МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет»

Инженерная школа

Кафедра автоматизации и управления

Старков Артем Дмитриевич

РЕАЛИЗАЦИЯ АЛГОРИТМА ITERATIVE CLOSEST POINT

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

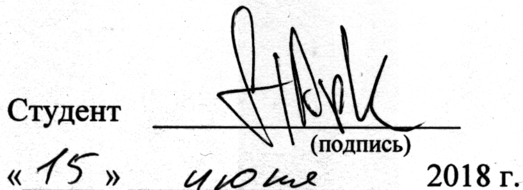
по направлению подготовки бакалавров

15.03.06 - Мехатроника и робототехника

профиль «*Мехатроника и робототехника*»

г. Владивосток

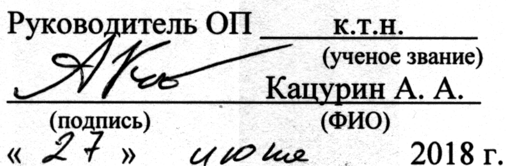
2018



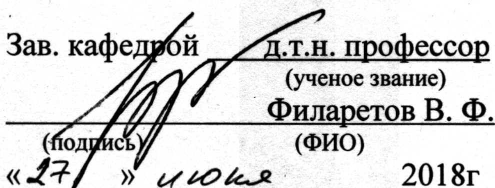
Студент ____________________
(подпись)
«_15_» ____июня____ 2018 г.

Руководитель квалификационной выпускной работы (проекта) ____к.т.н.____
(должность, ученое звание)
____________________ Губанков А.С.
(подпись) (ФИО)
«_15_» ____июня____ 2018 г.

«Допустить к защите»

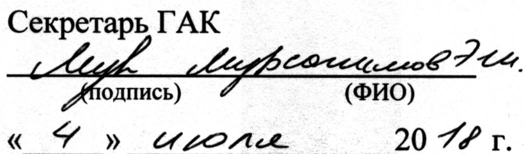


Руководитель ОП ____к.т.н.____
(ученое звание)
____________ Кацурин А. А.
(подпись) (ФИО)
«_27_» ____июня____ 2018 г.



Зав. кафедрой ____д.т.н. профессор____
(ученое звание)
____________ Филаретов В. Ф.
(подпись) (ФИО)
«_27_» ____июня____ 2018г

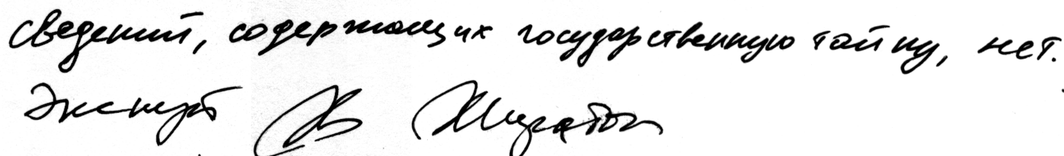
Защищена в ГАК с оценкой ____отлично____



Секретарь ГАК
____________ Мирсалимова Э.М.
(подпись) (ФИО)
«_4_» ____июля____ 20_18_г.

В материалах данной выпускной квалификационной работы не содержатся сведения, составляющие государственную тайну, и сведения, подлежащие экспертному контролю.

Директор ИШ ДВФУ А.Т. Беккер



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет»

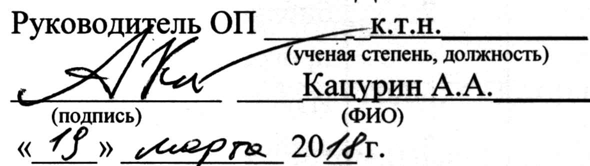
Инженерная школа

Кафедра автоматизации и управления

УТВЕРЖДЕНО

Руководитель ОП к.т.н.

(ученая степень, должность)



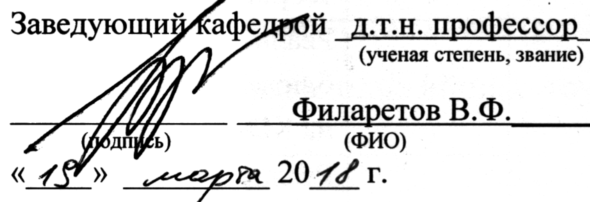
Кацурин А.А.

(подпись) (ФИО)

«19» марта 2018 г.

Заведующий кафедрой д.т.н. профессор

(ученая степень, звание)



Филаретов В.Ф.

(подпись) (ФИО)

«19» марта 2018 г.

З А Д А Н И Е

на выпускную квалификационную работу

Студенту **Старкову Артему Дмитриевичу** Группа **Б3421**

(Фамилия, Имя, Отчество) (номер группы)

1. Наименование темы **Реализация алгоритма Iterative closest point**

2. Основания для разработки **Выполнение НИР**

3. Источники разработки

4. Технические требования (параметры) **Время совмещения для тестовых данных не более 2 минут**

5. Дополнительные требования **Отсутствуют**

6. Перечень разработанных вопросов **Поиск ближайших точек, сингулярное разложение матриц, ICP алгоритм**

7. Перечень графического материала (с точным указанием обязательных плакатов) **Презентация**

КАЛЕНДАРНЫЙ ГРАФИК ВЫПОЛНЕНИЯ РАБОТЫ

№ п/п	Наименование этапов дипломного проекта (работы)	Срок выполнения этапов дипломного проекта (работы)	Примечание
1	Ознакомление с заданием и алгоритмом iterative closest point	15.02.18-25.02.18	Изучение принципов работы алгоритма и его приложений
2	Выбор метода оптимизации и библиотек для построения программы.	26.02.18-05.03.18	Обзор существующих решений
3	Изучение PCL библиотеки и SVD алгоритма. Описание теоретической части.	08.03.18-15.03.18	Изучение документации
4	Реализация алгоритма поиска ближайших точек основанного на KD-деревьях.	16.03.18-27.03.18	
5	Реализация SVD алгоритма при помощи средств библиотеки Eigen и PCL.	03.04.18-16.04.18	
6	Реализация основной части ICP алгоритма.	17.04.18-22.04.18	
7	Реализация методов визуализации и чтения облаков точек. Тестирование.	24.04.18-05.05.18	
8	Написание и оформление ПЗ. Получение отзыва руководителя.	12.05.18-18.06.18	
9	Защита ВКР	04.07.18	

Дата выдачи задания 15.03.2018г

Срок представления к защите июнь 2018 года

Руководитель ВКР _______ (подпись) Губанков А.С. (ФИО)

Студент _______ (подпись) Старков А.Д. (ФИО)

АННОТАЦИЯ

Работа направлена на практическое применение известных методов оптимизации и работы с данными в таких областях как робототехника. Целью было реализовать итеративный алгоритм ближайших точек (Iterative closest point, ICP), а также использовать его для совмещения облаков точек.

В основе выбранного алгоритма совмещения лежат методы, актуальность которых растёт с каждым годом, особенно в сфере автоматизации. Издано множество работ по математическому программированию и по алгоритму ICP в частности, в которых совершенствуются методы поиска ближайших точек и методы совмещения.

Алгоритм ICP поочерёдно производит поиск ближайших точек и оптимальное преобразование между двумя наборами данных для их совмещения. Основной трудностью данного подхода является чувствительность к выбросам и недостаточное количество данных. Большинство практических реализаций ICP алгоритма решают эту проблему с помощью коррекции весов [6]. Однако эти методы могут быть ненадежными и трудными, что часто требует существенной ручной настройки.

Основной задачей рассматриваемого алгоритма является поиск аргументов, которые минимизируют критерий совмещения, переводя множество точек облака $\mathcal{S}$ в множество точек облака $\mathcal{T}$.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

САПР − система автоматизированного проектирования

ICP – итеративный алгоритм ближайших точек (iterative closest point)

KD дерево – дерево k-мерного пространства

SVD – сингулярное разложение (singular value decomposition)

PCL – библиотека облака точек (point cloud library)

PCD – формат облака точек (point cloud data)

ВВЕДЕНИЕ

Современная промышленность нуждается в новых методах и подходах автоматизации производства, способных работать в режиме реального времени с приемлемыми показателями качества. Одним из важнейших этапов изготовления изделия является получение его математической модели. Рано или поздно возникает необходимость описания реального физического объекта с помощью математического аппарата для последующей обработки в различных пакетах САПР.

Для автоматизации и ускорения процесса перевода поверхности модели в цифровой формат используется трёхмерное сканирование. Во время построения цифровых моделей появляется необходимость сканирования объекта с нескольких ракурсов в связи с перекрытием некоторых частей детали и ограниченным диапазоном датчиков. Результат сканирования представляет собой поверхность объекта, которая описана с помощью облака точек. Полученный скан и модель должны быть выровнены в общую систему отсчёта для дальнейшей обработки, чтобы иметь возможность измерять и классифицировать возможные ошибки в производстве.

В мобильной робототехнике сопоставление скана сцены с её моделью обычно используется для уточнения местоположения. В последнее время устройства 3D-сканирования (например, Microsoft Kinect) привели к растущему интересу в области надежных алгоритмов совмещения. Низкая стоимость этих устройств сбора данных обеспечивается за счет плохого качества сканирования, что требует алгоритмов, которые способны иметь дело с большими объемами шума и выбросов.

Алгоритм ближайших точек имеет широкое применение в промышленной робототехнике. Он позволяет решать задачи позиционирования манипулятора во время обработки детали. Для того, чтобы выработать необходимый управляющий сигнал и обработать деталь по заданной траектории, необходима

её CAD модель. Однако модель находится в некоторой системе координат и вычислить траекторию обработки детали можно только в системе координат модели. Поэтому обработать деталь таким образом возможно лишь закрепив её в заранее известной точке.

Лазерное сканирование позволяет решить эту проблему. Благодаря сканеру и итеративному алгоритму ближайших точек возможно вычислить траекторию движения схвата манипулятора в его системе координат и обработать произвольно закреплённую деталь. Описанная выше задача решается путём вычисления матрицы преобразования из системы координат робота в систему координат модели. Зная эту матрицу, можно трансформировать траекторию движения схвата в системе координат модели в аналогичную траекторию в системе координат робота и обработать реальную произвольно расположенную деталь.

1 ОБЩЕЕ ОПИСАНИЕ РАБОТЫ АЛГОРИТМА

Пусть $\mathcal{T} = \{\tau_1, \tau_2, \cdots \tau_N\}$ и $\mathcal{S} = \{s_1, s_2, \cdots s_N\}$ – два множества точек в конечномерном вещественном векторном пространстве $\mathbb{R}^n$.

Необходимо найти преобразование множества точек $\mathcal{S}$ во множество точек $\mathcal{T}$ с помощью матрицы поворота R и вектора смещения t, зависимость которых представлена в виде:

$$\tau_i = Rs_i + t. \tag{1}$$

где $i = 0,1, \cdots, n$.

1.1 Основные этапы работы алгоритма

Итеративный алгоритм ближайших точек можно условно разбить на четыре этапа [3]:

а) Нахождение соответствующей ближайшей пары τ_j для точки s_i, такой, что:

$$S(\tau) = \left\|\tau_j - s_i\right\|_2, \tag{2}$$

$$(\tau) = \underset{s_i \epsilon \mathcal{S},\, \tau_j \epsilon \mathcal{T}}{\operatorname{argmin}} S(\tau). \tag{3}$$

Под s_i здесь понимается заданная точка в пространстве $\mathbb{R}^3$, для которой необходимо найти кратчайшее расстояние до множества $\mathcal{T}$. То есть, найти такую точку τ_j, которая будет доставлять наименьшее значение функционалу (2).

b) Вычисление вектора смещения t и матрицы поворота R, которые доставляют минимум функционалу:

$$\mathcal{J}(R,t) = \sum_{i=1}^{N} \|(Rs_i + t) - \tau_i\|_2^2, \quad (4)$$

$$(R,t) = \underset{R\in SO(3),t\in R^3}{\operatorname{argmin}} \mathcal{J}(R,t). \quad (5)$$

Матрица R представляет собой вещественную ортогональную матрицу 3×3 с определителем равным единице, то есть принадлежит специальной ортогональной группе вращений $SO(3)$;

Вектор t представляет собой смещение множества точек вдоль вектора $[x,y,z]^T$;

$\|\cdot\|_2^2$ − квадрат нормы l_2 или квадрат Евклидового расстояния между двумя точками.

c) Преобразование трансформируемого облака точек с помощью найденной матрицы поворота и вектора смещения в новое облако точек

$$s_i = Rs_i + t. \quad (6)$$

d) Повторение всего итерационного процесса алгоритма пока $\mathcal{J}(R,t) \geq \varepsilon$, где в качестве трансформируемого облака точек теперь используется облако, полученное на предыдущем этапе.

1.2 Поиск ближайших точек

Рассмотрим два идентичных облака точек, которые смещены друг относительно друга. Каждая точка пронумерована и верно, что две точки из различных облаков с одинаковым индексом имеют идентичное положение относительно совмещаемых объектов. Очевидно, что совместить облака точек, означает совместить все пары точек с равными индексами из различных множеств. Другими словами, две поверхности должны совпадать, а все точки из совмещаемого множества должны принадлежать поверхности целевого облака точек.

Рассмотренная идеальная модель обычно не имеет ничего общего с практикой, потому что точки с равными индексами не соответствуют идентичному положению относительно совмещаемых объектов. Такую задачу можно решить, прибегая к геометрическому соображению, которое заключается в том, что расстояние между точкой и поверхностью определяется как кратчайшее расстояние между ними. Сократив это расстояние, точка будет принадлежать поверхности, что и является целью совмещения.

Из всего выше сказанного следует вывод, что для того, чтобы совместить два облака точек, необходимо сокращать расстояния между всеми парами ближайших точек. Чтобы ускорить этот процесс необходимо отсортировать точки целевого облака с помощью геометрической интерпретации бинарного дерева – KD-дерева [5].

KD-дерево представляет собой двоичное дерево поиска, где данные в каждом узле являются K-мерной точкой в пространстве. Смысл построения дерева заключается в пространственном разбиении структуры данных для организации точек в пространстве.

Не листовой узел в дереве делит пространство на две части, называемые полупространствами. Точки левого полупространства представлены левым поддеревом этого узла, а точки правого полупространства представлены правым поддеревом. Другими словами, если необходимо найти ближайшую точку в дереве к некоторой заданной точке, расположенной в левом полупространстве, следует спуститься в левое поддерево, тем самым исключая необходимость перебора всех точек. Далее, аналогичным образом, сравнивая по второй координате, можно перейти в верхнее или нижнее полупространство левого полупространства.

На каждом уровне дерева сравнение происходит попеременно по каждой координате (рисунок 1). Что позволяет произвести сортировку в любом возможном направлении, переходя на всё меньшую область, в которой содержится одна точка (лист).

Для демонстрации построения 3D-дерева целесообразно построить 2D-дерево, по причине сложности графического представления и принципиальной схожести процесса. В трёхмерном случае все операции производятся аналогичным образом с добавлением координаты z.

Рассмотрим построение двумерного дерева [2, 7] для набора следующих точек: (3, 6), (17, 15), (13, 15), (6, 12), (10, 19), (9, 1), (2, 7).

Алгоритм построения:

а) Вставить первую точку (3, 6): поскольку дерево пустое, сделать её корнем;
б) Вставить вторую точку (17, 15): сравнить координату x этой точки с координатой x точки в корне. Так как $17 > 3$, то точка идёт в правое поддерево;
в) Вставить точку (13, 15): сравнить координату x этой точки с координатой x точки в корне. Так как 13 > 3, то точка идёт в правое поддерево. Далее сравнить координату y этой точки с координатой y точки в правом поддереве. Так как $15 = 15$, то точка идёт в правое поддерево текущего корневого узла;
г) Вставить точку (6, 12): так как 6 > 3, то точка идёт в правое поддерево. Так как 12 < 15, то точка будет лежать в левом поддереве корня (17, 15);
д) Повторить аналогичные действия и добавить оставшиеся точки.

Результат представлен на рисунке 1.

1.3 Геометрическая интерпретация KD-дерева

Представленный выше алгоритм сортировки имеет следующий геометрический смысл [7].

На первом этапе точка (3, 6) делит пространство вертикальной прямой на две части. Все точки, которые меньше (левее) по координате x, окажутся в левом поддереве. Все точки, которые больше (правее) – в правом (рисунок 2).

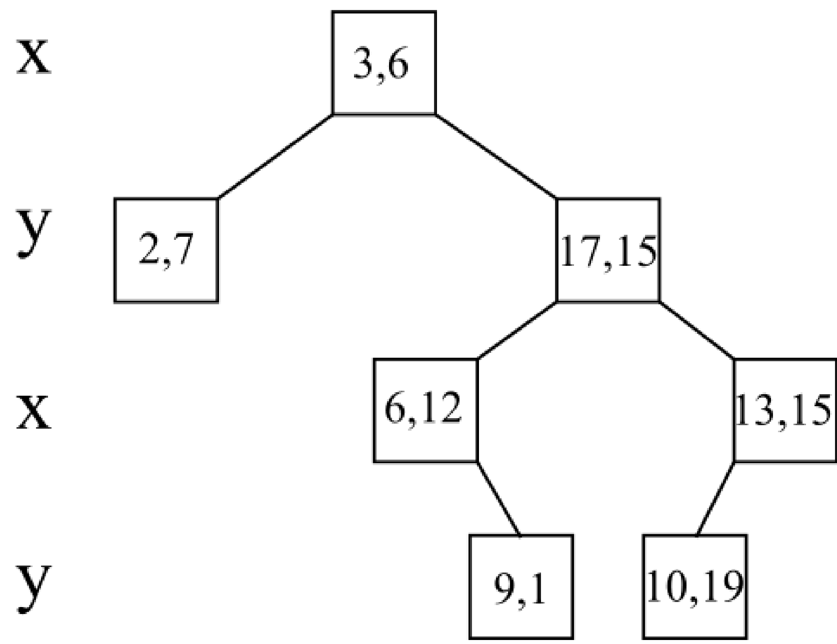


Рисунок 1 – 2D-дерево

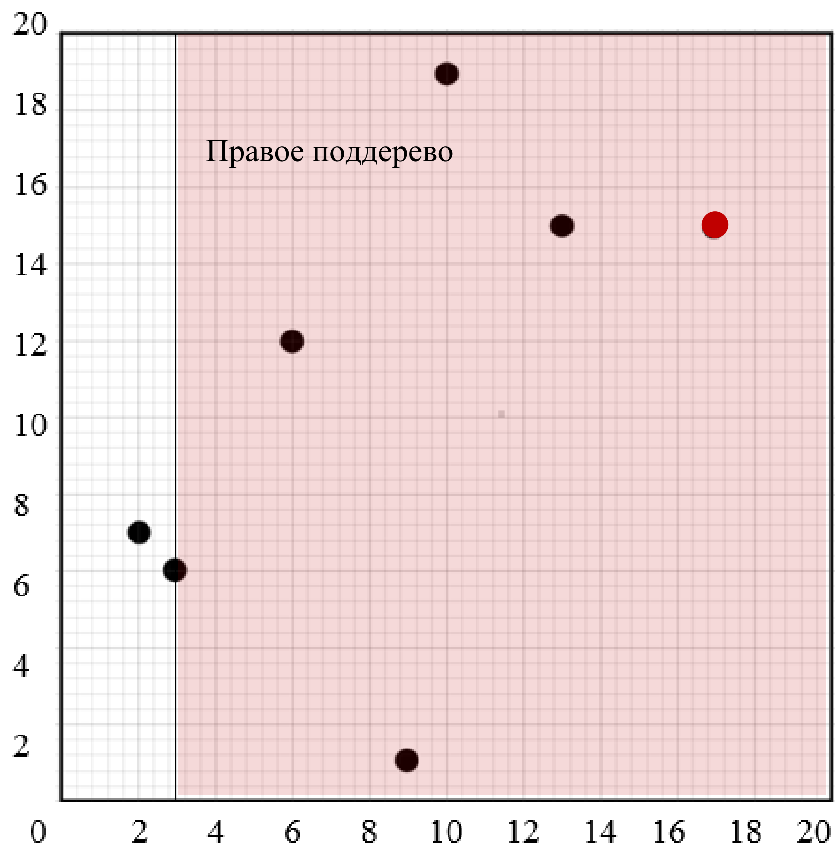


Рисунок 2 – Правое поддерево. Добавление второй точки

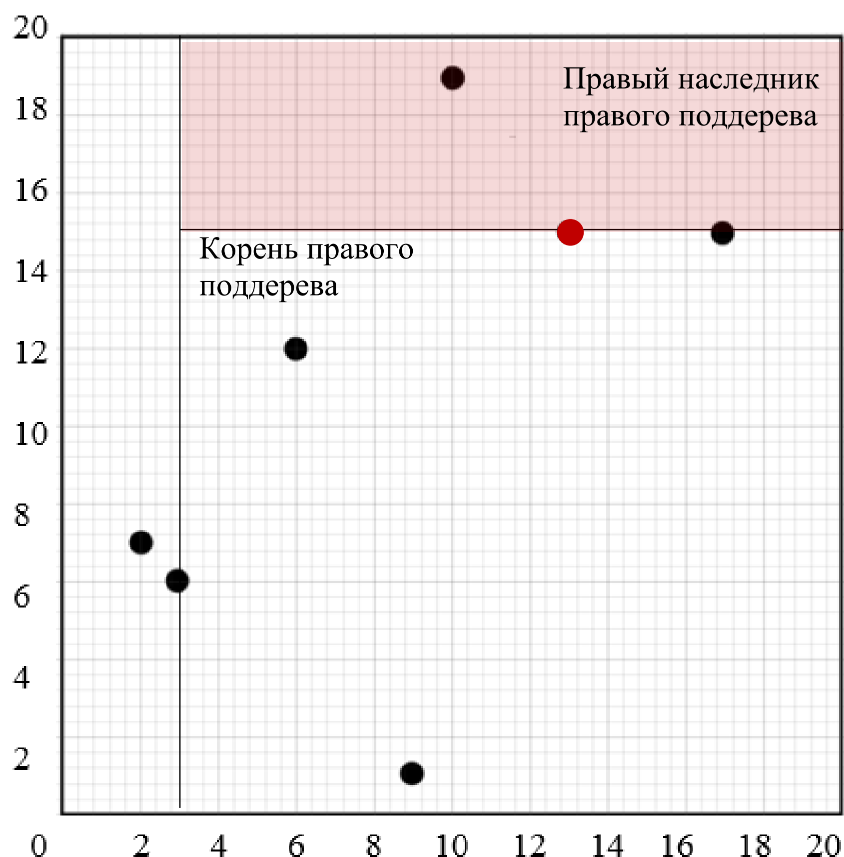


Рисунок 3 – Наследник правого поддерева

Очевидно, что вторая точка из списка (точка (17,15)) находится в красной области и станет узлом правого поддерева, в котором сравнение происходит по координате y (рисунок 3). Точка (13,15) станет наследником правого поддерева, так как принадлежит красной области (рисунок 3).

Точка (6, 12) станет левым наследником (рисунок 5), так как сравнение в корне правого поддерева происходит по второй координате.

Следующая точка (рисунок 4) окажется листовым узлом и будет левым наследником правого наследника правого поддерева. Так как правый наследник находится на втором уровне дерева и сравнение с ним производится по второй координате, то сравнение с его наследниками будет происходить по первой координате.

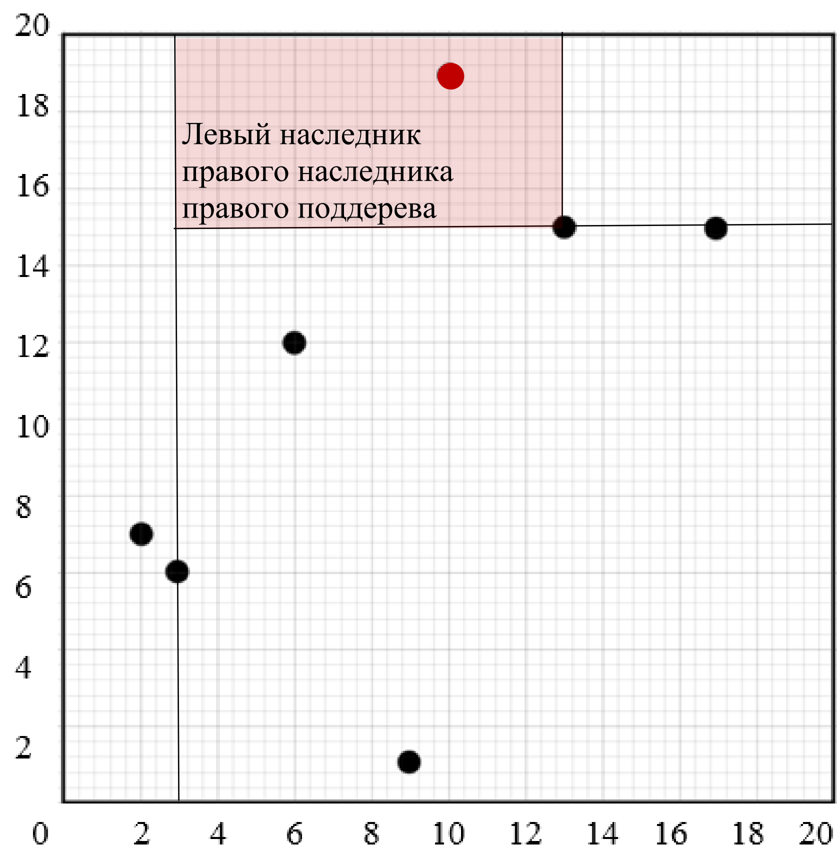


Рисунок 4 – Листовой узел

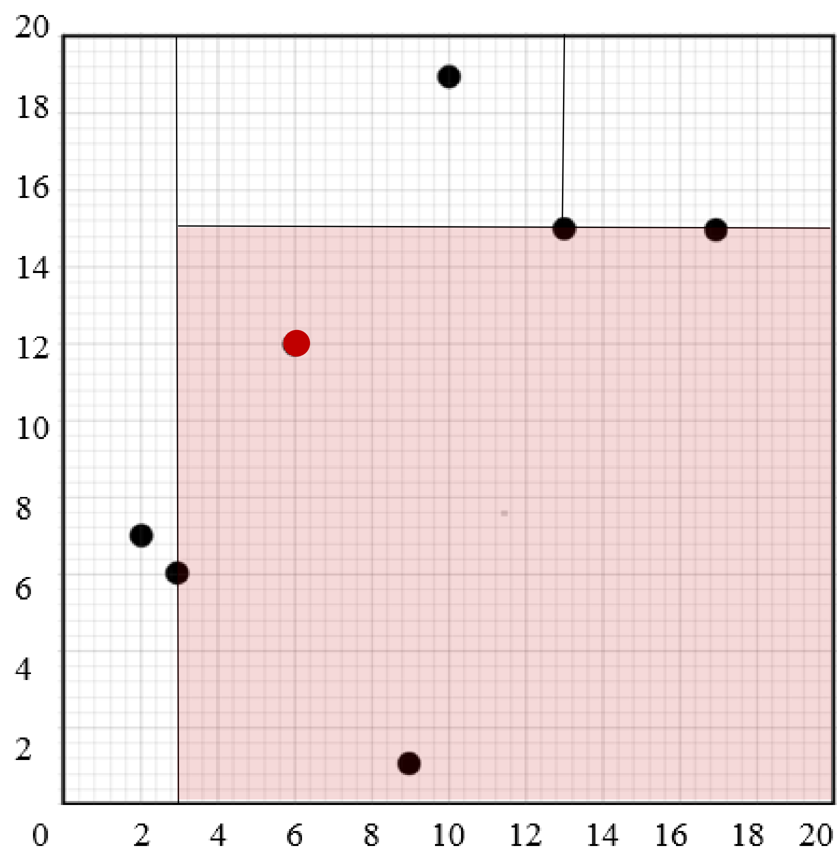


Рисунок 5 – Левый наследник правого поддерева

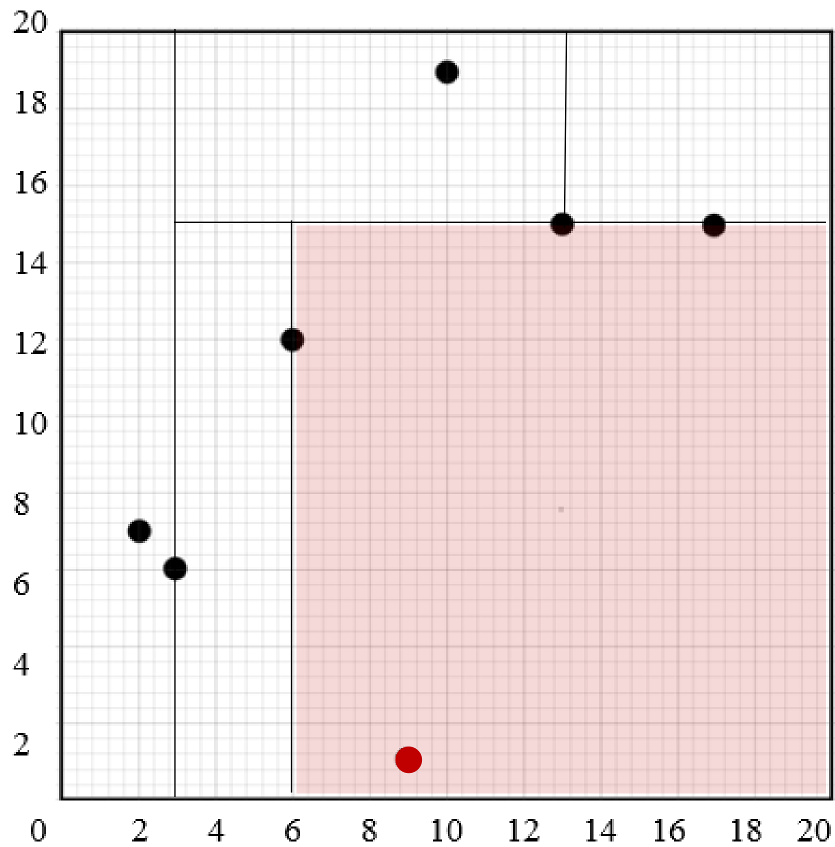


Рисунок 6 – Добавление последней точки

На рисунке 6 представлена последняя точка – точка (9, 1), которая является листовым узлом. Так же листовым узлом является точка (2, 7).

В результате построения, дерево получилось несбалансированным, то есть количество точек в правом и левом поддереве различается. Для устранения данного эффекта можно использовать разделение по медиане или среднему значению точек по каждой из координат.

2 РАСЧЕТ ПАРАМЕТРОВ ПРЕОБРАЗОВАНИЯ

Расчет параметров представляет собой один из основных этапов работы алгоритма. Результатом расчета является матрица поворота и вектор смещения, которые переводят одно множество точек в другое. Применив данное преобразование, можно совместить облака точек.

Чтобы совместить два облака точек, сначала необходимо совместить их центры масс. Затем, используя алгоритм оптимизации, вычислить матрицу поворота и совместить оставшиеся точки. В качестве алгоритма для расчета параметров поворота было использовано сингулярное разложение (singular value decomposition, SVD). Выбор данного метода обусловлен его распространённостью и широким применением в других областях науки и техники.

Пусть $\mathcal{T} = \{\tau_1, \tau_2, \cdots \tau_N\}$ и $\mathcal{S} = \{s_1, s_2, \cdots s_N\}$ $-$ два множества точек в конечномерном вещественном векторном пространстве $\mathbb{R}^3$.

2.1 Вычисление вектора смещения

Чтобы найти смещение, минимизирующее критерий (4), необходимо вычислить его частную производную по аргументу $\boldsymbol{t}$ и приравнять к нулю:

$$\frac{\partial \mathcal{J}}{\partial t} = 2\sum_{i=1}^{N}(Rs_i + t - \tau_i),$$

$$\sum_{i=1}^{N}Rs_i + \sum_{i=1}^{N}t - \sum_{i=1}^{N}\tau_i = 0,$$

$$\sum_{i=1}^{N}Rs_i + tN - \sum_{i=1}^{N}\tau_i = 0.$$

Для того, чтобы избавиться от параметра t, введём константы:

$$\bar{s} = \frac{\sum_{i=1}^{N} s_i}{N},$$
$$\bar{\tau} = \frac{\sum_{i=1}^{N} \tau_i}{N}. \qquad (7)$$

Данная замена имеет геометрический смысл, который заключается в том, что точки с координатами $\bar{\boldsymbol{s}}$ и $\bar{\boldsymbol{\tau}}$ представляют собой положение центров масс для поверхностей $\mathcal{S}$ и $\mathcal{T}$ соответственно. Проведя аналогию с центром масс материальных точек в классической механике, введённые замены можно получить, приняв массы точек равными единице.

Тогда:

$$R\bar{s}N + tN - \bar{\tau}N = 0,$$
$$t = \bar{\tau} - R\bar{s}. \qquad (8)$$

Выражение (8) позволяет найти оптимальное смещение. Используем данное равенство в исходном критерии (4):

$$\sum_{i=1}^{N} \|(Rs_i + t) - \tau_i\|_2^2 = \sum_{i=1}^{N} \|Rs_i + \bar{\tau} - R\bar{s} - \tau_i\|_2^2 = \sum_{i=1}^{N} \left\|R(s_i - \bar{s}) - (\tau_i - \bar{\tau})\right\|_2^2,$$

И введём замену:

$$a_i = s_i - \bar{s}, b_i = \tau_i - \bar{\tau}, \qquad (9)$$

Которая представляет собой координаты векторов, проведённых из центра масс к каждой точке соответствующей поверхности, называемые центроидами. Тогда исходная задача может быть переформулирована как поиск одного

аргумента – параметра $\boldsymbol{R}$, так как центры масс уже совмещены с помощью найденного ранее оптимального смещения. Эти преобразования позволяют перейти к обновлённому критерию минимизации:

$$R = \underset{R \in SO(3)}{\operatorname{argmin}} \sum_{i=1}^{N} \|Ra_i - b_i\|_2^2 . \quad (10)$$

2.2 Вычисление матрицы поворота

По определению (7), норма L_2 порождает метрику ℓ_2, которая представляет собой скалярное произведение векторов, что соответствует произведению вектора строки на вектор столбец:

$$\|Ra_i - b_i\|_2^2 = (Ra_i - b_i)^T (Ra_i - b_i),$$
$$(Ra_i - b_i)^T (Ra_i - b_i) = \left(a_i{}^T R^T - b_i{}^T\right)(Ra_i - b_i) =$$
$$= a_i{}^T R^T R a_i - a_i{}^T R^T b_i - b_i{}^T R a_i + b_i{}^T b_i,$$

Так как матрица R ортогональна, то $R^T R = I$. Можно заметить, что слагаемое $a_i{}^T R^T b_i$ – это скаляр, так как $a_i{}^T$ имеет размер 1×3, R^T имеет размер 3×3, $b_i - 3 \times 1$**.**

Для произвольного скаляра верно $a^T = a$, поэтому:

$$a_i{}^T R^T b_i = (a_i{}^T R^T b_i)^{\boldsymbol{T}} = b_i{}^T R a_i,$$
$$\|Ra_i - b_i\|_2^2 = a_i{}^T R^T R a_i + b_i{}^T b_i - 2 b_i{}^T R a_i = a_i{}^T a_i + b_i{}^T b_i - 2 b_i{}^T R a_i.$$

Вернёмся к задаче минимизации:

$$\underset{R \in SO(3)}{\operatorname{argmin}} \sum_{i=1}^{N} \|Ra_i - b_i\|_2^2 = \underset{R \in SO(3)}{\operatorname{argmin}} \sum_{i=1}^{N} a_i{}^T a_i + b_i{}^T b_i - 2 b_i{}^T R a_i \,.$$

Очевидно, что первые два слагаемых не содержат аргумента минимизации и не имеют влияния на результат оптимизации, поэтому запись можно упростить:

$$R = \underset{R \in SO(3)}{\operatorname{argmin}} \sum_{i=1}^{N} -2{b_i}^T R a_i \,.$$

По той же аналогии можно избавиться от константы и, переформулировав задачу в поиск аргумента максимизации, опустить знак вычитания:

$$R = \underset{R \in SO(3)}{\operatorname{argmin}} \textstyle\sum_{i=1}^{N} -2{b_i}^T R a_i = \underset{R \in SO(3)}{\operatorname{argmax}} \textstyle\sum_{i=1}^{N} {b_i}^T R a_i \,. \tag{11}$$

Критерий (12) можно представить в виде следа матрицы A, полученной умножением:

$$A = B^T R A = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} R \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} = \begin{bmatrix} {b_1}^T R a_1 & * & * & * \\ * & {b_2}^T R a_2 & * & * \\ * & * & \ddots & * \\ * & * & * & {b_n}^T R a_n \end{bmatrix},$$

$$tr(A) = tr(B^T R A) = \textstyle\sum_{i=1}^{N} {b_i}^T R a_i. \tag{12}$$

С учётом свойств следа, можно записать:

$$tr(B^T R A) = tr(R A B^T).$$

Пусть:

$$A B^T = S. \tag{13}$$

Необходимо найти сингулярное разложение:

$$S = U\Sigma V^T. \tag{14}$$

Тогда, введя замену $M = V^T RU$:

$$tr(RS) = tr(RU\Sigma V^T) = tr(\Sigma M).$$

Очевидно, что для того, чтобы максимизировать критерий (12), достаточно максимизировать M. Заметим, что матрицы V, R и U ортогональны, поэтому матрица M так же ортогональна. Это означает, что столбцы матрицы M образуют систему ортонормированных векторов, для которой верно $m_j^T m_j = 1$. Из этого следует, что элементы матрицы $|m_{ij}| \leq 1$.

Таким образом, чтобы максимизировать критерий (12), достаточно, чтобы элементы матрицы M на главной диагонали были равны единице:

$$M = V^T RU = I \Longrightarrow R = VU^T. \tag{15}$$

Суммируя всё выше сказанное, можно записать краткий алгоритм работы, который будет использоваться в программном коде для решения задачи совмещения.

2.3 Конечный алгоритм

а) Вычисление центроидов для каждой поверхности:

$$\bar{s} = \frac{\sum_{i=1}^{N} s_i}{N},$$

$$\bar{\tau} = \frac{\sum_{i=1}^{N} \tau_i}{N}.$$

б) Вычисление центрированных векторов:

$$a_i = s_i - \bar{s},$$

$$b_i = \tau_i - \bar{\tau}.$$

в) Вычисление матрицы:

$$S = AB^T.$$

г) Вычисление сингулярного разложения матрицы S.

д) Вычисление матрицы поворота:

$$R = VU^T.$$

е) Вычисление вектора смещения:

$$t = \bar{\tau} - R\bar{s}.$$

2.4 Полное сингулярное разложение матрицы

Сингулярное разложение представляет собой мощный вычислительный инструмент для анализа матриц и задач с ними связанными, который имеет приложения в различных областях.

Сингулярным разложением действительной $m \times n$ матрицы A называется факторизация вида:

$$A = U\Sigma V^T, \tag{16}$$

где $U -$ ортогональная матрица $m \times m$;
$V -$ ортогональная матрица $n \times n$;
$\Sigma -$ диагональная матрица $m \times n$, у которой $\sigma_{ij} = 0$ при $i \neq j$ и
$\sigma_{ii} = \sigma_i \geq 0$.

Величины σ_{ii} называются сингулярными числами матрицы A, а столбцы матриц U и V – левыми и правыми сингулярными векторами.

За сингулярным разложением скрывается такая идея: при надлежащем выборе матриц U и V можно обратить большинство элементов матрицы Σ в нули; более того, можно даже сделать её диагональной матрицей с неотрицательными диагональными элементами.

Сингулярное разложение матрицы обладает свойством, которое связывает задачу факторизации и нахождения собственных векторов. Собственным вектором $\boldsymbol{x}$ матрицы A называется такой вектор, при котором выполняется условие $A\boldsymbol{x} = \lambda\boldsymbol{x}$, где $\lambda -$ собственное число.

Из (17) и ортогональности матриц U и V следует, что:

$$AA^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T,$$
$$A^T A = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T,$$

До множив на U и V соответственно:

$$AA^TU = U\Sigma^2, \quad (17)$$

$$A^TAV = V\Sigma^2. \quad (18)$$

Откуда следует, что столбцы матриц U и V являются собственными векторами матриц AA^T и A^TA соответственно, а собственными значениями являются квадраты сингулярных чисел Σ.

Для вычисления матриц U и V необходимо найти соответствующие матрицы собственных векторов U^* и V^*, а затем провести их ортогонализацию с помощью алгоритма Грама-Шмидта (см. Приложение А).

2.5 Процесс ортогонализации Грама-Шмидта

Пусть $v = (v_1, v_2, \cdots, v_n) -$ некоторый базис в n-мерном евклидовом пространстве, который можно модифицировать преобразованием его в новый ортонормированный базис $e = (e_1, e_2, \cdots, e_n)$.

Алгоритм сводится к последовательному вычислению g_i и e_i:

$$g_1 = v_1,\ e_1 = \frac{g_1}{\|g_1\|_2},$$

$$g_2 = v_2 - (v_2, e_1)e_1,\ e_2 = \frac{g_2}{\|g_2\|_2},$$

$$g_3 = v_3 - (v_3, e_1)e_1 - (v_3, e_2)e_2,\ e_3 = \frac{g_3}{\|g_3\|_2},$$

$$g_n = v_n - (v_n, e_1)e_1 - \cdots - (v_n, e_{n-1})e_{n-1},\ e_n = \frac{g_n}{\|g_n\|_2}.$$

В результате, для матриц U^* и V^* можно получить матрицы U и V.

3 ПРИМЕНЕНИЕ АЛГОРИТМА

Реализация работы алгоритма требует изучения документации различных библиотек и построения структуры программы. Первый и самый затруднительный этап алгоритма – это поиск ближайших точек. Наиболее перспективные и быстрые библиотеки, реализующие построение и поиск в KD-деревьях – это библиотеки flann и nanoflann. В качестве рабочей библиотеки была использована flann. Также необходимо произвести некоторые матричные преобразования и использовать сингулярное разложение матрицы.

В настоящее время создано большое количество пакетов для матричных преобразований. В реализованном алгоритме используется библиотека Eigen. Она довольно просто позволяет производить матричные операции, что особенно удобно при работе с большими объёмами данных, такими как облака точек.

3.1 Структура программы

Программа состоит из нескольких файлов:

main.cpp

IO.h

Optimization.cpp

Optimization.h

Registration.cpp

Registration.h

Файл IO.h содержит объявления классов для чтения и визуализации данных.

Файл Optimization.h содержит объявление класса для преобразования и оптимизации облака точек, в Optimization.cpp представлена соответствующая реализация методов.

Файл Registration.h содержит абстрактный класс для совмещения облаков точек и класс-наследник ICP. Registration.cpp содержит соответствующую реализацию.

Диаграмма классов представлена на рисунке 7.

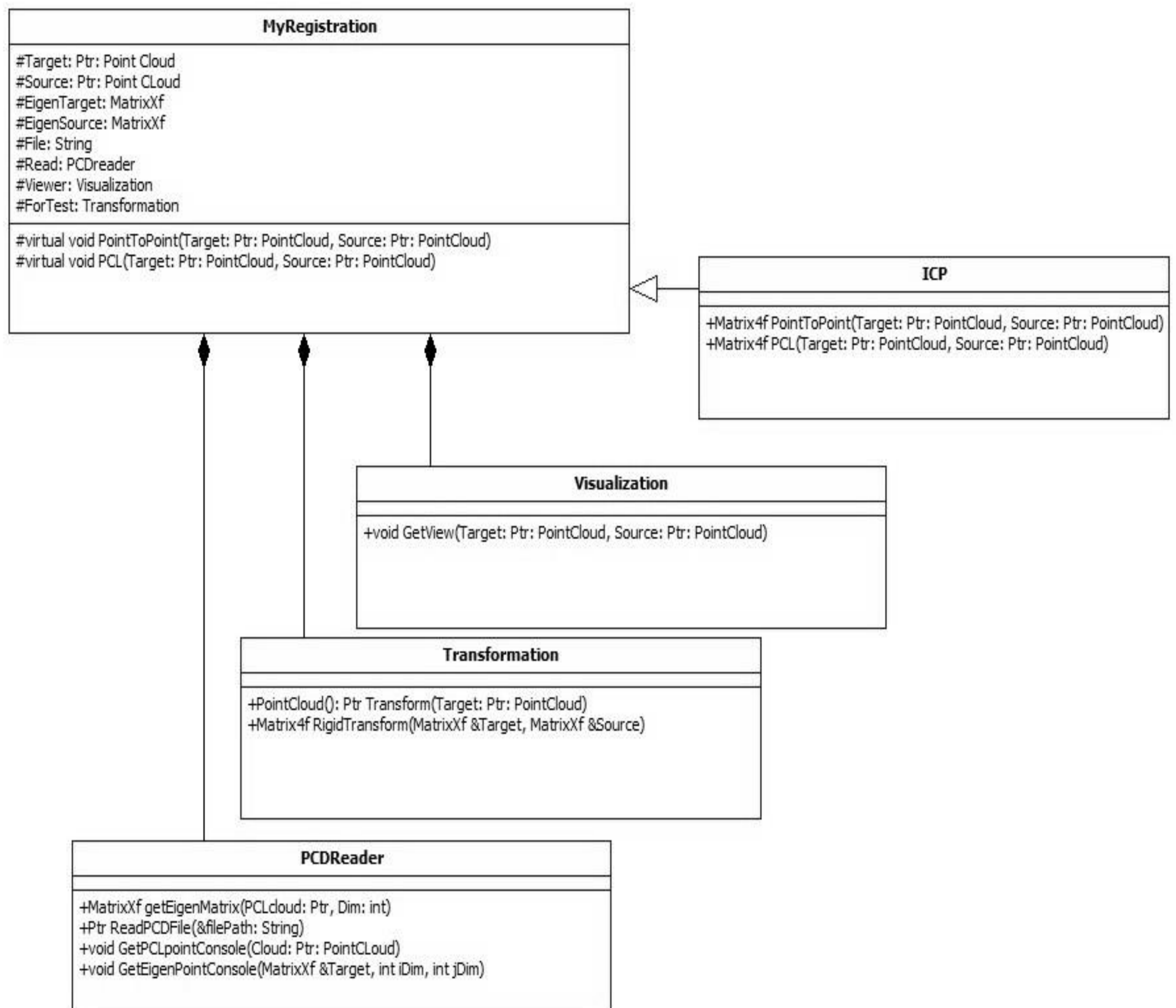


Рисунок 7 – Диаграмма классов

3.2 Алгоритм работы

Основные этапы программы, содержащиеся в main.cpp можно разбить на:

1. Target = Read("cat.pcd");
2. Source = Transform(Target);
3. Transformation = PointToPoint (Target, Source);
4. GetView (Target, Source);

На первом этапе происходит чтение файла. В качестве входных данных использованы облака точек в формате point cloud data (PCD), представляющие собой матрицу $3 \times N$. Далее задаётся преобразование для облака точек и выполняется вычисление параметров для его устранения. На последнем этапе отображаются совмещённые облака точек.

Первый и второй пункты реализованы с помощью PCL библиотеки, их реализация представлена в файлах IO.h Optimization.cpp соответственно (см. Приложение Б). В методе PointToPoint вычисляется и применяется преобразование, минимизирующее критерий оптимизации на каждой итерации. Метод возвращает оптимальное преобразование в формате Matrix4f из библиотеки Eigen в следующем виде:

$$Result = \begin{bmatrix} & & & t_x \\ & R & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (19)$$

Общий алгоритм работы PointToPoint следующий:

1. EigenTarget = getEigenMatrix(Target);
2. setInputCloud(Target);
3. nearestKSearch(Source[index]);
4. Result = ForTest.RigidTransform(NNMatrix, EigenSource);
5. transformPointCloud(*Source, *Source, Result);

Для работы с SVD разложением и библиотекой Eigen необходим другой формат данных, отличный от PCD. Поэтому на первом этапе происходит преобразование облака точек в формат матриц Eigen для последующей работы. Далее необходимо построить 3D-дерево для целевого облака точек и найти в нём пару ближайших точек для каждой точки из совмещаемого облака (этап 3) и заполнить матрицу ближайших точек NNMatrix, в которую повекторно из EigenTarget копируются строки. Таким образом могут быть получены пары индексов ближайших точек.

Далее с помощью сингулярного разложения (JacobiSVD) вычисляется результат в формате (19), который затем применяется к совмещаемому облаку точек Source. После этого весь процесс повторяется до тех пор, пока не завершится цикл работы ICP алгоритма либо не будет достигнуто приемлемое значение критерия оптимизации.

3.3 Результат совмещения

В качестве входных данных была использована тестовая модель в формате PCD (Point cloud data), содержащая 3400 точек.

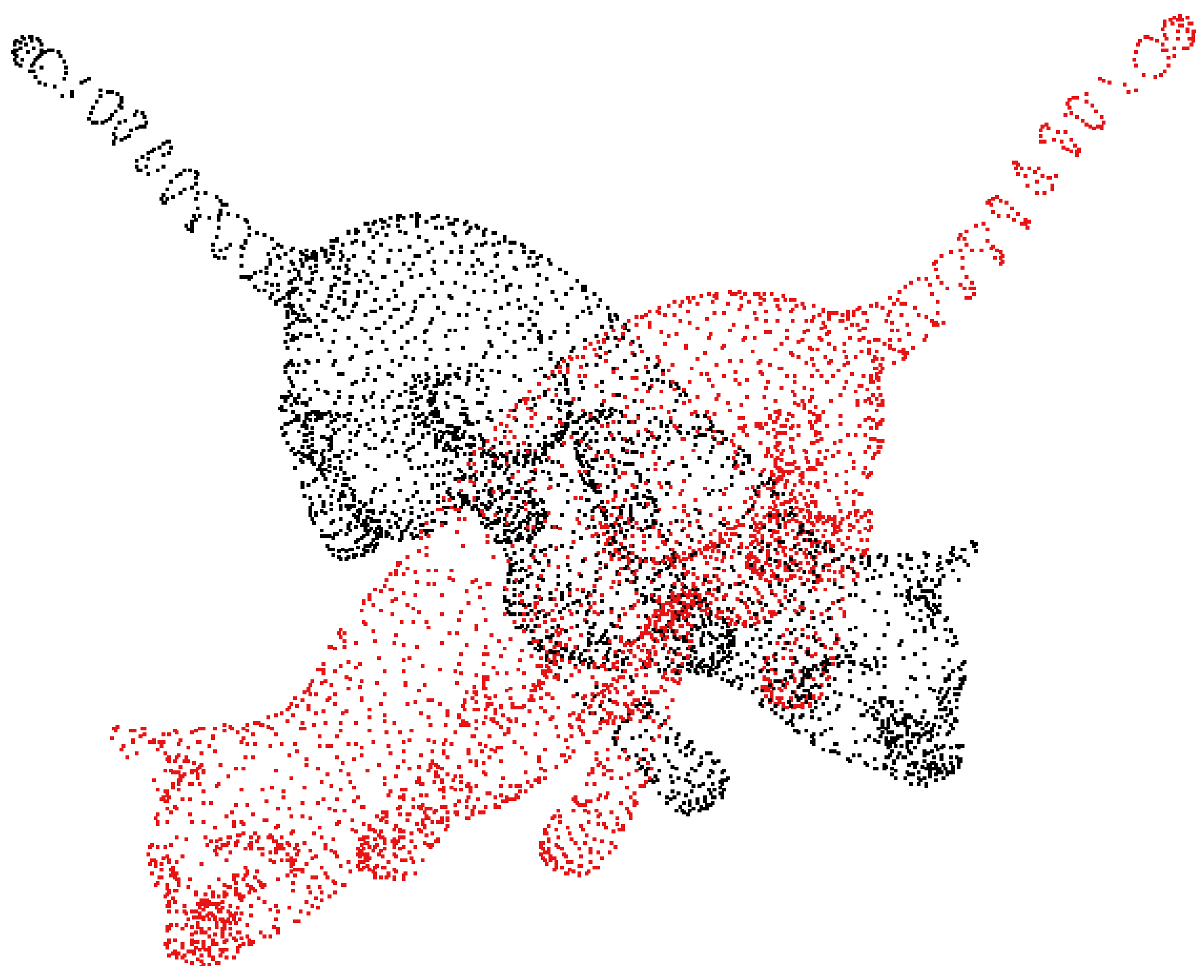


Рисунок 8 – Тестовые облака точек

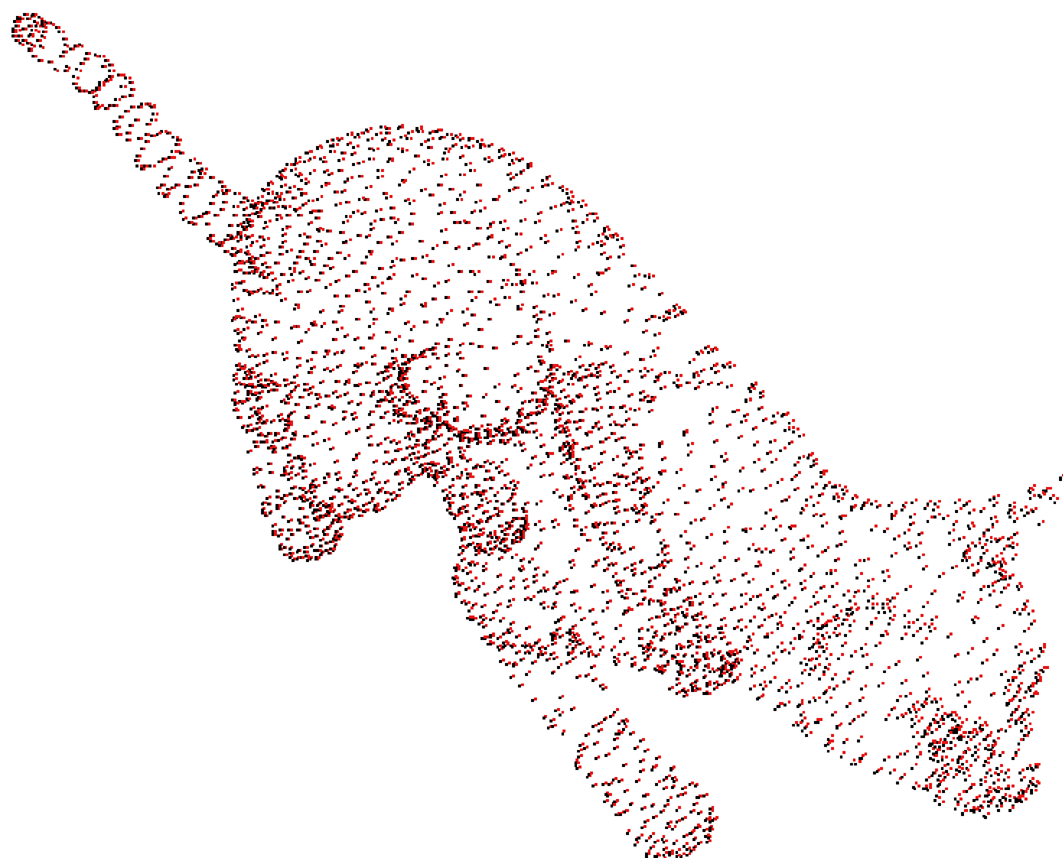


Рисунок 9 – Результат первого теста

Для целевого облака точек (точки чёрного цвета) в первом тесте (рисунок 8) был задан произвольный поворот по оси Z (60°); смещение отсутствует. Результат сходимости представлен на рисунке 10. Критерий останова не использовался.

Как видно из графика, было достаточно 15-20 итераций, чтобы совместить тестовые облака точек. Всё зависит от требуемой точности совмещения, то есть требуемой близости к нулю критерия оптимизации.

В результате применения найденного преобразования к облаку с красными точками, можно получить результат, представленный на рисунке 9.

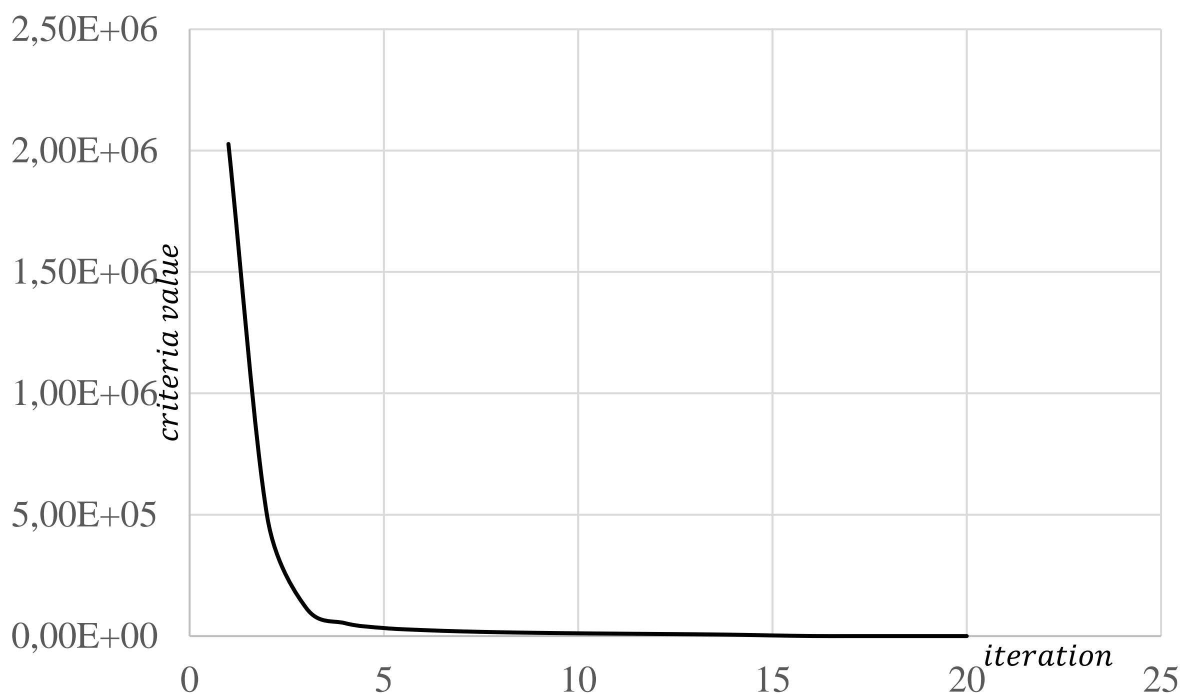


Рисунок 10 – Сходимость первого теста

Для целевого облака точек во втором тесте (рисунок 11) был задан произвольный поворот по оси Z (60°) и смещение. Результат сходимости представлен на рисунке 13.

В результате применения найденного преобразования к облаку с красными точками, можно получить результат, представленный на рисунке 13. Визуально облака точек не совмещены, потому что оказалось недостаточно 25-ти заданных итераций. Очевидно, что второй тест сходится медленнее.

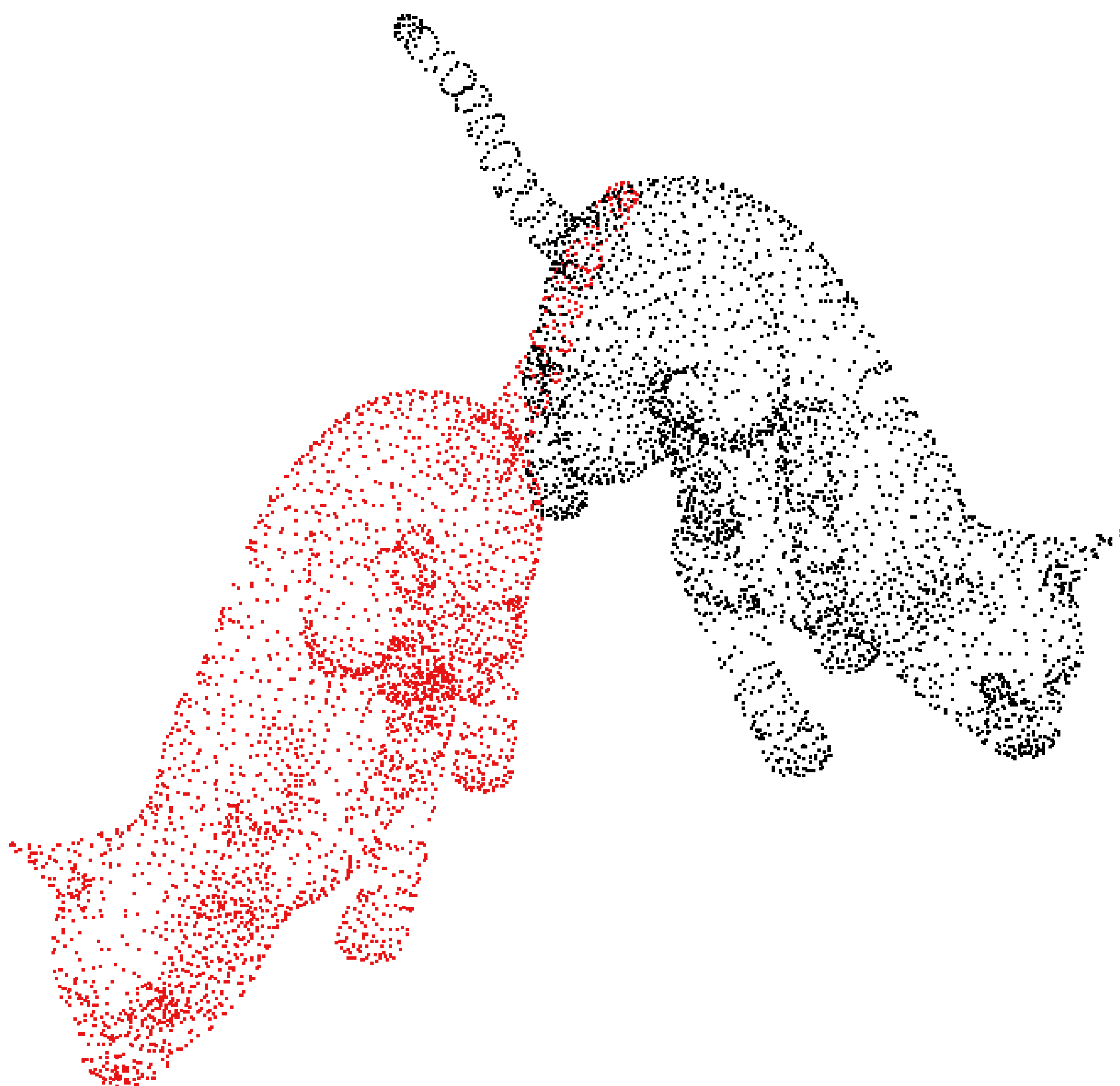


Рисунок 11 – Второй тест

Основной проблемой совмещения является формирование соответствующих пар точек. Как следствие, увеличивается не только время сходимости, но и качество совмещения. Очевидно, что при более сложных положениях облаков точек друг относительно друга, времени для совмещения понадобится больше, иногда и вовсе критерий может не сойтись к минимуму (при больших поворотах или при симметричных объектах совмещения).

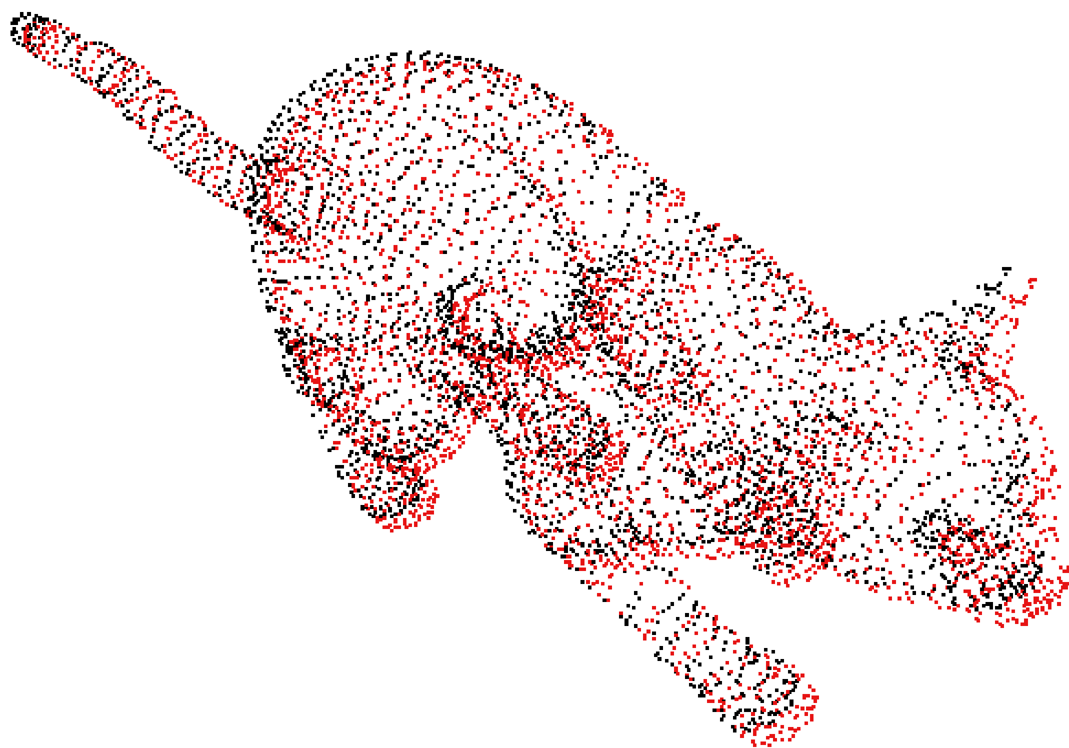


Рисунок 12– Результат второго теста

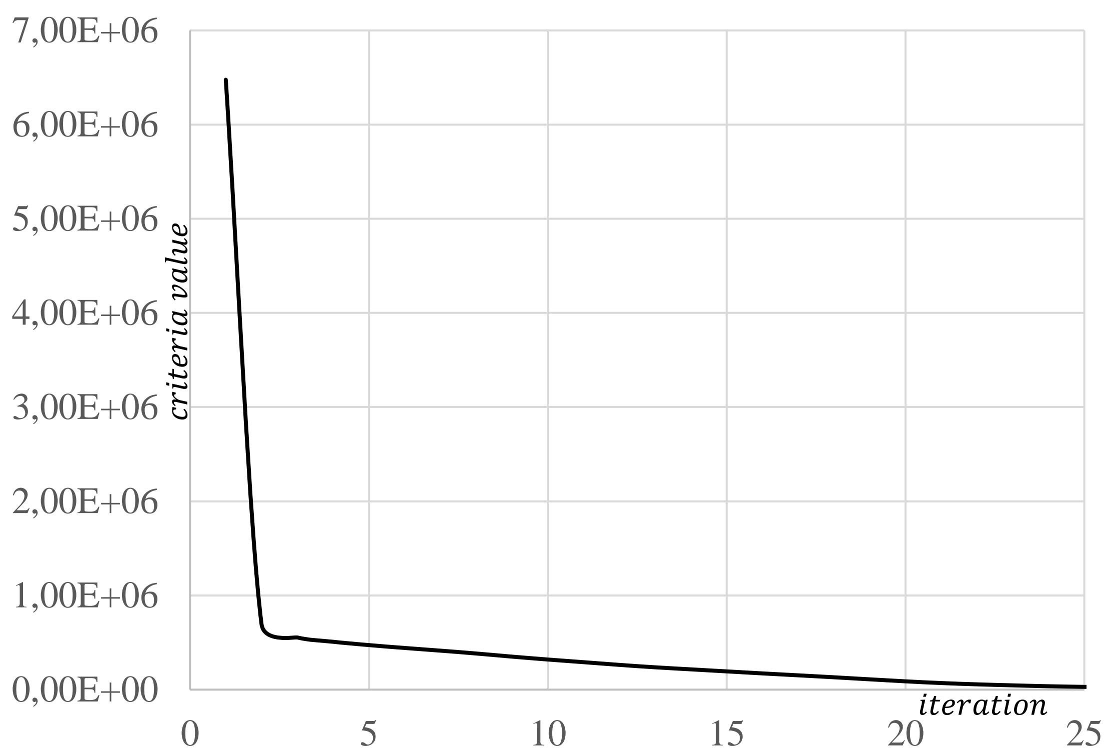


Рисунок 13 – Сходимость второго теста

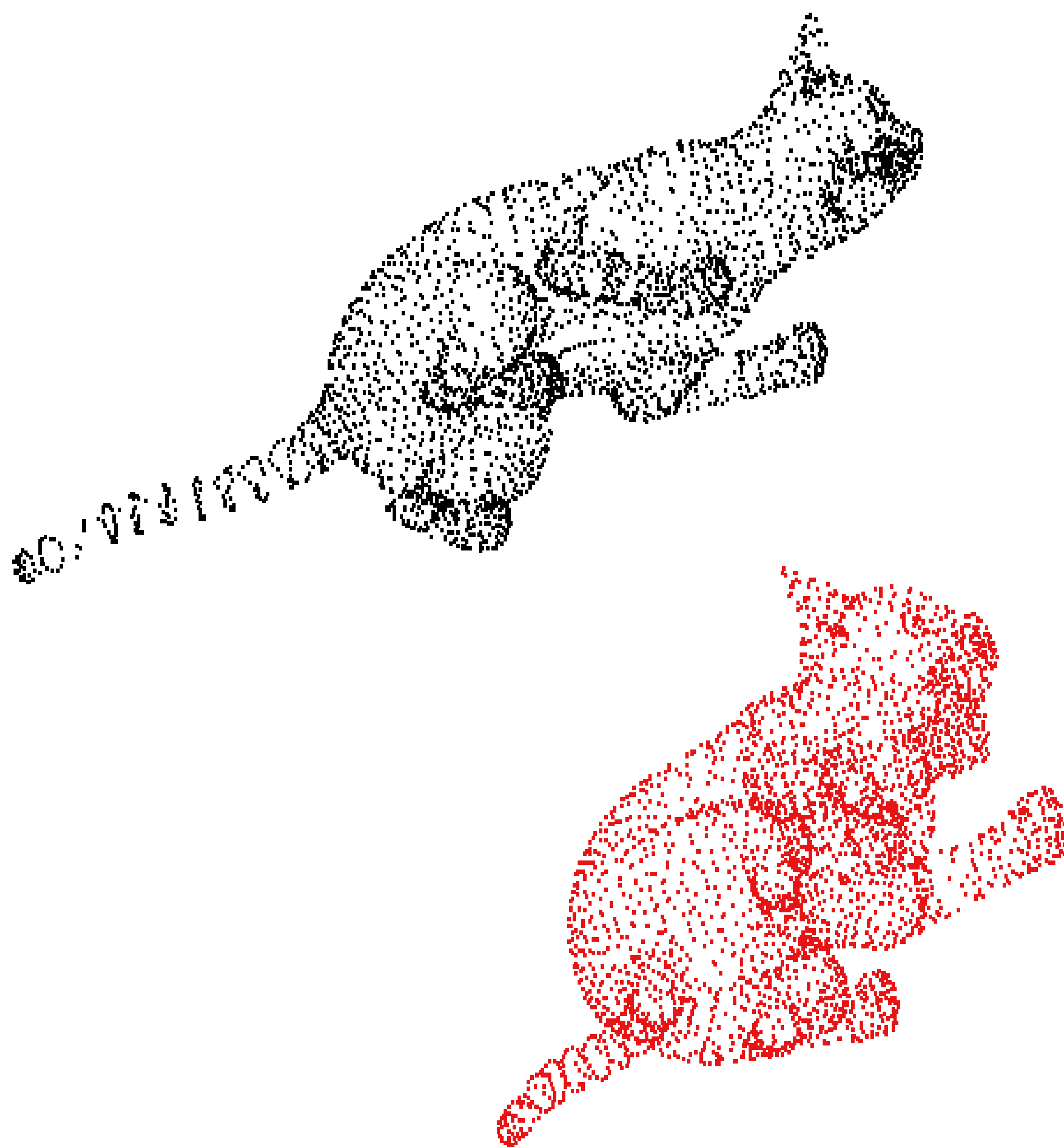


Рисунок 14 – Третий тест

Для целевого облака точек в третьем тесте (рисунок 14) был задан произвольный поворот по оси X и Y, а также некоторое смещение. Результат сходимости представлен на рисунке 16.

Как видно из графика, было достаточно 15-20 итераций, чтобы совместить тестовые облака точек. В результате применения найденного преобразования к

облаку с красными точками, можно получить результат, представленный на рисунке 15.

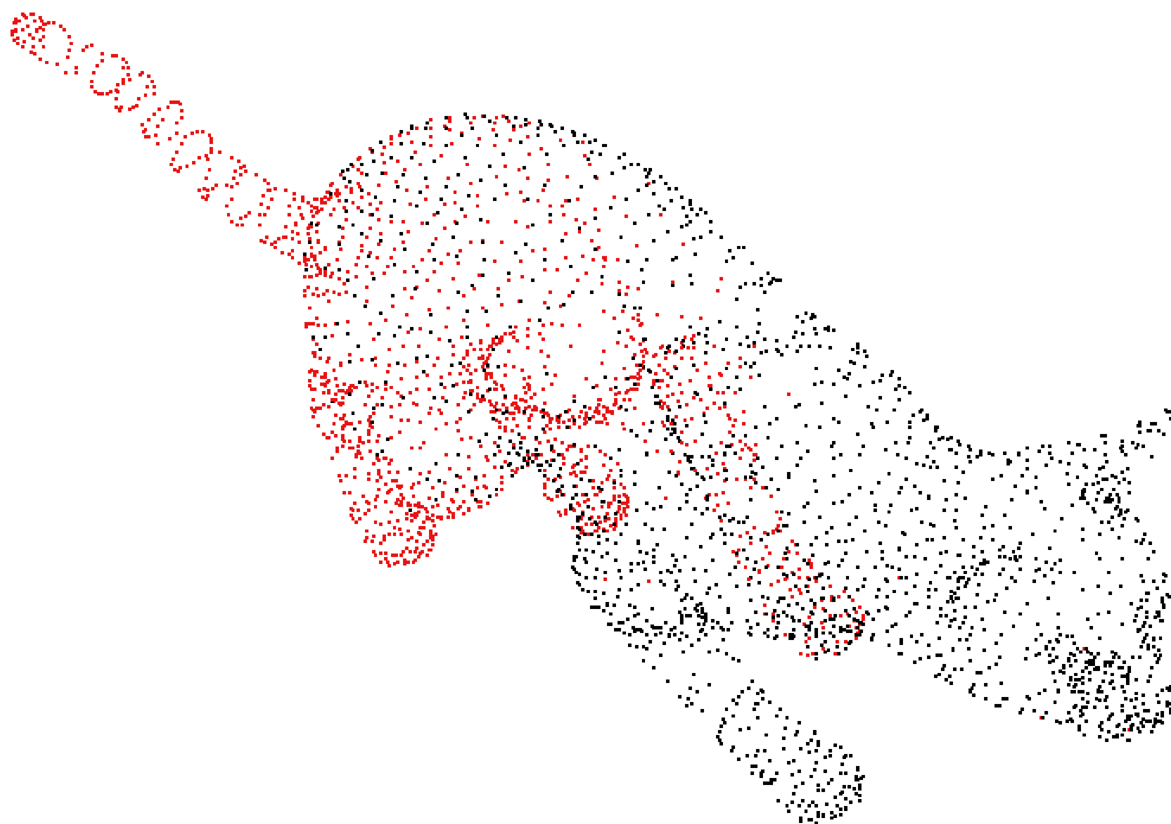


Рисунок 15 – Результат третьего теста

Также для совмещения облаков точек из второго теста была использована библиотека PCL. Она содержит различные методы для реконструкции поверхностей и трёхмерного совмещения.

Для сравнения было задано то же количество итераций, что и во втором тесте. Совмещение представлено на рисунке 17. Очевидно, что результаты схожи.

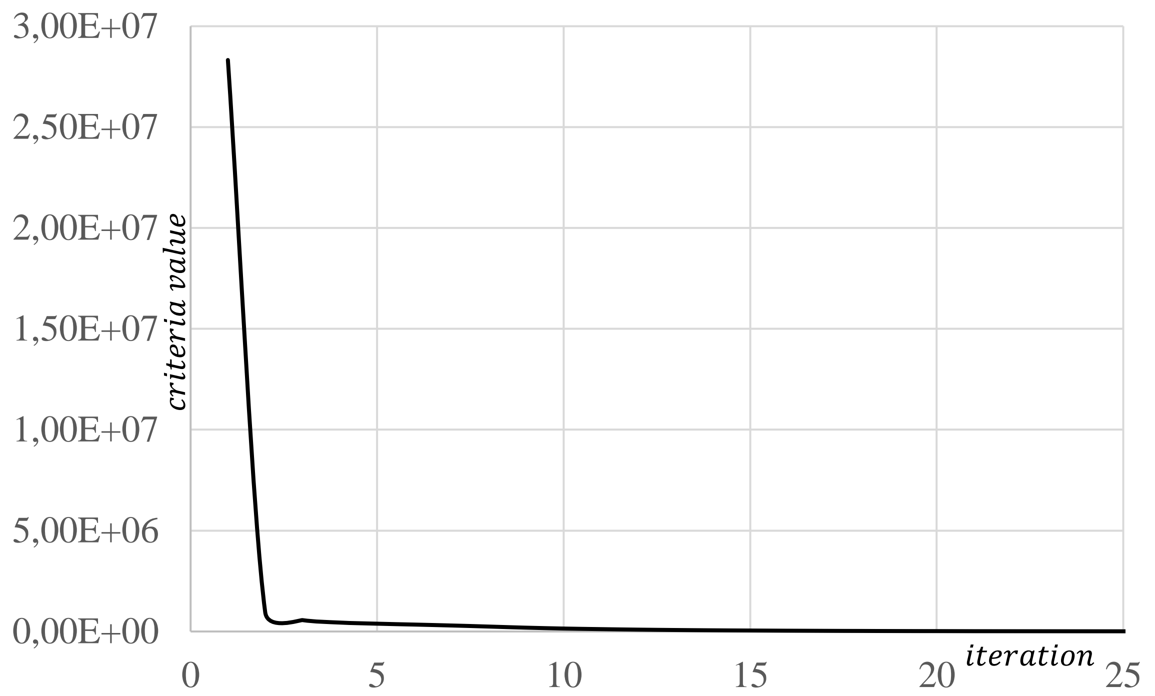


Рисунок 16 – Сходимость третьего теста

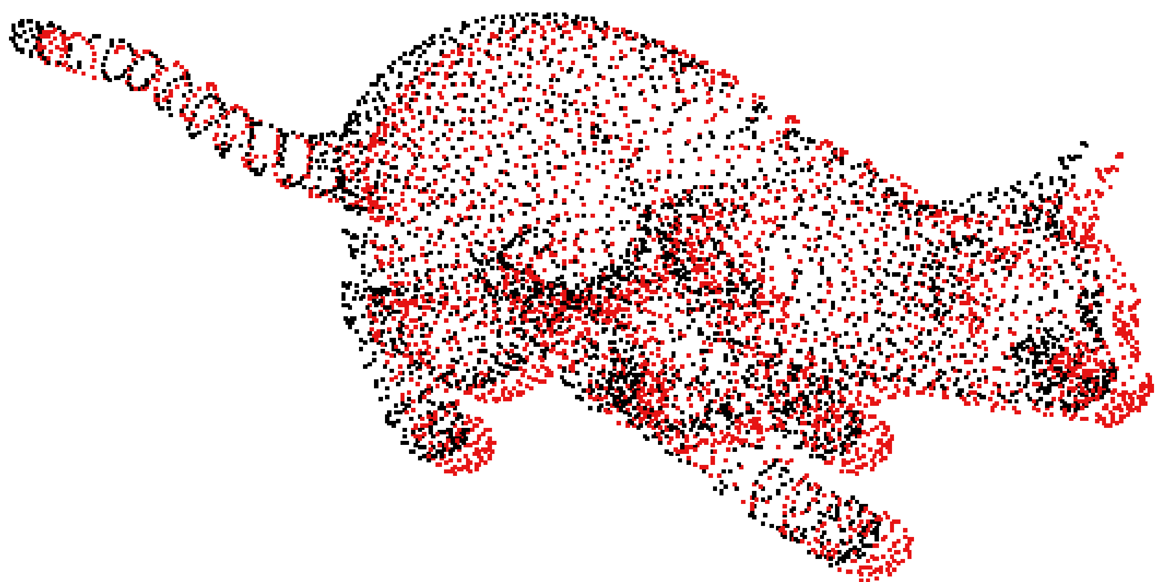


Рисунок 17 – Результат второго теста PCL

4 ЗАКЛЮЧЕНИЕ

Алгоритмы сингулярного разложения имеют широкое практическое применение в различных областях, что делает их незаменимым инструментом, который следует изучать. В частности, SVD используют такие современные алгоритмы ICP как sparse, что позволяет сделать шаг на пути к изучению самых актуальных и передовых методов в области сканирования и технического зрения.

В процессе работы был изучен классический ICP алгоритм. В силу распространённости, а также удобства использования при работе с матрицами, выбран метод оптимизации с использованием сингулярного разложения матриц. Работа была направлена на практическое применение итеративного алгоритма ближайших точек. Результатом работы является рабочий программный код.

В процессе тестирования программы, были получены совмещённые облака точек и показана скорость сходимости критерия оптимизации. Недостатками данного алгоритма является плохое качество совмещения при больших углах поворота облаков точек относительно друг друга. Основные проблемы изученного алгоритма, которые необходимо решить для его улучшения, лежат в области формирования соответствующих пар точек и времени их поиска.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Стрижов, В.В. Информационное моделирование [Электронный ресурс]: [Конспект лекций] / Стрижов, В.В. – Электрон. текстовые дан. – Москва: МФТИ, 2003. – Режим доступа: URL: http://strijov.com/teaching/
2. A practical implementation of KD trees [Электронный ресурс]: Engineering blog – Режим доступа: URL: https://programmizm.sourceforge.io/blog/2011/a-practical-implementation-of-kd-trees
3. Franco P.Preparata. Computational geometry an introduction / Franco P.Preparata, Michael lan Shamos // Springer-Verlag Berlin, Heidelberg, 1985. – C. 52-57.
4. Olga Sorkine-Hornung. Least-Squares Rigid Motion Using SVD / Olga Sorkine-Hornung, Michael Rabinovich // Department of Computer Science, ETH Zurich January 16, 2016.
5. Bentley, Jon Louis . Multidimensional Binary Search Trees Used for Associative Searching / Jon Louis Bentley // Communications of the ACM, vol.18, 1975. – C. 509-517.
6. Paul J.Besl. A Method for Registration of 3-D Shapes / Paul J.Besl, Neil D.McKay // Transactions on pattern analysis and machine intelligence, vol.14, NO.2, FEBRUARY 1992. – C. 239-255.
7. K Dimensional Tree: Set 1 (Search and Insert) [Электронный ресурс]: Geeksforgeeks: A computer science portal for geeks – Режим доступа: URL: https://www.geeksforgeeks.org/k-dimensional-tree/
8. Kirk Baker. Singular Value Decomposition Tutorial, March 29, 2005.
9. Sofien Bouaziz. Sparse Iterative Closest Point / Sofien Bouaziz, Andrea Tagliasacchi, Mark Pauly // Eurographics Symposium on Geometry Processing, vol.32, number 5, 2013.

СОДЕРЖАНИЕ

АННОТАЦИЯ **5**
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ **6**
ВВЕДЕНИЕ **7**
1 ОБЩЕЕ ОПИСАНИЕ РАБОТЫ АЛГОРИТМА **9**
1.1 Основные этапы работы алгоритма **9**
1.2 Поиск ближайших точек **11**
1.3 Геометрическая интерпретация KD-дерева **13**
2 РАСЧЕТ ПАРАМЕТРОВ ПРЕОБРАЗОВАНИЯ **18**
2.1 Вычисление вектора смещения **18**
2.2 Вычисление матрицы поворота **20**
2.3 Конечный алгоритм **23**
2.4 Полное сингулярное разложение матрицы **24**
2.5 Процесс ортогонализации Грама-Шмидта **25**
3 ПРИМЕНЕНИЕ АЛГОРИТМА **26**
3.1 Структура программы **26**
3.2 Алгоритм работы **28**
3.3 Результат совмещения **30**
4 ЗАКЛЮЧЕНИЕ **38**
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **39**
СОДЕРЖАНИЕ **40**
ПРИЛОЖЕНИЕ А **41**
ПРИЛОЖЕНИЕ Б **48**
Б.1. Файл main.cpp **48**
Б.2. Файл IO.h **49**
Б.3. Файл Optimization.cpp **51**
Б.4. Файл Optimization.h **54**
Б.5. Файл Registration.cpp **54**
Б.6. Файл Registration.h **57**

ПРИЛОЖЕНИЕ А

Сингулярное разложение матриц

Пусть дана матрица А:

$$A = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}.$$

Найдём AA^T:

$$AA^T = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}.$$

Далее необходимо найти собственные вектора AA^T. По определению:

$$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \quad (А.1)$$

Перепишем в виде системы двух равенств:

$$\begin{cases} 11x_1 + x_2 = \lambda x_1 \\ x_1 + 11x_2 = \lambda x_2 \end{cases},$$

$$\begin{cases} (11 - \lambda)x_1 + x_2 = 0 \\ x_1 + (11 - \lambda)x_2 = 0 \end{cases}, \quad (А.2)$$

И запишем характеристический многочлен:

$$\begin{vmatrix} (11 - \lambda) & 1 \\ 1 & (11 - \lambda) \end{vmatrix} = 0,$$

$$(11-\lambda)(11-\lambda)-1=0,$$
$$\lambda^2-11\lambda-11\lambda+121-1=0,$$
$$\lambda^2-22\lambda+120=0.$$

Решив его, можно найти собственные числа:

$$\lambda=10, \lambda=12.$$

Подставив найденные корни характеристического многочлена в систему (А.2), можно найти матрицу собственных векторов U^*:

Для $\lambda=12$:

$$x_1+(11-12)x_2=0,$$
$$x_1=x_2.$$

Для простоты примем, что $x_1=1$, тогда $\boldsymbol{v}_1=[1 \quad 1]$.

Для $\lambda=10$:

$$(11-10)x_1+x_2=0,$$
$$x_1=-x_2.$$

Для простоты примем, что $x_1=1$, тогда $\boldsymbol{v}_2=[1 \quad -1]$. В итоге получим:

$$U^*=[\,\boldsymbol{v}_1 \quad \boldsymbol{v}_2]=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \quad (А.3)$$

Найдём $A^T A$:

$$A^T A = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix}.$$

Собственные вектора найдём из:

$$\begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{A.4}$$

Перепишем в виде системы равенств:

$$\begin{cases} 10x_1 + 2x_3 = \lambda x_1 \\ 10x_2 + 4x_3 = \lambda x_2 \\ 2x_1 + 4x_2 + 2x_3 = \lambda x_3 \end{cases},$$

$$\begin{cases} (10 - \lambda)x_1 + 2x_3 = 0 \\ (10 - \lambda)x_2 + 4x_3 = 0 \\ 2x_1 + 4x_2 + (2 - \lambda)x_3 = 0 \end{cases}. \tag{A.5}$$

И запишем характеристический многочлен:

$$\begin{vmatrix} (10 - \lambda) & 0 & 2 \\ 0 & (10 - \lambda) & 4 \\ 2 & 4 & (2 - \lambda) \end{vmatrix} = 0.$$

Разложим по первой строке:

$$(10-\lambda)\begin{vmatrix}(10-\lambda) & 4 \\ 4 & (2-\lambda)\end{vmatrix} + 2\begin{vmatrix}0 & (10-\lambda) \\ 2 & 4\end{vmatrix} = = (10-\lambda)[(10-\lambda)(2-\lambda)-16] + 2[0-2(10-\lambda)] = 0.$$

Решив, можно найти собственные числа:

$$\lambda = 0, \lambda = 10, \lambda = 12.$$

Подставив найденные корни характеристического многочлена в систему (А.5), можно найти матрицу собственных векторов V^*.

Для $\lambda = 12$:

$$(10-12)x_1 + 2x_3 = 0,$$
$$x_1 = x_3,$$
$$x_1 = 1, x_3 = 1.$$

$$(10-12)x_2 + 4x_3 = 0,$$
$$x_2 = 2x_3 = 2,$$
$$\boldsymbol{v}_1 = [1 \quad 2 \quad 1].$$

Для $\lambda = 10$:

$$(10-10)x_1 + 2x_3 = 0,$$
$$x_3 = 0,$$

$$2x_1 + 4x_2 = 0,$$
$$x_1 = -2x_2,$$
$$x_1 = 2, x_2 = -1,$$

$$\boldsymbol{v}_2 = [2 \quad -1 \quad 0].$$

Для $\lambda = 0$:

$$(10-0)x_1 + 2x_3 = 0,$$
$$x_3 = -5x_1,$$
$$x_1 = 1,$$
$$x_3 = -5.$$

$$2x_1 + 4x_2 + (2-0)x_3 = 0,$$
$$x_2 = 2x_1 = 2,$$

$$\boldsymbol{v}_3 = [1 \quad 2 \quad -5].$$

Тогда матрица V^*:

$$V^* = [\,\boldsymbol{v}_1 \quad \boldsymbol{v}_2 \quad \boldsymbol{v}_3] = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 2 \\ 1 & 0 & -5 \end{bmatrix} \qquad \text{(A.6)}$$

Преобразуем матрицу (А.3) в ортогональную с помощью ортогонализации Грама-Шмидта:

$$\boldsymbol{g}_1 = [1 \quad 1],$$

$$\boldsymbol{e}_1 = \frac{[1 \quad 1]}{\sqrt{1^2 + 1^2}} = \left[\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}\right],$$

$$\boldsymbol{g}_2 = [1 \quad -1] - [1 \quad -1] \cdot \left[\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}\right]\left[\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}\right] = [1 \quad -1] - [0 \quad 0] = [1 \quad -1],$$

$$\boldsymbol{e}_2 = \frac{[1 \quad -1]}{\sqrt{1^2 + 1^2}} = \left[\frac{1}{\sqrt{2}} \quad -\frac{1}{\sqrt{2}}\right].$$

$$U = [\,\boldsymbol{e}_1 \quad \boldsymbol{e}_2] = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}.$$

Аналогичный процесс проделаем для матрицы (А.6):

$$\boldsymbol{g}_1 = [1 \quad 2 \quad 1],$$

$$\boldsymbol{e}_1 = \frac{[1 \quad 2 \quad 1]}{\sqrt{1^2 + 2^2 + 1^2}} = \left[\frac{1}{\sqrt{6}} \quad \frac{2}{\sqrt{6}} \quad \frac{1}{\sqrt{6}}\right],$$

$$\boldsymbol{g}_2 = [2 \quad -1 \quad 0] - [2 \quad -1 \quad 0] \cdot \left[\frac{1}{\sqrt{6}} \quad \frac{2}{\sqrt{6}} \quad \frac{1}{\sqrt{6}}\right]\left[\frac{1}{\sqrt{6}} \quad \frac{2}{\sqrt{6}} \quad \frac{1}{\sqrt{6}}\right] = [2 \quad -1 \quad 0],$$

$$\boldsymbol{e}_2 = \frac{[2 \quad -1 \quad 0]}{\sqrt{2^2 + 1^2 + 0^2}} = \left[\frac{2}{\sqrt{5}} \quad \frac{-1}{\sqrt{5}} \quad 0\right],$$

$$\boldsymbol{g}_3 = [1 \quad 2 \quad -5] - [1 \quad 2 \quad -5] \cdot \left[\frac{1}{\sqrt{6}} \quad \frac{2}{\sqrt{6}} \quad \frac{1}{\sqrt{6}}\right]\left[\frac{1}{\sqrt{6}} \quad \frac{2}{\sqrt{6}} \quad \frac{1}{\sqrt{6}}\right] - [1 \quad 2 \quad -5]$$
$$\cdot \left[\frac{2}{\sqrt{5}} \quad \frac{-1}{\sqrt{5}} \quad 0\right]\left[\frac{2}{\sqrt{5}} \quad \frac{-1}{\sqrt{5}} \quad 0\right] = [1 \quad 2 \quad -5],$$

$$\boldsymbol{e}_3 = \frac{[1 \quad 2 \quad -5]}{\sqrt{1^2 + 2^2 + 5^2}} = \left[\frac{1}{\sqrt{30}} \quad \frac{2}{\sqrt{30}} \quad \frac{-5}{\sqrt{30}}\right],$$

$$V = [\, \boldsymbol{e}_1 \quad \boldsymbol{e}_2 \quad \boldsymbol{e}_3] = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix}.$$

ПРИЛОЖЕНИЕ Б

Листинг программы

Б.1. Файл main.cpp

```
#include "Registration.h"
#include <ctime>


int main()
{
	// Pointers to Buffers for Target and Source Point Clouds
	pcl::PointCloud<pcl::PointXYZ>::Ptr Target;
	pcl::PointCloud<pcl::PointXYZ>::Ptr Source;

	Matrix4f Transformation;

	ICP Start;

	// Start Reading and Sparse ICP Algorithm
	// Read File
	Target = Start.Read.ReadPCDfile("cat.pcd");

	// Transform Target Point Cloud into Source Point Cloud
for test
	// We want to get two sets of Point Cloud (Initial and
Transformed) for Registration
	// Now Source is transformed Target Point Cloud
	Source = Start.ForTest.Transform(Target);

	Transformation = Start.PointToPoint(Target, Source);
	//Transformation = Start.PCL(Target, Source);

	Start.Viewer.GetView(Target, Source);

	return 0;
}
```

Б.2. Файл IO.h

```
#include <iostream>
#include <string>

#include <Eigen/Dense>

namespace IO
{
	class ReadFile
	{
	public:
		ReadFile();
		~ReadFile();
	};

	class PCDreader : public ReadFile
	{
	public:
		inline const Eigen::MatrixXf
getEigenMatrix(pcl::PointCloud<pcl::PointXYZ>::Ptr PCLcloud,
int Dim)
		{
			Eigen::MatrixXf EigenCloud = PCLcloud-
>getMatrixXfMap(Dim, 4, 0);

			return EigenCloud;
		}

		inline pcl::PointCloud<pcl::PointXYZ>::Ptr
PCDreader::ReadPCDfile(const std::string &filePath)
		{
			pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new
pcl::PointCloud<pcl::PointXYZ>);

			if
(pcl::io::loadPCDFile<pcl::PointXYZ>(filePath, *cloud) == -1)
			{
				PCL_ERROR("Couldn't Read File\n");
			}
			else std::cout << "File Read Successfully!\n";
```

```
            return cloud;
        }

        inline void
GetPCLpointConsole(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
        {
            for (size_t i = 0; i < cloud->points.size();
++i)
            {
                std::cout << cloud->points[i].x << " " <<
cloud->points[i].y << " " << cloud->points[i].z << "\n";
            }
        }

        inline void GetEigenPointConsole(Eigen::MatrixXf
&Target, int iDim, int jDim)
        {
            for (int j = 0; j < jDim; j++)
            {
                for (int i = 0; i < iDim; i++)
                {
                    std::cout << Target(i, j) << " ";
                }

                std::cout << "\n";
            }
        }
    };

    class Visualization
    {
    public:
        inline void
GetView(pcl::PointCloud<pcl::PointXYZ>::Ptr Target,
pcl::PointCloud<pcl::PointXYZ>::Ptr Source)
        {
            pcl::visualization::PCLVisualizer
viewer("Registration");



    pcl::visualization::PointCloudColorHandlerCustom<pcl::Poin
tXYZ> target_handler(Target, 230, 20, 20);
```

```
			viewer.addPointCloud(Target, target_handler, 
"Target");


	pcl::visualization::PointCloudColorHandlerCustom<pcl::Poin
tXYZ> source_handler(Source, 0, 0, 0);

			viewer.addPointCloud(Source, source_handler, 
"Source");

			viewer.setBackgroundColor(255, 255, 255, 0);


	viewer.setPointCloudRenderingProperties(pcl::visualization
::PCL_VISUALIZER_POINT_SIZE, 1.5, "Source");

	viewer.setPointCloudRenderingProperties(pcl::visualization
::PCL_VISUALIZER_POINT_SIZE, 1.5, "Target");

			while (!viewer.wasStopped())
			{
				viewer.spinOnce();
			}
		}
	};
}
```

Б.3. Файл Optimization.cpp

```
#include "Optimization.h"

using namespace Optimizer;


pcl::PointCloud<pcl::PointXYZ>::Ptr 
Transformation::Transform(pcl::PointCloud<pcl::PointXYZ>::Ptr 
Target)
{
	pcl::PointCloud<pcl::PointXYZ>::Ptr Source(new 
pcl::PointCloud<pcl::PointXYZ>);
```

```
    Eigen::Affine3f Transformation =
Eigen::Affine3f::Identity();

    float theta = M_PI;

    Transformation.rotate(Eigen::AngleAxisf(theta,
Eigen::Vector3f::UnitY()));
    //Transformation.rotate(Eigen::AngleAxisf(theta,
Eigen::Vector3f::UnitX()));
    Transformation.translation() << 70.0, 40.0, 90.0;


    pcl::transformPointCloud(*Target, *Source,
Transformation);

    return Source;
}

Matrix4f Transformation::RigidTransform(const MatrixXf
&Target, const MatrixXf &Source)
{
    Vector3f CentroidForS(0,0,0);
    Vector3f CentroidForT(0,0,0);

    MatrixXf CenteredVectorsForS(Source.rows(),3);
    MatrixXf CenteredVectorsForT(Source.rows(),3);

    Matrix4f Transformation = MatrixXf::Identity(4, 4);
    //---------------------------------------------------

    MatrixXf U;
    MatrixXf S;
    MatrixXf V;
    MatrixXf Vt;
    Matrix3f R;
    Vector3f t;

    //----------------------------------- Get Centroids

    //-- Get Sum of coordinates
    for (int i = 0; i < Target.rows(); i++)
    {
        CentroidForT += Target.block<1, 3>(i, 0).transpose();
        CentroidForS += Source.block<1, 3>(i, 0).transpose();
```

```
	}
	//--
	CentroidForS /= Source.rows();
	CentroidForT /= Source.rows();
	//---------------------------------------------------

	//------------------------------------ Get Centered Vectors
	for (int i = 0; i < Source.rows(); i++)
	{

		CenteredVectorsForS.block<1, 3>(i, 0) = 
Source.block<1, 3>(i, 0) - CentroidForS.transpose();
		CenteredVectorsForT.block<1, 3>(i, 0) = 
Target.block<1, 3>(i, 0) - CentroidForT.transpose();
	}
	//---------------------------------------------------------
	S = CenteredVectorsForS.transpose()*CenteredVectorsForT;


	//------------------------------------ Singular Value 
Decomposition of S = AB^T
	JacobiSVD<MatrixXf> svd(S, ComputeFullU | ComputeFullV);

	U = svd.matrixU();
	V = svd.matrixV();

	//------------------------------------ Get Rotation
	R = V*U.transpose();

	if (R.determinant() < 0)
	{
		V.transpose().block<1, 3>(2, 0) *= -1;
		R = V*U.transpose();
	}

	//------------------------------------ Get Translation
	t = CentroidForT - R*CentroidForS;

	Transformation.block<3, 3>(0, 0) = R;
	Transformation.block<3, 1>(0, 3) = t;

	return Transformation;
}
```

Б.4. Файл Optimization.h

```
#pragma once
#include <pcl/point_types.h>
#include <pcl/common/transforms.h>

#include <Eigen/Dense>

using namespace Eigen;

namespace Optimizer
{

	class Transformation
	{
	public:
		pcl::PointCloud<pcl::PointXYZ>::Ptr
Transform(pcl::PointCloud<pcl::PointXYZ>::Ptr Target);

		Matrix4f RigidTransform(const MatrixXf &Target, const
MatrixXf &Source);

	};
}
```

Б.5. Файл Registration.cpp

```
#include <pcl/registration/icp.h>
#include "Registration.h"
#include "Optimization.h"
#include <fstream>

MyRegistration::MyRegistration()
{

}

MyRegistration::~MyRegistration()
{
}
```

```
Matrix4f ICP::PointToPoint(PointCloud<PointXYZ>::Ptr Target,
PointCloud<PointXYZ>::Ptr Source)
{
	EigenTarget = Read.getEigenMatrix(Target, 3);



	pcl::KdTreeFLANN<pcl::PointXYZ> kdtree;
	kdtree.setInputCloud(Target);

	MatrixXf NNMatrix(3, EigenTarget.cols());
	Matrix4f Result;
	MatrixXf Matrix(3, EigenTarget.cols());
	ofstream file("file.txt");
	vector<int> pointIdxNKNSearch(1);
	vector<float> pointNKNSquaredDistance(1);
	vector<float> value(100);
	float error = 0;

	for (int i = 0; i < 50; i++)
	{
		EigenSource = Read.getEigenMatrix(Source, 3);
		//--- kd Nearest Neighbour search
		for (int index = 0; index < Source->size(); index++)
		{

			if (kdtree.nearestKSearch(Source->points[index],
1, pointIdxNKNSearch, pointNKNSquaredDistance) > 0)
			{


				NNMatrix.block<3, 1>(0, index) =
EigenTarget.block<3, 1>(0, pointIdxNKNSearch[0]);

				error = error + pointNKNSquaredDistance[0];

			}
		}

		value[i] = error;
		file << value[i] << "\n";
		cout << error << "\n";
		//system("pause");
```

```
            //--------------------------------

            error = 0;

            Result = ForTest.RigidTransform(NNMatrix.transpose(),
EigenSource.transpose());
            //cout << Result;

            //cout << value; system("pause");
            transformPointCloud(*Source, *Source, Result);

      }

      system("pause");


      return Result;
}

Matrix4f ICP::PCL(PointCloud<PointXYZ>::Ptr Target,
PointCloud<PointXYZ>::Ptr Source)
{
      IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp;

      icp.setInputCloud(Source);

      icp.setInputTarget(Target);

      icp.setMaximumIterations(20);
      PointCloud<pcl::PointXYZ> Final;
      icp.align(Final);


      Matrix4f Result = icp.getFinalTransformation();

      transformPointCloud(*Source, *Source, Result);

      return Result;
}
```

Б.6. Файл Registration.h

```
#pragma once
#include "Optimization.h"
#include "IO.h"

#include <pcl/point_cloud.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <vector>
#include "nanoflann.hpp"


#include <ctime>
#include <cstdlib>

using namespace std;
using namespace pcl;
using namespace Eigen;
using namespace nanoflann;

class MyRegistration
{
public:
	MyRegistration();
	~MyRegistration();

	IO::PCDreader Read;
	IO::Visualization Viewer;
	Optimizer::Transformation ForTest;


protected:

	// Composition for working with necessary class
	string File;
	int MaxIterations = 1;

	MatrixXf EigenTarget;
	MatrixXf EigenSource;

	virtual Matrix4f PointToPoint(PointCloud<PointXYZ>::Ptr
Target, PointCloud<PointXYZ>::Ptr Source) = 0;
```

```
    virtual Matrix4f PCL(PointCloud<PointXYZ>::Ptr Target,
PointCloud<PointXYZ>::Ptr Source) = 0;
};

class ICP : public MyRegistration
{
public:
    // Generalization of Constructor from Registration
    // Init FileName for Reading
    ICP() : MyRegistration(){}

    Matrix4f PointToPoint(PointCloud<PointXYZ>::Ptr Target,
PointCloud<PointXYZ>::Ptr Source);
    Matrix4f PCL(PointCloud<PointXYZ>::Ptr Target,
PointCloud<PointXYZ>::Ptr Source);
};
```

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Дальневосточный федеральный университет»

Инженерная школа
Кафедра автоматизации и управления

ОТЗЫВ РУКОВОДИТЕЛЯ

На выпускную квалификационную работу студента(ки) ______________________
Старкова Артема Дмитриевича
(фамилия, имя, отчество)

Направление подготовки Мехатроника и робототехника

группа Б3421

Руководитель ВКР к.т.н. Губанков А.С.
(ученая степень, ученое звание) (ФИО)

На тему Реализация алгоритма iterative closest point

Дата защиты ВКР «04» июля 2018 г.

Выпускная квалификационная работа Старкова А.Д. посвящена программной реализации одной из модификаций известного алгоритма поиска ближайших точек – ICP (Iterative Closest Points) на основе сингулярного разложения матриц. Актуальность поставленной в работе задачи обусловлена широким распространением этого алгоритма в различных областях робототехники, в частности в промышленной и мобильной.

В работе рассмотрено теоретическое описание алгоритма, выполнен вывод базовых соотношений, подготовлены структуры данных, разработана диаграмма классов. Далее был написан код программы на языке C++, выполнена ее отладка и рассмотрен модельный пример.

В целом выпускная квалификационная работа Старкова А.Д. соответствует заданию, выполнена самостоятельно и на достаточном техническом уровне. За время выполнения работы студент приобрел навыки самостоятельной работы с неизвестным материалом и англоязычной документацией.

Из недостатков можно выделить очень поверхностный анализ существующих разновидностей алгоритма ICP, не всегда логичное и ясное изложение материала при написании пояснительной записки, а также большое количество стилистических ошибок.

Оригинальность текста составляет 96,71% (по данным системы «Антиплагиат», https://www.antiplagiat.ru).

Считаю, что выпускная квалификационная работа заслуживает оценки «удовлетворительно», а студент Старков А.Д. – присвоения квалификации «бакалавр» по направлению 15.03.06 - Мехатроника и робототехника.

Руководитель ВКР ______________________ Губанков А.С.

(подпись) (ФИО)

«15» июня 20 18 г.