МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

Институт кибернетики
Кафедра проблем управления

ОТЧЁТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ

Тема работы: Экспериментальные исследования алгоритма оценки положения и формы объектов

Приказ университета о направлении на практику
от 16.04.2021 №1803-с

Отчет представлен к рассмотрению:

Студент группы КРБО-01-17 _____ П.Р. Кабанов

«__» _____ 2021 г.

Отчет утвержден
Допущен к защите:

Руководитель работы от кафедры _____ С.А.К. Диане
«__» _____ 2021 г.

Руководитель работы от университета _____ А. А. Сухоленцева

«__» _____ 2021 г.

ОТЧЕТ
по преддипломной практике
студента 4 курса учебной группы КРБО-01-17
Института кибернетики

Кабанова Павла Романовича

1. Преддипломную практику с 20.04.2021г. по 17.05.2021г.
В научно-исследовательской лаборатории Г-100 кафедры проблем управления МИРЭА – Российского технологического университета
2. Задание на практику выполнил в полном объеме
Не выполнены следующие задания: -
Подробное содержание выполненной на практике работы и достигнутые результаты: Изучил методы оценки качества аппроксимации облаков точек геометрическими примитивами; серию экспериментов по оценке качества аппроксимации различных объектов в формате облака точек; Ознакомился с возможностями по регистрации облаков точек с бортовой камеры робота KUKA youBot.
3. Предложения по совершенствованию организации и прохождения практики: предложений нет
Студент _____(П.Р. Кабанов) «__»_____2021г.
4. Заключение руководителя практики от профильной организации:
Приобрел следующие профессиональные навыки:

- способность разрабатывать программное обеспечение, необходимое для обработки информации и управления в мехатронных и робототехнических системах, а также для их проектирования
- готовность участвовать в составлении аналитических обзоров и научно-технических отчетов по результатам выполненной работы, в подготовке публикаций по результатам исследований и разработок

Проявил себя как: организованный, целеустремленный специалист, способный самостоятельно решать поставленные перед ним инженерно-технические задачи и представить отчет по практике в полном объеме и в установленный срок и заслуживает оценки «_____».

Руководитель практики от кафедры проблем управления

к.т.н., доцент _____ С.А.К. Диане

Отчет проверил:
Руководитель практики от Университета

_____ А.А. Сухоленцева

Оглавление

Введение

В задачах составления и анализа визуальной одометрии мобильного робота основными способами получения данных являются камера с глубиной или LiDAR. Оба устройства на выходе дают несистематизированное облако точек. Для более качественной оценки окружения робота требуется провести анализ положения и формы окружающих объектов. В данной работе будет рассмотрен способ, с помощью которого можно сегментировать облако точек с помощью набора форм-примитивов.

1 Методы оценки качества аппроксимации облаков точек геометрическими примитивами

1.1 Описание алгоритма программы

В предыдущих работах были изучены и экспериментально протестированы способы поиска объектов в пространстве с помощью алгоритмов RANSAC и эволюционного алгоритма. Были проведены эксперименты с поиском объектов формы параллелепипеда и параллелограмма в трехмерном и двумерном облаке точек с помощью эволюционного алгоритма, а также был протестирован поиск линий с помощью RANSAC. По результатам экспериментов можно сделать следующие выводы:

1) Для работы генетического алгоритма требуется очень тонкая подстройка параметров мутации, числа итераций, а также тщательный выбор способов селекции и скрещивания особей;
2) Алгоритм RANSAC более гибок за счет своей простоты, однако достаточно сильно зависит от случайного выбора точек, попадающих в окрестность поиска;
3) Алгоритм RANSAC оперирует только в пространстве облака точек, в то время как генетический алгоритм оперирует во всей близлежащей

области и может быть ограничен пользователем вручную. Однако это не улучшает положение дел, так как поиск ведется во всем объеме некоторой области, что создает дополнительную неопределенность, так как различных вариантов выбора модели становится в разы больше. В лабораторных тестовых условиях данную проблему можно было решить путем ограничения максимального расстояния, на которое примитив может удалиться от центра масс искомого облака, однако реальные объекты сложных форм как правило имеют центр масс не строго по середине, и искомая форма может быть смещена:

4) Поиск модели параллелепипеда, заданного параметрами координаты вершины, длины, ширины, высоты граней и угла поворота достаточно сложен ввиду большого количества изменяемых параметров. Более того, эти параметры сложно сделать зависимыми от полученного особью результата.

Оценить качество работы алгоритма можно с помощью метрики Intersection over Union (далее – IoU)

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

Или другими словами – отношение объема пересечения к объему объединения. Сравнение следует проводить с размеченным набором данных.

По результатам экспериментов было предложено следующее решение: с помощью сверточной нейронной сети осуществляется поиск целевого объекта на RGB изображении, вырезается область интереса из RGB-D и строится облако точек, в котором с помощью RANSAC производится первичный поиск примитивов, которые впоследствии доводятся до лучшего состояния с помощью генетического алгоритма. Подобный подход должен обеспечить соблюдение баланса между объемом вычислительных мощностей, временем работы алгоритма и эффективностью, что может

позволить использовать его в качестве встраиваемой системы для автономных мобильных роботов.

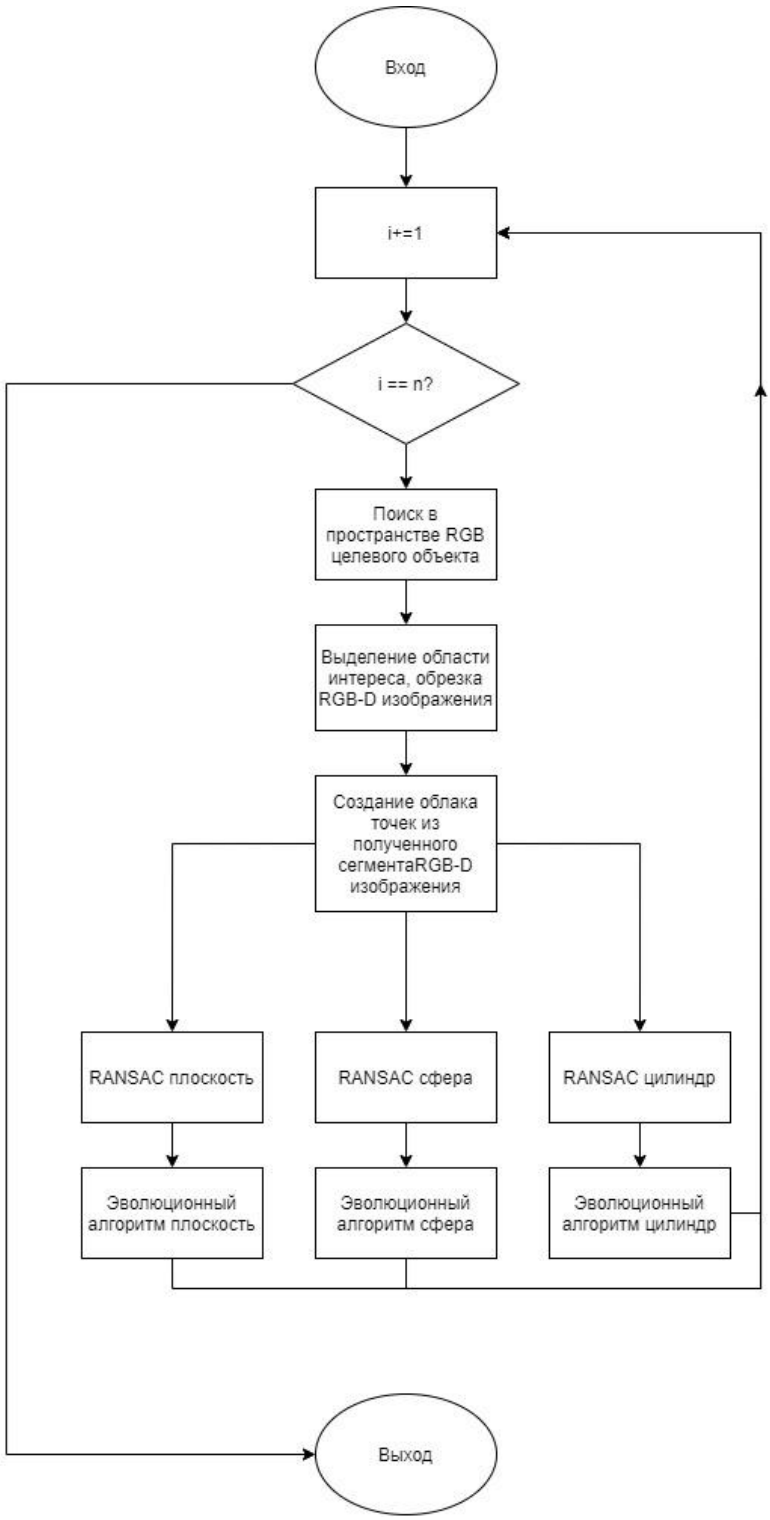

Рисунок 1. Блок-схема программного комплекса для оценки положения и формы объектов

1.1 Алгоритм RANSAC

В качестве основного алгоритма поиска примитивов используется алгоритм RANSAC (Random Sample Consensus). Этот алгоритм позволяет путем случайно выбранных точек задать математическую модель объекта, качественно отфильтровав выбросы и лишние точки. Основной идеей является поиск такой модели, чтобы в окрестность ее математической функции попадало как можно большее количество точек из общего облака. Функция оценки найденной модели имеет вид

$$f = \frac{N_{inliers}}{N_{outliers} + 1},$$

где N inliers – количество точек, попавших в окрестность, N outliers – число точек, не попавших в нее, равно N всех точек на изображении – N inliers. Алгоритм в процессе работы стремится максимизировать функцию.

Как было описано выше, RANSAC имеет недостаток, который состоит в слишком большом влиянии случайных величин на работу алгоритма. Т.к. точки для поиска моделей выбираются случайным образом, нахождение нужной модели может занять продолжительное время из-за длительного перебора точек. Как правило, при реализации алгоритма используется нормальное распределение вероятностей при выборе точек, из-за чего выбор производится вслепую.

В данной работе для решения этой проблемы была использована априорная и апостериорная вероятность при выборе точек. Рассмотрим принцип на примере плоскости.

Для построения уравнения плоскости необходимо выбрать три точки, по которым вычисляются коэффициенты уравнения плоскости в каноническом виде.

1) Выбирается первая точка в пространстве облака, используя нормальное распределение
2) После выбора первой точки производится перераспределение вероятностей выбора остальных точек:

1 Рассчитываются расстояния от выбранной точки A до каждой точки в облаке;

2 Округляются до некоторого порядка (подобрать в зависимости от разрешения входного облака и изображения), производится поиск уникальных элементов, т.е. точек, до которых округленное расстояние одинаково;

3 Рассчитывается дискретное распределение вероятностей (Рис.2), т.е. считается отношение количества повторений n-го элемента списка уникальных элементов к общему числу элементов;

4 Согласно новому распределению выбирается расстояние и набор точек, которые удалены на него от выбранной A;

5 В этом наборе точек случайно выбирается одна, которая становится следующей выбранной точкой B

3) Рассчитывается средняя точка отрезка AB, относительно нее производится шаг 1 и далее.

Такой способ позволяет с большей вероятностью выбрать среди точек, находящихся в определенном радиусе от исходной, что важно для поиска в пространстве облака, где могут быть резкие скачки расстояний между точками, и может помочь с поиском небольших объектов.

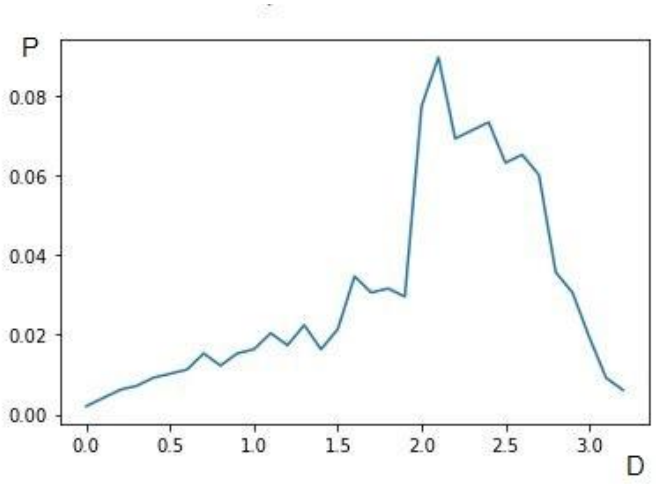

Рисунок 2. Распределение вероятностей при выборе точек в облаке куба

Модификация алгоритма, приведенная в работе, имеет следующую схему (Рис.3):

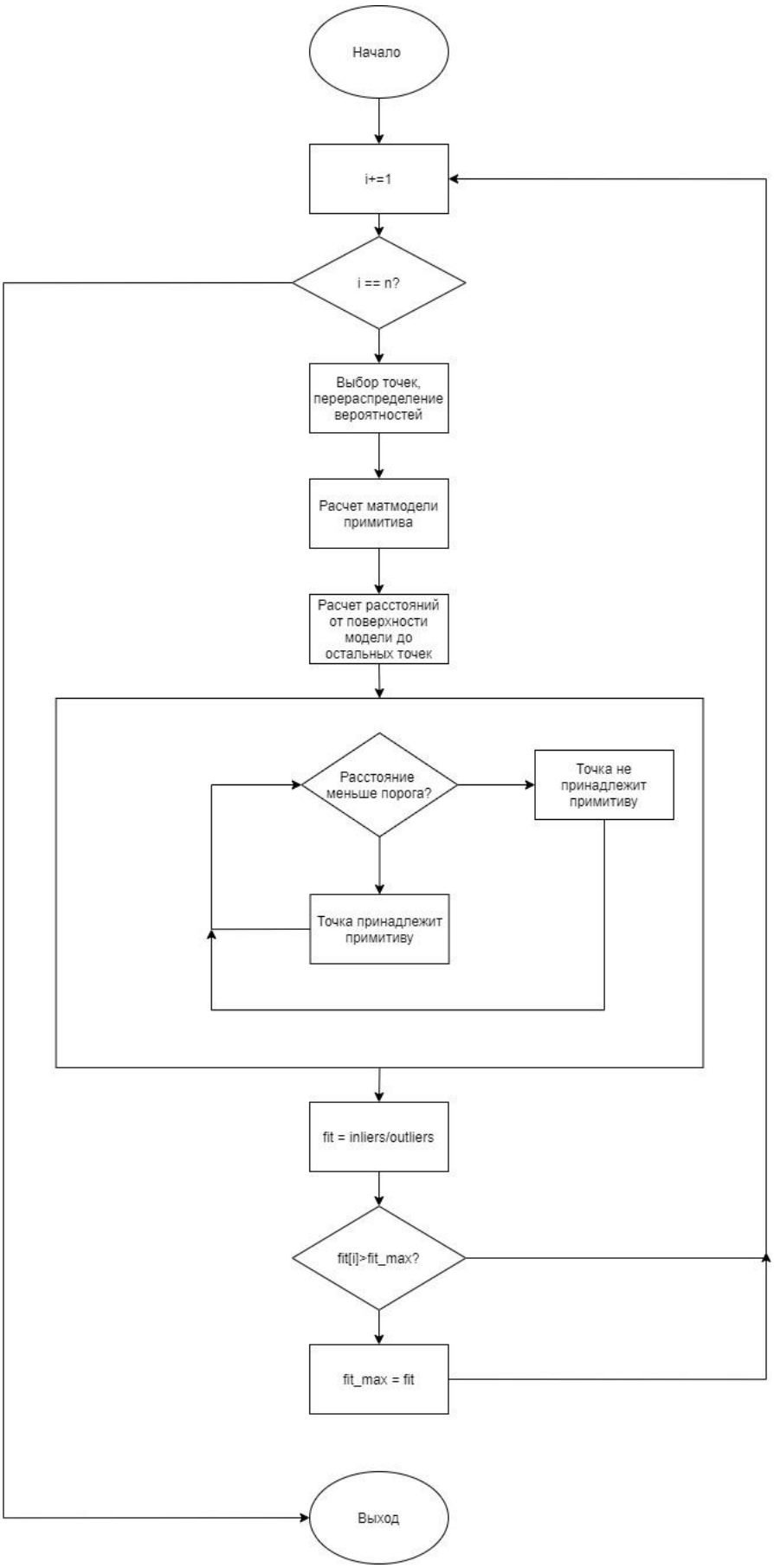

Рисунок 3. Блок-схема алгоритма RANSAC

2 Геометрический анализ объектов

Первичный выбор примитивов основан на выборе таких точек, по которым можно составить их уравнение.

Для плоскости уравнение примет вид

$$Ax + By + Cz + D = 0,$$

Где $A, B, C \neq 0$.

Точки $M0(x0,y0,z0)$, $M1(x1,y1,z1)$, $M2(x2,y2,z2)$ не принадлежат одной плоскости.

Составим уравнение

$$\begin{matrix} x - x0 & x1 - x0 & x2 - x0 \\ y - y0 & y1 - y0 & y2 - y0 \\ z - z0 & z1 - z0 & z2 - z0 \end{matrix} = 0$$

Вычислим миноры по 1 столбцу

$$M1 = \begin{vmatrix} y1 - y0 & y2 - y0 \\ z1 - z0 & z2 - z0 \end{vmatrix}$$

$$M2 = \begin{vmatrix} x1 - x0 & x2 - x0 \\ z1 - z0 & z2 - z0 \end{vmatrix}$$

$$M3 = \begin{vmatrix} x1 - x0 & x2 - x0 \\ y1 - y0 & y2 - y0 \end{vmatrix}$$

Получаем коэффициенты

$A = det(M1)$

$B = -det(M2)$

$C = det(M3)$

$D = -x0\ det(M1) + y0\ det(M2) - z0\ det(M3)$

Расстояние от точки до плоскости можно вычислить по формуле

$$R = \frac{|Ax0 + By0 + Cz0 + D|}{\sqrt{A^2 + B^2 + C^2}}$$

В случае сферы для более качественного результата следует несколько усложнить дальнейшие операции.

Сферу можно описать двумя точками и их нормалями как к реальной поверхности. Нормали ищутся методом главных компонент по ковариационной матрице случайных векторов ближайших к точке соседей.

$$C = \frac{1}{k}\sum_{i=1}^{k}\left(p^i - \overline{p}\right) * \left(p^i - \overline{p}\right)^T, C * \overrightarrow{v_j} = \lambda_j * \overrightarrow{v_j}, j \in \{0,1,2\},$$

Где k – число точек-соседей точки pi, $\bar{p}$ – центроид всех ближайших соседей, λ_j – собственное значение матрицы ковариантности и $\overrightarrow{v_j}$ ее собственный вектор.

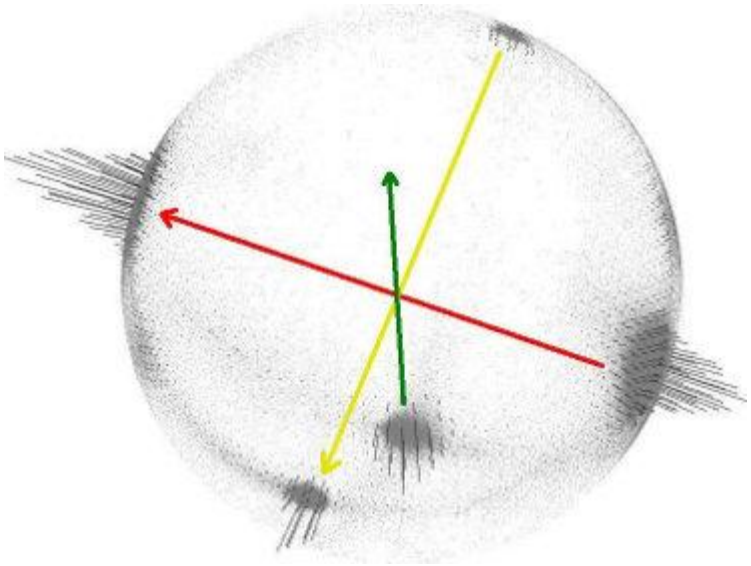

Рисунок 4. Пример нахождения нормалей точек

Точки (p1,p2) – выбранные случайным образом точки, (n1,n2) – их нормали.

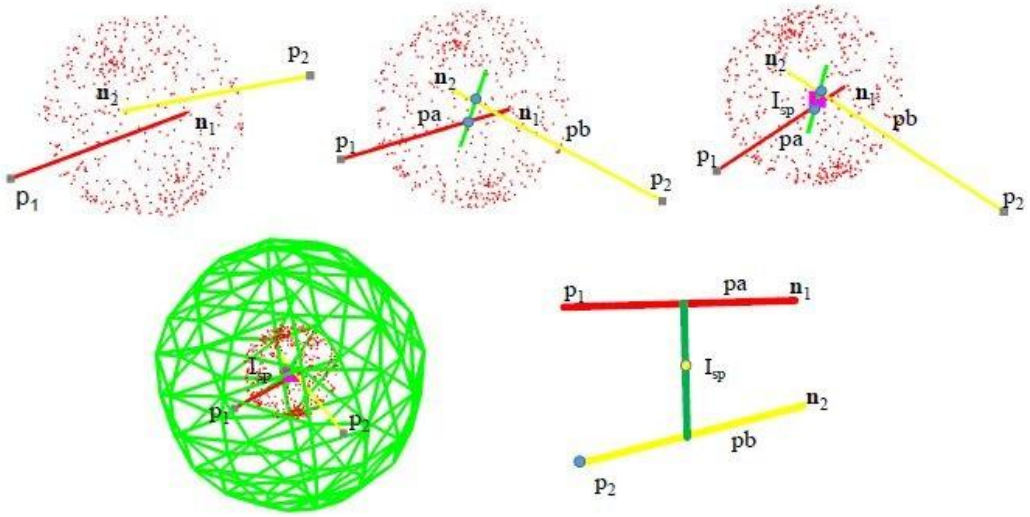

Рисунок 5. Нахождение центра и радиуса сферы по двум точкам и их нормалям

Построим скрещивающиеся прямые (p1,pa) и (p2,pb) в соответствии нормалям и найдем линию с кратчайшим расстоянием между ними. Центр этой линии и примем за центр сферы c. Найдем радиус

$$r = \frac{||p1 - c|| + ||p2 - c||}{2}$$

Так как точки мы выбираем случайным образом, стоит принять некоторую дистанцию A, на которую p1 и p2 могут отклоняться от найденного радиуса. Также перед началом вычислений стоит проверить, что угол между нормалями не превышает заданный угол alpha.

В это случае можно не строить сферу по какому-либо уравнению, достаточно проверять удаленность точек от центра сферы.

3 Использование генетического алгоритма

Как и в предыдущих работах, генетический алгоритм претерпевает мало изменений. Однако в данном случае первоначальное поколение будет

создаваться вокруг найденного примитива, а значения мутации будут несколько снижены, чтобы объект не отходил слишком далеко от оригинала. Это позволит более точно определить ориентацию той или иной фигуры, что должно перекрыть недостатки RANSAC’а.

В качестве генов для плоскости используются коэффициента A,B,C,D, в качестве генов для сферы ее центр и радиус.

Рисунок 6. Блок-схема генетического алгоритма

4 Регистрация облака точек с бортовой камеры робота из RGB-D изображения

Для формирования облака точек с RGB-D изображения используем фреймворк open3d. Формат RGB-D предстваляет собой два изображения: RGB и grayscale изображения. Глубина характеризуется интенсивностью пикселей на черно-белом изображении.

Рисунок 6. RGB канал RGB-D изображения

Рисунок 7. Карта глубины RGB-D изображения

Для качественного формирования облака точек нужно учитывать матрицу параметров внутренней калибровки камеры

$$A = \begin{bmatrix} ax & \Upsilon & u0 \\ 0 & ay & v0 \\ 0 & 0 & 1 \end{bmatrix}$$

Где ax, ay – фокусное расстояние, измеренное в ширине и высоте пикселя, u0, v0 – координаты принципиальной точки, а $\Upsilon = ax * tan\varphi$, где $\varphi -$ угол наклона пикселя.

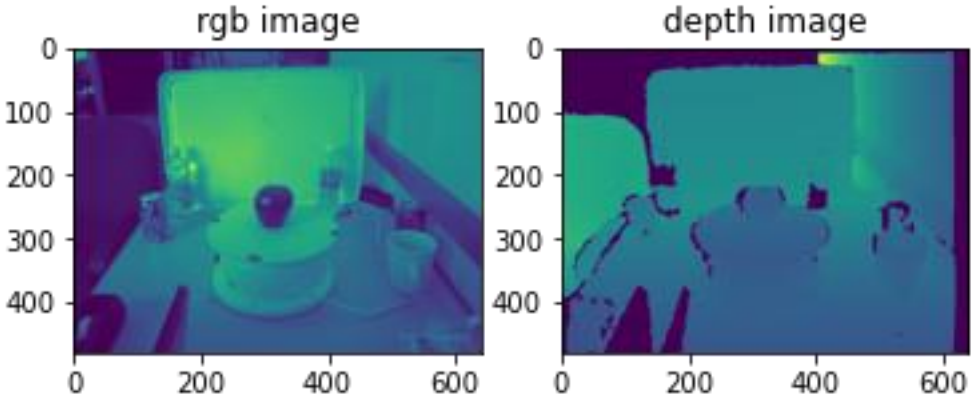

Рисунок 7. Соответствие карты глубины RGB изображению

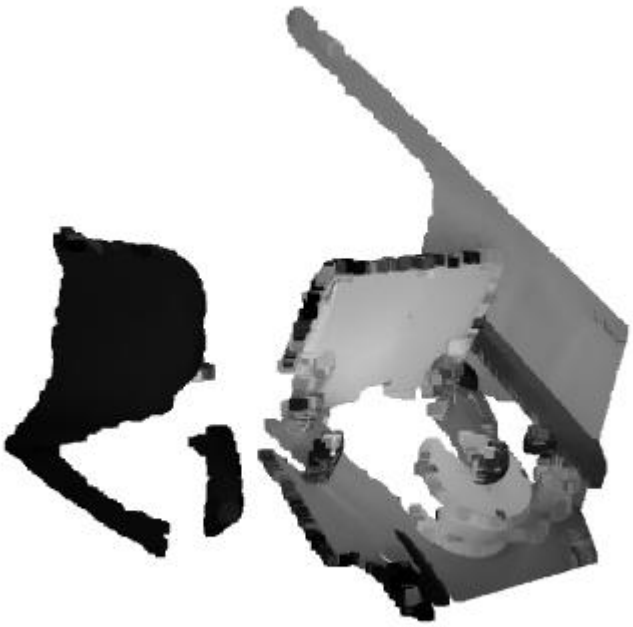

Рисунок 8. Результат создания облака точек

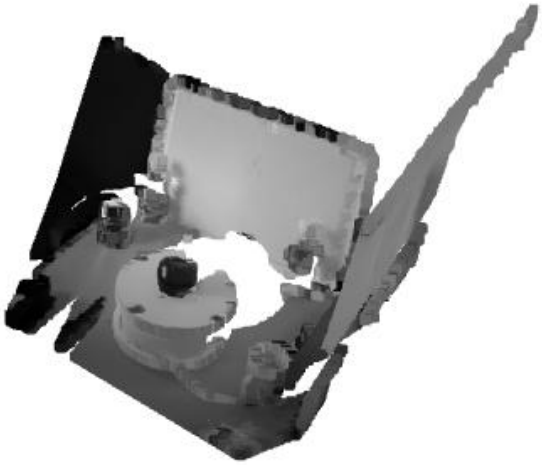

Рисунок 9. Результат создания облака точек

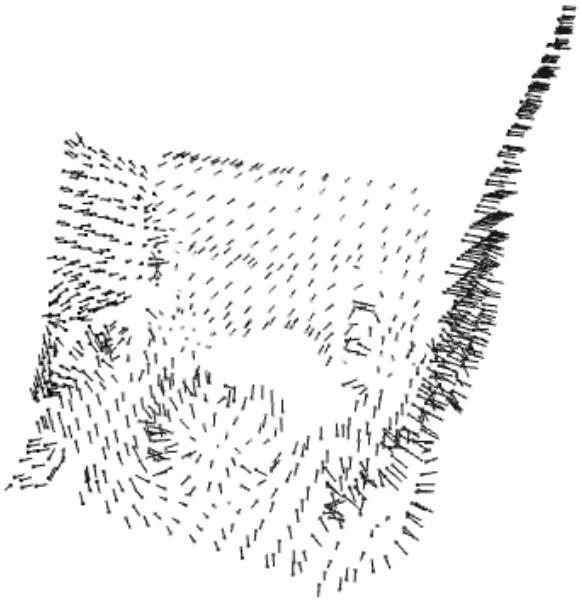

Рисунок 10. Результат построения нормалей точек

5 Проведение экспериментов по оценке качества аппроксимации различных объектов в формате облака точек

Был проведен набор тестов и доказана эффективность алгоритма, как на тестовых отдельных объектах, так и на целых сценах.

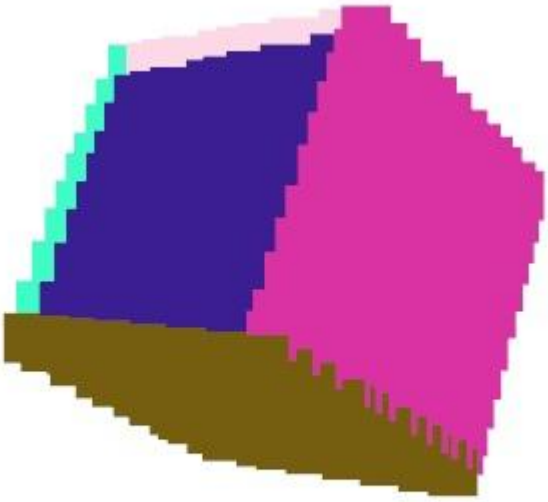

Рисунок 11. Нахождение граней куба

В сравнении с генетическим алгоритмом, используемым ранее, результат стал гораздо точнее и стабильнее.

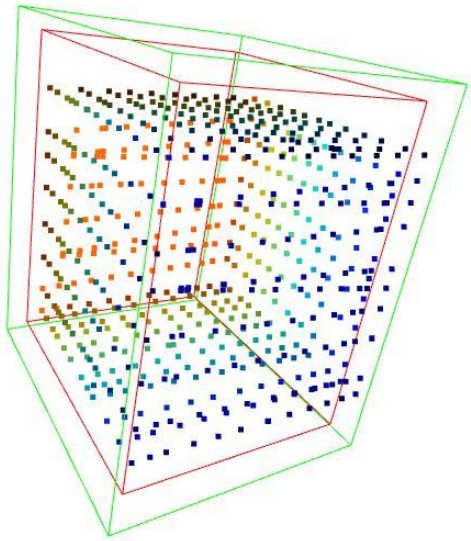

Рисунок 12. Результат работы предыдущей версии алгоритма

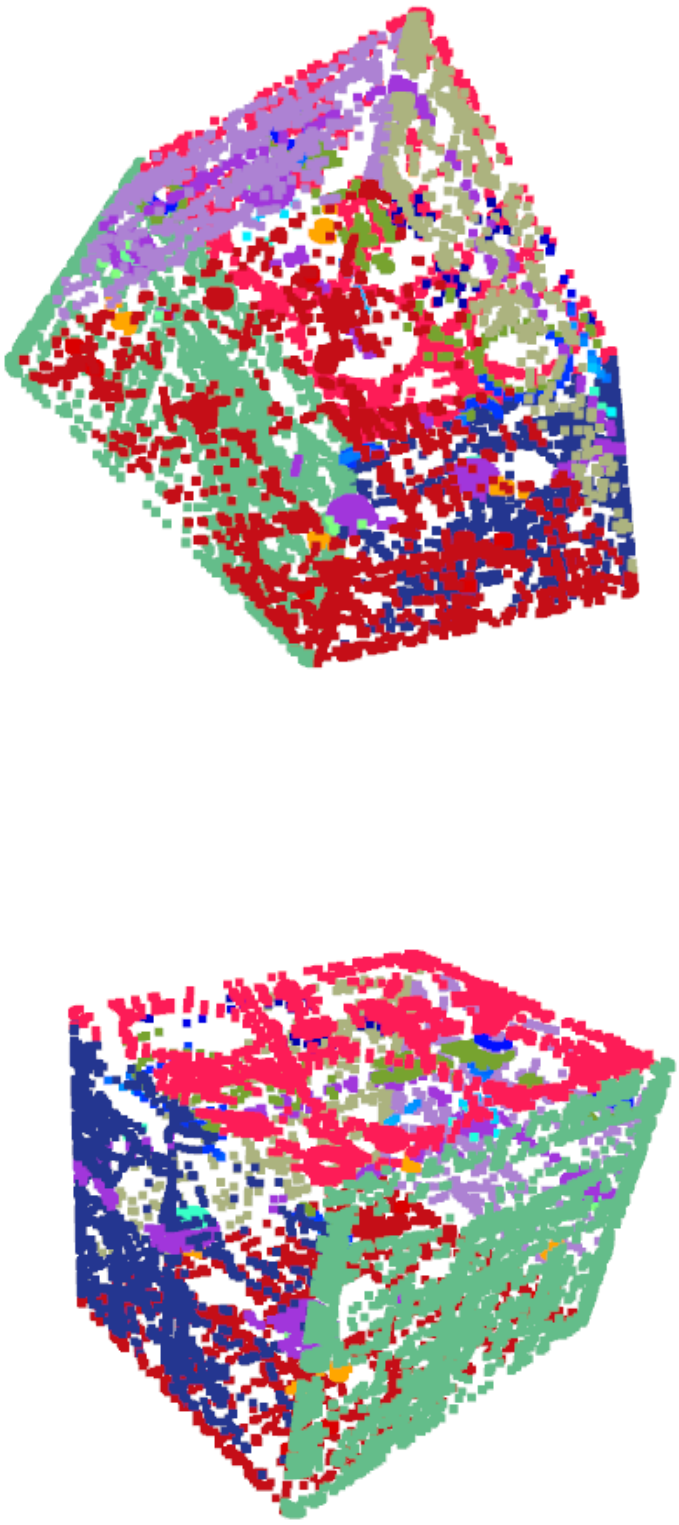

Рисунок 13. Определение деталей на 3D – модели видеокамеры

Также можно продемонстрировать результаты работы алгоритма на больших сценах.

Рисунок 14. Результат работы алгоритма на сцене 1

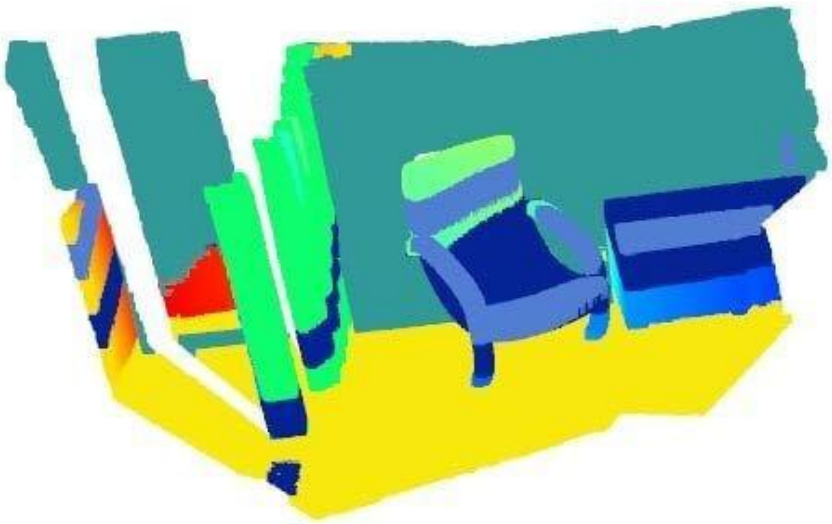

Рисунок 15. Результат работы алгоритма на сцене 1

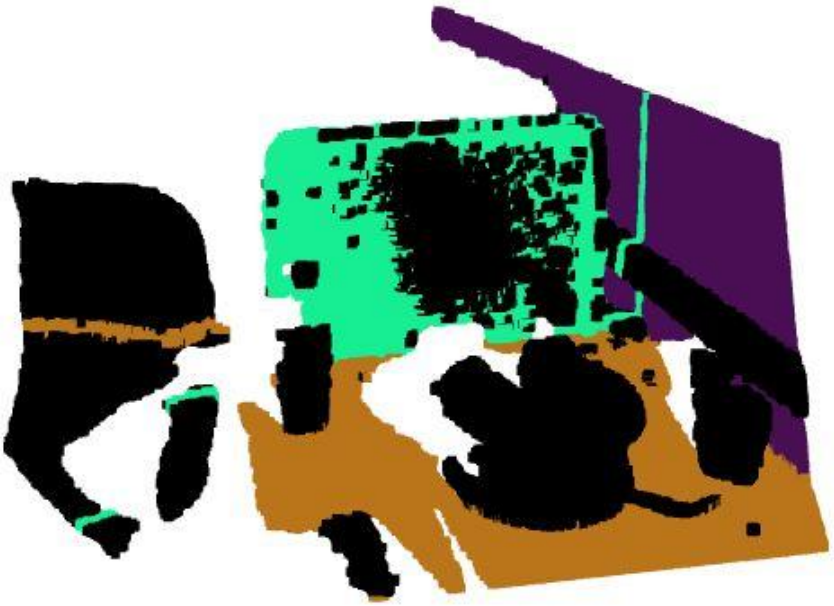

Рисунок 16. Результат работы алгоритма на сцене с Рис. 6

Заключение

По приведенным тестам видно, что алгоритм в приблизительно 85% случаев успешно позволяет найти плоские поверхности в облаке. Однако для качественной сегментации требуется доработать отсечение лишних частей фигур плоскостей, которые пересекают все облако, а также доработать работу со сферами и цилиндрами. План дальнейших исследований – протестировать весь программный комплекс целиком и оценить работу алгоритма с добавленными модификациями и без, а также провести статистические тесты и выяснить стабильность выдаваемых результатов.

Список источников

1. R. Schnabel Efficient RANSAC for Point-Cloud Shape Detection // Bonn University, Computer Graphics Group – 2007
2. S. Xia, D. Chen Geometric Primitives in LiDAR Point Clouds: A Review // IEEE journal of selected topics in applied earth observations and remote sensing – 2020
3. L. Li M. Sung Supervised Fitting of Geometric Primitives to 3D Point Clouds // Computer Vision Foundation – 2019
4. Z. Hang, Z. Li Primitive Fitting Based on the Efficient multiBaySAC Algorithm // School of Land Science and Technology – 2015

Приложение

Листинг кода программы

```
import open3d as o3d
import numpy as np
from numpy.random import default_rng
from matplotlib import pyplot as plt
import os
import cv2
import sys
import math
import random
from time import sleep
import matplotlib.pyplot as plt


def find_hull(points):

    inliers_pcd = o3d.geometry.PointCloud()
    inliers_pcd.points = o3d.utility.Vector3dVector(points)
    #print(type(inliers_pcd))
    hull, _ = inliers_pcd.compute_convex_hull()
    hull_ls = o3d.geometry.LineSet.create_from_triangle_mesh(hull)
    color = np.random.rand(3).tolist()
    hull_ls.paint_uniform_color(color)
    return hull_ls

def color_pts(points):
    inliers_pcd = o3d.geometry.PointCloud()
    inliers_pcd.points = o3d.utility.Vector3dVector(points)
    color = np.random.rand(3)
    #color = np.array([0,0,0])
    point_colors = np.tile(color, (points.shape[0],1))
    inliers_pcd.colors = o3d.utility.Vector3dVector(point_colors)
    return inliers_pcd

class plane:
    def __init__(self, M0, M1, M2):
        A0 = np.linalg.det(np.array([[M1[1]-M0[1],M2[1]-M0[1]],[M1[2]-
M0[2],M2[2]-M0[2]]]))
        A1 = np.linalg.det(np.array([[M1[0]-M0[0],M2[0]-M0[0]],[M1[2]-
M0[2],M2[2]-M0[2]]]))
        A2 = np.linalg.det(np.array([[M1[0]-M0[0],M2[0]-M0[0]],[M1[1]-
M0[1],M2[1]-M0[1]]]))
        self.A = A0
        self.B = -A1
        self.C = A2
        self.D = -M0[0]*A0+M0[1]*A1-M0[2]*A2
    def calc_dist(self, pcd):
        R =
np.absolute(self.A*pcd[:,0]+self.B*pcd[:,1]+self.C*pcd[:,2]+self.D)/(np.sqrt(
self.A**2+self.B**2+self.C**2)+ sys.float_info.epsilon)
        #print(type(R))
        return R

def find_pts_plane(pcd):
    indexA = np.random.randint(0,pcd.shape[0])
    A = pcd[indexA]
    distancesA = np.sqrt((pcd[:,0] - A[0])**2+(pcd[:,1] - A[1])**2+ (pcd[:,2]
- A[2])**2)
    #Calculating B probabilities
    roundedA = np.around(distancesA,1)
```

```
    uniqueB, countsB = np.unique(roundedA, return_counts = True)
    probabilitiesB = countsB/distancesA.shape[0]
    plt.plot(uniqueB, probabilitiesB)
    plt.show()
    choose_thresholded_value_B = np.random.choice(uniqueB, p=probabilitiesB)
    B_candidate_indices = np.argwhere(roundedA==choose_thresholded_value_B)
    B_candidate_indices =
B_candidate_indices.reshape(B_candidate_indices.shape[0])
    indexB = np.random.choice(B_candidate_indices)
    B = pcd[indexB]
    pcd = np.delete(pcd, [indexA,indexB],0)
    #Calculating C probabilities
    mid_AB = np.array([(A[0]+B[0])/2,(A[1]+B[1])/2,(A[2]+B[2])/2])
    distances_mid_AB = np.sqrt((pcd[:,0] - mid_AB[0])**2+(pcd[:,1] -
mid_AB[1])**2+ (pcd[:,2] - mid_AB[2])**2)
    roundedAB = np.around(distances_mid_AB,1)
    uniqueC, countsC = np.unique(roundedAB, return_counts = True)
    probabilitiesC = countsC/distances_mid_AB.shape[0]

    #plt.plot(uniqueC, probabilitiesC)
    #plt.show()
    choose_thresholded_value_C = np.random.choice(uniqueC, p=probabilitiesC)
    C_candidate_indices = np.argwhere(roundedAB==choose_thresholded_value_C)
    C_candidate_indices =
C_candidate_indices.reshape(C_candidate_indices.shape[0])
    indexC = np.random.choice(C_candidate_indices)
    C = pcd[indexC]


    return A, B, C


def find_plane(pcd, iterations, threshold):
    best_fit = 0
    best_inliers_pts = None

    inliers = None
    best_model = list()
    for i in range(iterations):
        A, B, C = find_pts_plane(np.asarray(pcd.points))
        p = plane(A,B,C)
        R = p.calc_dist(np.asarray(pcd.points))
        inliers = np.where(R<=threshold)[0]
        inliers_len = inliers.shape[0] +3
        inliers_pts = np.take(np.asarray(pcd.points),inliers,axis=0)
        outliers = np.asarray(pcd.points).shape[0] - inliers_len
        fit = inliers_len/outliers
        if fit>best_fit:
            best_fit = fit
            print('Found better fintess: ',best_fit, 'at', i )
            best_model = [A,B,C]
            best_inliers_pts = inliers_pts
            best_inlier_indices = inliers

    points = np.array(best_model)
    return best_model, best_inliers_pts, best_inlier_indices

class sphere:
    def __init__(self, M1, M2, N1, N2):
        M1_2 = M1 + N1
        M2_2 = M2 + N2
```

```
pcd = o3d.io.read_point_cloud("models/cube_test1.ply")
#pcd = o3d.io.read_point_cloud("models/fragment.ply")
#pcd = o3d.io.read_point_cloud("aivisum.ply")

#Read rgb-d
'''
color_raw = o3d.io.read_image("models/apple_1_1_12.png")
depth_raw = o3d.io.read_image("models/apple_1_1_12_depth.png")
rgbd_image = o3d.geometry.RGBDImage.create_from_color_and_depth(
    color_raw, depth_raw)
print(rgbd_image)
plt.subplot(1, 2, 1)
plt.title('Redwood grayscale image')
plt.imshow(rgbd_image.color)
plt.subplot(1, 2, 2)
plt.title('Redwood depth image')
plt.imshow(rgbd_image.depth)
plt.show()
pcd = o3d.geometry.PointCloud.create_from_rgbd_image(
    rgbd_image,
    o3d.camera.PinholeCameraIntrinsic(
        o3d.camera.PinholeCameraIntrinsicParameters.PrimeSenseDefault))
# Flip it, otherwise the pointcloud will be upside down
'''
#pcd.transform([[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, 0], [0, 0, 0, 1]])
vis = o3d.visualization.Visualizer()
vis.create_window(width = 800, height = 600)
vis.add_geometry(pcd)
print(pcd)
models_list = []
inliers_list = []
for i in range(1):
    model, inliers, inliers_indices = find_plane(pcd,1,0.09)
    if inliers.shape[0]>15:
        models_list.append(model)
        inliers_list.append(inliers)
        pts = color_pts(inliers)
        vis.add_geometry(pts)
        pcd.points =
o3d.utility.Vector3dVector(np.delete(np.asarray(pcd.points),
inliers_indices,0))
        print(pcd)
color = np.array([0,0,0])
point_colors = np.tile(color, (np.asarray(pcd.points).shape[0],1))
colors = o3d.utility.Vector3dVector(point_colors)
pcd.colors = o3d.utility.Vector3dVector(point_colors)
vis.run()
                                vis.destroy_window()
```