластера и текстовое описание (cluster_info), завершающее выполнение функции.

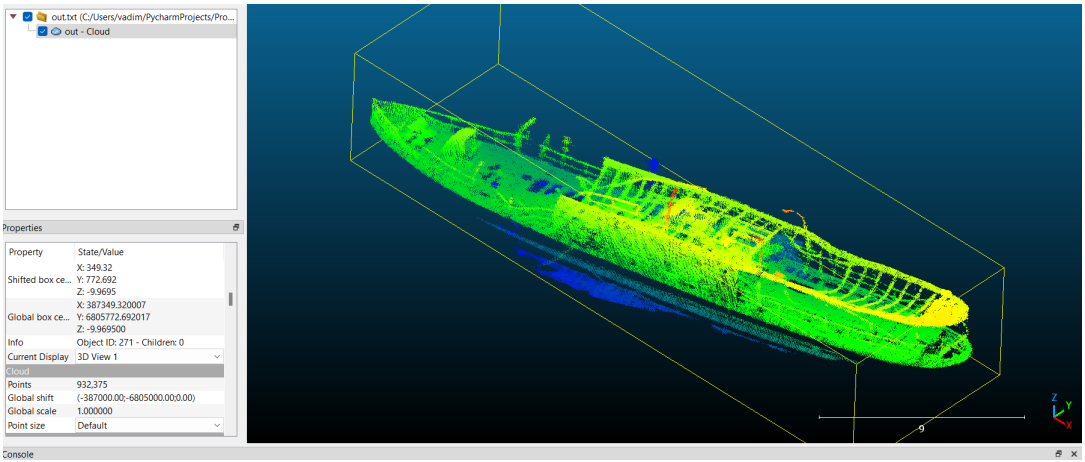
Таким образом, функция будет выглядеть следующим образом:

```
# Функция для кластеризации облака точек
def cluster_point_cloud(point_cloud, eps=0.3, min_points=100):
    # Применяем DBSCAN для кластеризации
    labels = np.array(point_cloud.cluster_dbscan(eps=eps,
    min_points=min_points, print_progress=True))

    # Находим количество кластеров
    max_label = labels.max()
    print(f"Количество кластеров: {max_label + 1}")

    # Инициализация для хранения данных о самом большом кластере
    largest_cluster_size = 0
    largest_cluster_cloud = None

    # Проходим по каждому кластеру
    for i in range(max_label + 1):
        # Выбираем точки, принадлежащие текущему кластеру
        cluster_indices = np.where(labels == i)[0]
        cluster_cloud =
    point_cloud.select_by_index(cluster_indices)

        # Оцениваем размер кластера
        cluster_size = len(cluster_indices)
        print(f"Кластер {i}: {cluster_size} точек")

        # Обновляем, если текущий кластер больше предыдущего
        if cluster_size > largest_cluster_size:
            largest_cluster_size = cluster_size
            largest_cluster_cloud = cluster_cloud

    # Проверяем наличие самого большого кластера и создаем описание
    if largest_cluster_cloud is not None:
        cluster_info = f"Самый большой кластер содержит
    {largest_cluster_size} точек."
        print(cluster_info)
    else:
        cluster_info = "Кластеры не найдены."
        print(cluster_info)

    return largest_cluster_cloud, cluster_info
```

Сохраним файл с самым большим кластером.

```
# Выходной файл
output_file = "out.txt"

# Входной файл
file_path = "wreck1.txt"

# Загружаем облако точек из файла
point_cloud = load_coordinates_from_txt(file_path)

# Сегментация плоскости
inlier_cloud, outlier_cloud = segment_plane(point_cloud)

# Кластеризация
largest_cluster, cluster_info = cluster_point_cloud(outlier_cloud)

# Сохранени облака в файл
save_point_cloud_to_txt(largest_cluster, output_file)

# Вывод информации о кластере
#print(cluster_info)

#Визуализация самого большого кластера
#if largest_cluster is not None:
    #visualize_point_cloud(largest_cluster)
#else:
    #print("Нет кластера для визуализации.")
```

Откроем записанный выходной файл в программе CloudCompare.



Задача выполнена успешно, корабль найден среди облака точек.

7 Заключение

8 Список использованной литературы

Кластеризация пространственных данных – плотностные алгоритмы и DBCSAN

Задачи кластеризации

Кластеризация DBSCAN в ML | кластеризация на основе плотности

кластеризация по плотности основы

Обработка облака точек

Осваиваем анализ лидарных данных и измеряем дорожные знаки

разработка методик обработки результатов сканирования