Работа победителя/призера заключительного этапа

Профиль «робототехника»

Матвеев Роман Николаевич

Класс: 11

Город: Чебоксары

Школа: МАОУ "ЛИЦЕЙ № 3"

Регион: Чувашская республика - Чувашия

Уникальный номер участника: 61157359

Команда на заключительном этапе: Gazebo one love

Параллель: 10-11 класс

Результаты заключительного этапа

ИТОГОВЫЙ ПРОТОКОЛ заключительного этапа

по профилю **Робототехника** в 2021/22 учебном году

Все баллы указаны с учётом нормирования

ФИО	Класс	Название команды	Предметный тур, балл по **Математика**	Предметный тур, балл по **Информатика**	Командный тур, балл по командной задаче	Финальный балл	Место
Симаков Михаил Алексеевич	10	Волновая Интерференция	62	40	204,07	306,07	Победитель
Фетисов Игорь Олегович	10	Gazebo one love	30	34,5	223,86	288,36	Победитель
Веретельников Никита Владиславович	11	team-name	11	43,2	221,86	276,06	Призёр
Булгаков Артём Сергеевич	11	Волновая Интерференция	14	52,8	204,07	270,87	Призёр
Матвеев Роман Николаевич	11	Gazebo one love	12	18	223,86	253,86	Призёр
Семёнов Вадим Романович	11	Gazebo one love	12	ЛОЖЬ	223,86	248,26	Призёр
Дубяга Данил Васильевич	11	Волновая Интерференция	10	33,4	204,07	247,47	Призёр
Провоторин Лев Сергеевич	11	team-name	20	0	221,86	241,86	0
Скворцов Алексей	11	team-name	0	0	221,86	221,86	0
Туркия Георгий Ражденович	11	Skills Building	30	28,4	117,21	175,61	0
Зуйков Максим Артемович	11	Ушки Газебы	0	14,8	146,71	161,51	0
Давлитьяров Эмиль Рамильевич	11	Ушки Газебы	0	11,2	146,71	157,91	0
Большаков Владислав Максимович	11	Skills Building	0	34,4	117,21	151,61	0
Ступин Тимур Русланович	10	PythonVSc--	15	14,8	115,29	145,09	0
Миронов Иван Николаевич	10	PythonVSc--	10	18	115,29	143,29	0
Савин Анатолий Антонович	11	Skills Building	0	15,6	117,21	132,81	0
Гутор Елизавета Сергеевна	10	PythonVSc--	0	0	115,29	115,29	0
Варченко Максим	11	Шелезяка	2	40	62,29	104,29	0
Куклин Павел Александрович	11	Безынтеллектуальные человеческие единицы	31	40	31	102	0
Репняк Тимофей Юрьевич	11	Шелезяка	0	18	62,29	80,29	0
Клюкин Александр Вячеславович	10	КомандаБ	10	0	63,21	73,21	0
Сапрыгин Игорь Андреевич	11	Future Gadget Lab v4	25	18	30	73	0
Калмыков Инал Витальевич	10	Future Gadget Lab v4	5	37,2	30	72,2	0
Ростов Николай Владимирович	10	ClosedCV	0	15,6	49	64,6	0
Кулов Владислав	11	КомандаБ	0	0	63,21	63,21	0
Степанов Егор Константинович	11	Шелезяка	0	0	62,29	62,29	0
Краснов Андрей Алексеевич	10	ClosedCV	0	12,4	49	61,4	0
Разон Владислав Юрьевич	11	Безынтеллектуальные человеческие единицы	4	19,6	31	54,6	0
Векшин Арсений Иванович	11	Future Gadget Lab v4	0	11,2	30	41,2	0
Омельяненко Кирилл Денисович	11	Безынтеллектуальные человеческие единицы	2	1,5	31	34,5	0

Дата: _____

Члены жюри и методической комиссии: Подпись:

_____ _____

_____ _____

_____ _____

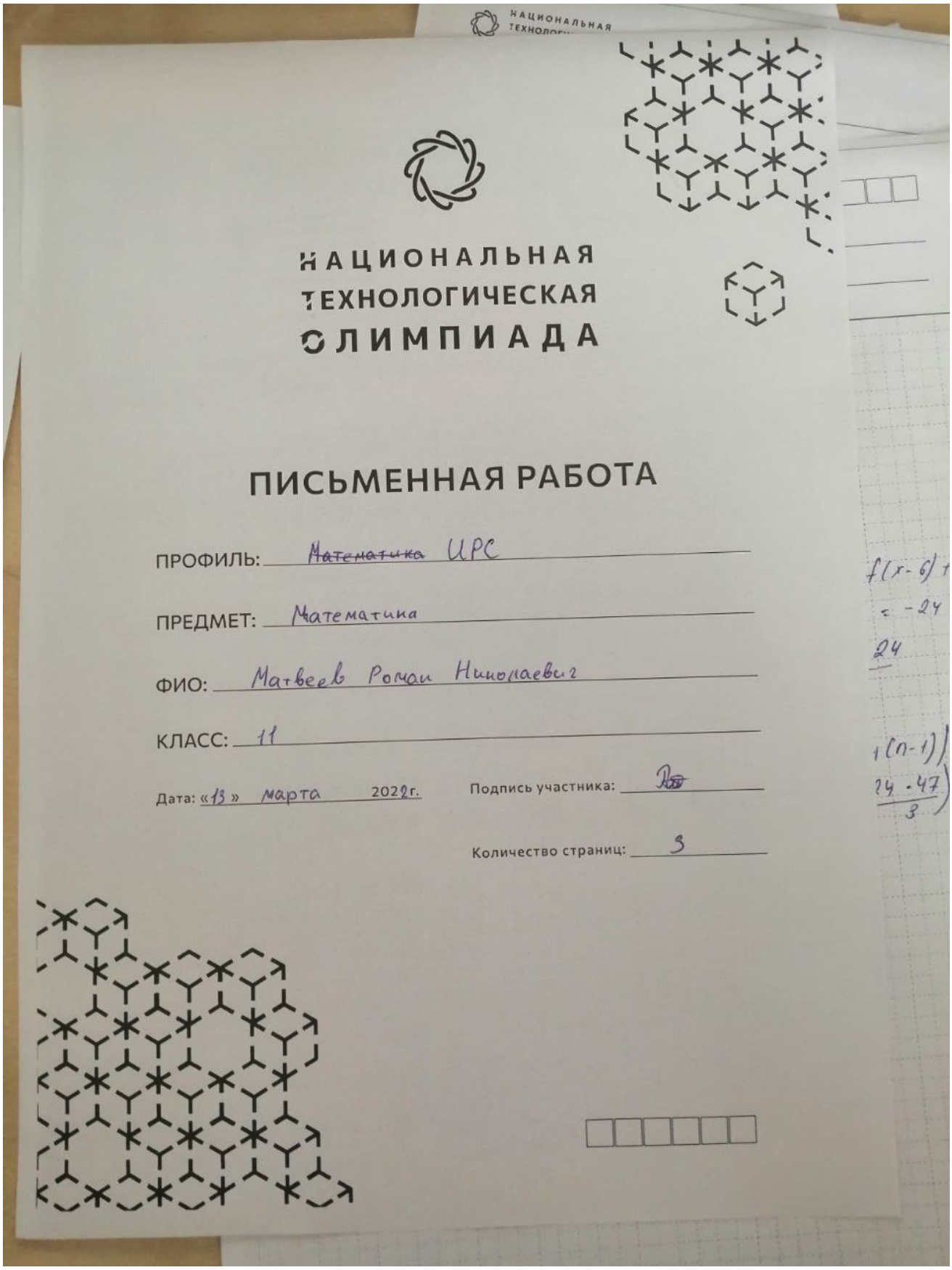
_____ _____

_____ _____

_____ _____

Индивидуальная часть

Персональный лист участника с номером 61157359


НАЦИОНАЛЬНАЯ ТЕХНОЛОГИЧЕСКАЯ ОЛИМПИАДА

ПИСЬМЕННАЯ РАБОТА

ПРОФИЛЬ: ~~Математика~~ ИРС

ПРЕДМЕТ: Математика

ФИО: Матвеев Роман Николаевич

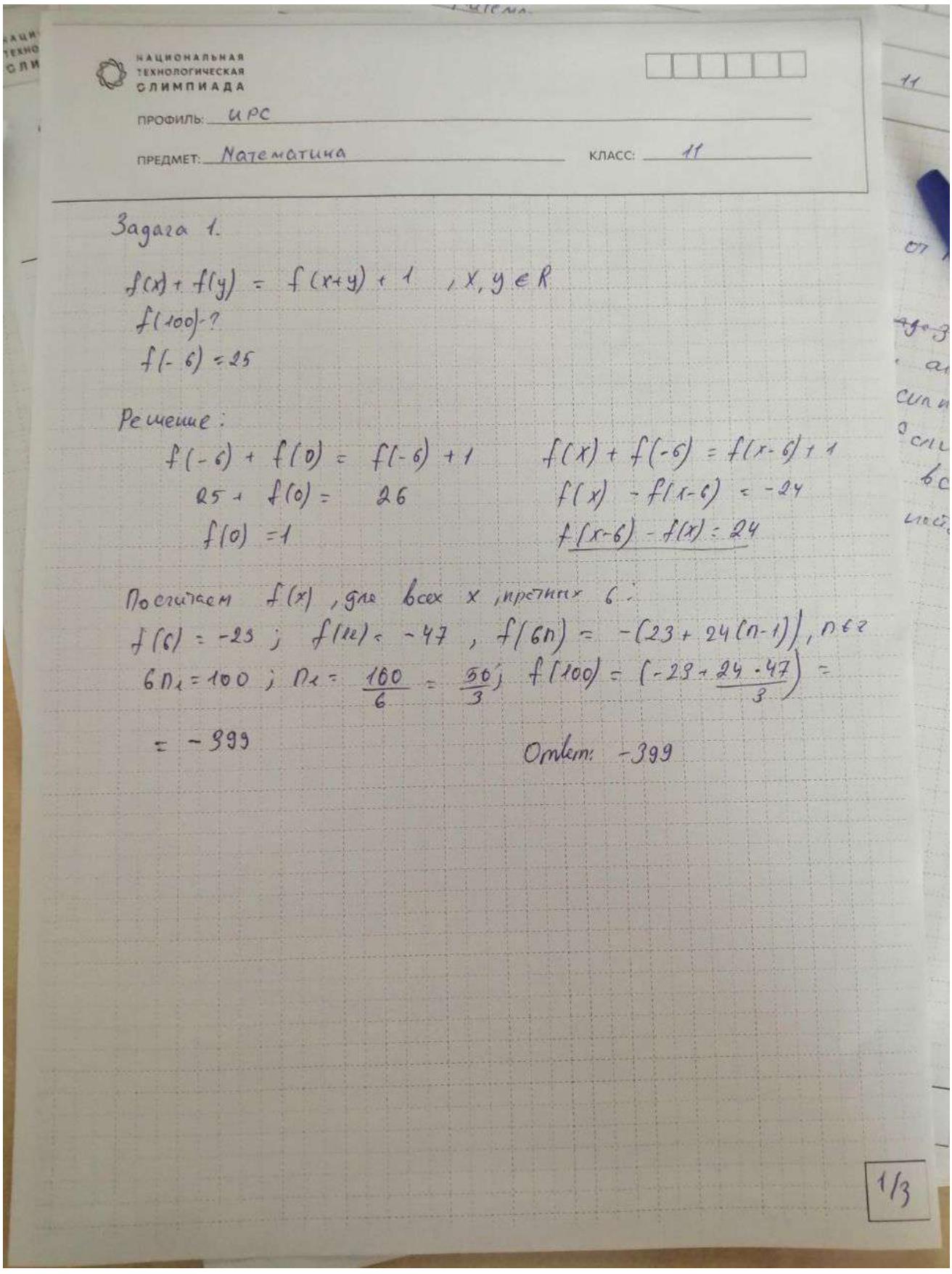
КЛАСС: 11

Дата: «13» марта 2022г. Подпись участника:

Количество страниц: 3

Математика

Лист 1:



НАЦИОНАЛЬНАЯ ТЕХНОЛОГИЧЕСКАЯ ОЛИМПИАДА

ПРОФИЛЬ: ИРС

ПРЕДМЕТ: Математика КЛАСС: 11

Задача 1.

$f(x) + f(y) = f(x+y) + 1$, $x, y \in R$

$f(100) - ?$

$f(-6) = 25$

Решение:

$f(-6) + f(0) = f(-6) + 1$

$25 + f(0) = 26$

$f(0) = 1$

$f(x) + f(-6) = f(x-6) + 1$

$f(x) - f(x-6) = -24$

$f(x-6) - f(x) = 24$

Посчитаем $f(x)$, для всех x, кратных 6:

$f(6) = -23$; $f(12) = -47$, $f(6n) = -(23 + 24(n-1))$, $n \in$

$6n_1 = 100$; $n_1 = \frac{100}{6} = \frac{50}{3}$; $f(100) = \left(-23 + \frac{24 \cdot 47}{3}\right) =$

$= -399$

Ответ: -399

1/3

Лист 2:

НАЦИОНАЛЬНАЯ ТЕХНОЛОГИЧЕСКАЯ ОЛИМПИАДА

ПРОФИЛЬ: ИРС

ПРЕДМЕТ: Математика КЛАСС: 11

Задача 2.

$[x] = a\{x\}$, $a \neq 0$

Пусть $x = \alpha + \beta$, где α - целое число, а $\beta \in [0; 1)$

тогда $\alpha = a\beta$, $\frac{\alpha}{\beta} = a$

$\frac{\alpha_1}{\beta_1}; \frac{\alpha_2}{\beta_2}; \ldots; \frac{\alpha_{2023}}{\beta_{2023}}$; $\alpha + \beta = x$

Если a будет целое, а β_1 не должно быть ≥ 1.

то β_1 может равняться $\frac{1}{2023}$, а $\alpha_1 = 1$

тогда $\frac{\alpha_1}{\beta_1} = 2023$; $\frac{2\alpha_1}{2\beta_1} = 2023$; $\frac{2022\alpha_1}{2022\beta_1} = 2023$.

$\frac{2023\alpha_1}{2023\beta_1}$ быть не может, т.к. $\frac{1}{2023}^{2023} = 1$,

аналогично и для $\frac{2024\alpha_1}{2024\beta_1}$ и больших множителей

Получаем, что $a = 1$ диалогично и для $-\frac{\alpha_1}{\beta_1}$,

только тогда $a = -2023$

Ответ: -2023 и 2023

Лист 3:

НАЦИОНАЛЬНАЯ ТЕХНОЛОГИЧЕСКАЯ ОЛИМПИАДА

ПРОФИЛЬ: ИРС

ПРЕДМЕТ: Математика КЛАСС: 11

Задача 4

Пусть поплавки это список.
~~тогда~~ Известно, что по 3 расстояниям от точек, можно вычислить положение (своё).
~~Возьмём 3 точки и их расстояния, тогда~~ затем ~~построим 3 треугольника~~ тогда отрежем длину половине списка. Переберем с каждым сигналом 3. Вычислим относительные координаты. Если все относительные координаты равны, тогда все поплавки исправны. Так бинарным поиском найдём

Информатика

Задача 1:

```cpp
#include<vector>
#include <iostream>
#include<algorithm>
#include<cmath>
#include<set>
#include<string>
#include<map>
#include <numeric>
#include<queue>
#include <functional>
#include <iomanip>
#include <fstream>
using namespace std;


int main() {
    long long z; cin >> z;
    vector<long long> a;
    a.push_back(0);
    while (z--) {
        string b; cin >> b;
        long long c = 0;
        string cc = "";
        for (int j = 1; j < b.size(); j++) {
            cc += b[j];
        }
        c = stol(cc.c_str());
        if (b[0] == '+') {
            a.push_back(c + a[a.size() - 1]);
        }
        else {
            cout << a[a.size() - 1] - a[a.size() - c - 1]<<endl;
            for (int j = 0; j < c; j++) a.pop_back();
        }
    }

}
```

Задача 2:

```python
n = int(input())
h1 = input().split()
h = []
for t in h1:
```

```
    h.append(int(t))
maxh = -1
sm = 0
for i in h:
    maxh = max(maxh, i)
    sm += i
s = maxh * n - sm

if s % 2 == 0:
    print(s // 2)
else:
    minadd = int(10e10)
    if maxh % 2 == 1:
        minadd = min(minadd, maxh)
    if n % 2 == 1:
        minadd = min(minadd, n)
    if (maxh + n + 1) % 2 == 1:
        minadd = min(minadd, (maxh + n + 1))
    s += minadd
    print(s // 2)
```

Задача 3:

Решение не предоставлено

Задача 4:

Решение не предоставлено

Задача 5:

Решение не предоставлено

Командная часть

Результаты были получены в рамках выступления команды: Gazebo one love

Фотография команды за работой:

Личный состав команды:

- Фетисов Игорь Олегович
- Матвеев Роман Николаевич
- Семенов Вадим Романович

Протокол выполнения заданий:

Этап первый:

Задание	Возможные баллы	Полученные баллы
sim определить цвета кубиков с самой большой и самой маленькой координатой центра по оси y получая на вход облако точек. Вывести 2 строки – цвет кубика с наибольшей координатой центра / цвет кубика с наименьшей координатой центра; цвета RED; BLUE	13	13
Робот с известной конфигурацией начинает свое движение. В достижимой для робота области расположены 2 кубика и 2 зоны выгрузки. Известно расположение объектов; но не их начальная ориентация. Зона выгрузки — обозначенная область; имеющая форму квадрата с длинной стороны равной 5см. и постоянные координаты. Робот смог определить цвета объектов.	6	3
Робот смог захватить и поднять первый объект	12	12
Робот успешно расположил первый объект в зоне любого цвета	10	10
Робот успешно расположил первый объект в зоне нужного цвета	12	6
Робот смог захватить и поднять второй объект	12	12
Робот успешно расположил второй объект в зоне любого цвета	10	10
Робот успешно расположил второй объект в зоне нужного цвета	12	6
Бонус за 1ый день	5	0
Всего за этап:		72

Этап второй:

Задание	**Возможные баллы**	**Полученные баллы**
sim Определить количество синих и красных объектов. Вывести количество синих и красных через пробел. Гарантируется; что все кубики видны с камеры. Допускается их частичное перекрытие (те с камеры объекты могут быть видны не полностью)	15	8
Каждый объект; перемещенный в любую из заранее известных мест. Всего 8. Всего баллов:	32	20
Каждый объект; перемещенный в место нужного цвета. Всего 8.Всего баллов:	32	18
В начальной зоне отсутствуют все объекты.	6	3
Бонус за 2ой день	6	0
Всего за этап:		49

Этап третий:

Задание	**Возможные баллы**	**Полученные баллы**
sim Определить в какой зоне находится новый объект? Гарантируется что данный объект находится полностью в этой зоне. Предварительно зоны определяются по координатным четвертям в плоскости XY.	15	14
Каждый объект; перемещенный в любую из заранее известных мест. Всего 8. Всего баллов:	32	16
Каждый объект; перемещенный в место нужного цвета. Всего 8.Всего баллов:	32	14
В начальной зоне отсутствуют все объекты (объект нового вида не в счет).	6	0
Отличающийся объект остался в начальной зоне. Баллы начисляются даже если 0 баллов за иное.	12	6
Бонус за 3ий день	3	0
Всего за этап:		50

Этап четвертый:

Задание	**Возможные баллы**	**Полученные баллы**
sim посчитать сколько объектов не касается основания. Гарантируется что все кубики видны на камере.	20	12.86
Объект перемещен в любую из заранее известных мест по цвету. Всего 10. Всего баллов:	40	24
В данном месте отсутствуют объекты другого цвета. Всего 2. Всего баллов:	16	16
Объект не касается основания; те опирается только на другие объекты; своего цвета. Всего 8. Всего баллов:	32	0
В начальной зоне отсутствуют все объекты.	6	0
Бонус за 4ый день	3	0
Всего за этап:		52.86

Исходный код программы команды для решения задачи:

Файл src/dots3d.py

```
from math import degrees, sqrt, acos, pi
import numpy as np
import cv2
import numpy as np
import open3d as o3d
from time import sleep, time

from OperateCamera import OperateCamera
from OperateRobot import OperateRobot


SHOW_IMG_ACCESS = 1
SAVE_IMG_ACCESS = 1

def show_img(mtx):
        if SHOW_IMG_ACCESS == 0: return
        from PIL import Image
        Image.fromarray(np.array(mtx, dtype=np.uint8)).show()
def save_img(mtx, name=1):
        if SAVE_IMG_ACCESS == 0: return
        import cv2
        cv2.imwrite(rf"{name}.png", cv2.cvtColor(np.array(mtx,
        ↪  dtype=np.uint8), cv2.COLOR_BGR2RGB))


class Tools:
        def __init__(self) -> None:
                self.red_low_1 = [0, 0.3, 0.35]
                self.red_high_1 = [26, 1, 1]
                self.red_low_2 = [340,0.4, 0.4]
                self.red_high_2 = [360, 1, 1]

                self.blue_low = [185, 0.3, 0.2]
                self.blue_high = [245, 1, 1]

        def rotate_image(self, image, angle):
                image_center = tuple(np.array(image.shape[1::-1]) / 2)
                rot_mat = cv2.getRotationMatrix2D(image_center, angle, 1.0)
                result = cv2.warpAffine(image, rot_mat, image.shape[1::-1],
                ↪  flags=cv2.INTER_LINEAR)
                return result

        def get_pic_from_file(self, path):
                pcd = o3d.io.read_point_cloud(path)

                return pcd
```

```
45
46          def dist(self, p1, p2):
47                  y1, x1 = p1
48                  y2, x2 = p2
49                  return sqrt((y2 - y1) * (y2 - y1) + (x2 - x1) * (x2 - x1))
50
51          def get_next_pnts(self, curPnt, short_set=True):
52                  nextPnts = []
53
54                  if curPnt[0] != 0:
55                          nextPnts.append((curPnt[0] - 1, curPnt[1]))
56                  if curPnt[1] != 0:
57                          nextPnts.append((curPnt[0], curPnt[1] - 1))
58                  if curPnt[0] != self.HEIGHT-1:
59                          nextPnts.append((curPnt[0] + 1, curPnt[1]))
60                  if curPnt[1] != self.WIDTH-1:
61                          nextPnts.append((curPnt[0], curPnt[1] + 1))
62                  if short_set:
63                          return nextPnts
64
65                  # will return 8 points around curPnt
66                  if curPnt[0] != 0 and curPnt[1] != 0:
67                          nextPnts.append((curPnt[0] - 1, curPnt[1] - 1))
68                  if curPnt[0] != 0 and curPnt[1] != self.WIDTH-1:
69                          nextPnts.append((curPnt[0] - 1, curPnt[1] + 1))
70                  if curPnt[0] != self.HEIGHT-1 and curPnt[1] != 0:
71                          nextPnts.append((curPnt[0] + 1, curPnt[1] - 1))
72                  if curPnt[0] != self.HEIGHT-1 and curPnt[1] != self.WIDTH-1:
73                          nextPnts.append((curPnt[0] + 1, curPnt[1] + 1))
74                  return nextPnts
75
76          def sort_clockwise(self, points, center=[0, 0]):
77                  """ points: [[y, x], [y, x], ..]
78                  Just sorts given points clockwise.
79                  """
80                  vectors = [[vec[0] - center[0], vec[1] - center[1]] for vec in
                    ↪  points]
81                  axe_v = [0, 1000]
82
83                  angles_dict = {}
84                  for vec in vectors:
85                          if vec[0] <= 0:
86                                  k1 = -1
87                          else:
88                                  k1 = 1
89                          cosinus = (vec[1] * axe_v[1] + vec[0] * axe_v[0]) /
                            ↪  (sqrt(vec[1] * vec[1] + vec[0] * vec[0]) *
                            ↪  sqrt(axe_v[1] * axe_v[1] + axe_v[0] * axe_v[0]))
90
91                          if cosinus < -1:
```

```
                                cosinus = -1
                        if cosinus > 1:
                                cosinus = 1
                        if k1 * acos(cosinus) not in angles_dict:
                                angles_dict[k1 * acos(cosinus)] = []
                        angles_dict[k1 * acos(cosinus)].append(vec)

                sorted_angles = sorted(angles_dict)
                clockwise = [[point[0] + center[0], point[1] + center[1]] for
                ↪  angle in sorted_angles for point in angles_dict[angle]]
                return clockwise

        def rgb2hsv(self, pixel):
                """Don't use this method outside the function.
                """
                r = pixel[0] / 255
                g = pixel[1] / 255
                b = pixel[2] / 255
                maxC = max([r, g, b])
                minC = min([r, g, b])

                diff = maxC - minC
                if diff == 0:
                        hue = 0
                elif maxC == r:
                        hue = 60 * ((g - b) / diff % 6)
                elif maxC == g:
                        hue = 60 * ((b - r) / diff + 2)
                elif maxC == b:
                        hue = 60 * ((r - g) / diff + 4)

                if maxC == 0:
                        sat = 0
                else:
                        sat = diff / maxC

                return [hue, sat, maxC]

        def is_red(self, pixel):
                """Takes RGB pixel as input.
                """
                pixel = self.rgb2hsv(pixel)
                for i in range(3):
                        if not (((self.red_low_1[i] <= pixel[i]) and (pixel[i]
                        ↪  <= self.red_high_1[i])) or ((self.red_low_2[i] <=
                        ↪  pixel[i]) and (pixel[i] <= self.red_high_2[i]))):
                                return False
                return True

        def is_blue(self, pixel):
```

```
                """Takes RGB pixel as input.
                """
                pixel = self.rgb2hsv(pixel)
                for i in range(3):
                        if not ((self.blue_low[i] <= pixel[i]) and (pixel[i]
                        ↪  <= self.blue_high[i])):
                                return False
                return True

        def most_common_color(self, area):
                red = 0
                blue = 0
                for (y, x) in area:
                        color = self.color_name(self.mt_color[y][x])
                        if (color == [255, 0, 0]):
                                red += 1
                        elif (color == [0, 0, 255]):
                                blue += 1
                if (red > blue):
                        return [255, 0, 0]
                else:
                        return [0, 0, 255]

        def color_name(self, pixel):
                if (pixel == [255, 0, 0]):
                        return "RED"
                elif (pixel == [0, 0, 255]):
                        return "BLUE"
                else:
                        return "UNKNOWN COLOR"


class Object(Tools):
        def __init__(self, points, heights, img, color=[0, 0, 0]) -> None:
                self.area = points #Point [(y, x), (y, x), ....]
                if (type(heights) is dict):
                        self.heights = dict(heights)
                else:
                        self.heights = {}
                        for i, point in enumerate(points):
                                self.heights[point] = heights[i]
                self.bare_img = [[[i for i in x] for x in line] for line in
                ↪  img]
                self.color = color

                self.y_size = len(img)

                self.contourxy = self.get_contourxy()
                self.contourz  = self.get_contourz()

```

```
187                 #self.centery, self.centerx, self.centerz = self.get_center()
188
189                 self.normal = self.get_normal()
190
191                 img = cv2.cvtColor(np.array(img, dtype=np.uint8),
                    ↪  cv2.COLOR_RGB2GRAY)
192                 cvcontours = cv2.findContours(img, cv2.RETR_TREE,
                    ↪  cv2.CHAIN_APPROX_SIMPLE)[0]
193                 cnt = sorted(cvcontours, key=cv2.contourArea, reverse=True)[0]
194                 rect = cv2.minAreaRect(cnt)
195                 self.box = np.int0(cv2.boxPoints(rect))
196
197                 center = np.round(sum(self.box) / 4)
198                 self.centerx, self.centery = map(int, center)
199
200                 self.centerz = sum(self.heights.values()) / len([i for i in
                    ↪  heights if i != 0]) # medium between all the heights of
                    ↪  the surface
201                 #default y, x, z
202                 self.globaly=-0.20
203                 self.globalx=-0.80
204                 self.globalz=0.65
205                 self.angle = None
206
207                 # get rotation angle
208                 self.box = [[y, x] for x, y in list(self.box)] # to (y, x)
209                 sorted_corners = self.sort_clockwise(self.box, [self.centery,
                    ↪  self.centerx])
210
211                 y_low_coord = max(sorted_corners, key=lambda x: x[0])
212
213                 ind = sorted_corners.index(y_low_coord) # find lowest point
214                 sorted_corners = sorted_corners[ind:] + sorted_corners[:ind]
215
216                 a = self.dist(sorted_corners[0], sorted_corners[1])
217                 b = self.dist(sorted_corners[0], sorted_corners[3])
218                 if (a > b):
219                         # tilted to the left
220                         self.angle = acos((sorted_corners[1][1] -
                            ↪  y_low_coord[1]) / a)
221                 else:
222                         # tilted to the right
223                         self.angle = acos((sorted_corners[3][1] -
                            ↪  y_low_coord[1]) / b)
224
225         def get_normal(self):
226                 xs = [i[1] for i in self.area]
227                 ys = [i[0] for i in self.area]
228                 zs = [-self.heights[i] for i in self.area]
229
```

```
230                 # do fit
231                 tmp_A = []
232                 tmp_b = []
233                 for i in range(len(xs)):
234                         tmp_A.append([xs[i], ys[i], 1])
235                         tmp_b.append(zs[i])
236                 b = np.matrix(tmp_b).T
237                 A = np.matrix(tmp_A)
238                 fit = (A.T * A).I * A.T * b
239                 errors = b - A * fit
240                 residual = np.linalg.norm(errors)
241                 A, B, C = -float(fit[0]), -float(fit[1]), 1
242                 A, B, C = A/np.sqrt(A*A+B*B+C*C), B/np.sqrt(A*A+B*B+C*C),
                    ↪  C/np.sqrt(A*A+B*B+C*C)
243                 normal = (A, B, C)
244                 return normal
245
246         def get_contourxy(self):
247                 points = set(self.area)
248                 contourxy = []
249                 for point in points:
250                         close_points = []
251                         for i in range(-1, 2):
252                                 for j in range(-1, 2):
253                                         if i * i + j * j == 1:
                                            ↪  close_points.append((point[0] + i,
                                            ↪  point[1] + j))
254                         x = 0
255                         for close_point in close_points:
256                                 if close_point in points:
257                                         x += 1
258                         if x != 4:
259                                 contourxy.append(point)
260                 return contourxy
261
262         def get_contourz(self):
263                 contourz = []
264                 for point in self.contourxy:
265                         contourz.append(self.heights[point])
266                 return contourz
267
268         def get_center(self):
269                 centery = centerx = 0
270                 for y, x in self.contourxy:
271                         centery += y
272                         centerx += x
273                 centery = int(np.round(centery / len(self.contourxy)))
274                 centerx = int(np.round(centerx / len(self.contourxy)))
275                 return centery, centerx, self.heights[(centery, centerx)]
276
```

```
277         def visualize(self):
278                 import open3d as o3d
279                 x, y, z = [], [], []
280                 cr, cg, cb = [], [], []
281 
282                 x.append(self.centerx)
283                 y.append(self.centery)
284                 z.append(self.centerz)
285                 cr.append(0)
286                 cg.append(0)
287                 cb.append(255)
288 
289                 for i ,j in self.area:
290                         x.append(j)
291                         y.append(i)
292                         z.append(self.heights[(i, j)])
293                         if((i, j) not in self.contourxy):
294                                 cr.append(self.color[0])
295                                 cg.append(self.color[1])
296                                 cb.append(self.color[2])
297                         else:
298                                 cr.append(0)
299                                 cg.append(0)
300                                 cb.append(0)
301 
302                 for i in range(15):
303                         x.append(self.centerx + self.normal[0] * i)
304                         y.append(self.centery + self.normal[1] * i)
305                         z.append(self.centerz - self.normal[2] * i)
306                         cr.append(0)
307                         cg.append(255)
308                         cb.append(0)
309 
310                 for i in range(-100, 100):
311                         x.append(self.centerx + i)
312                         y.append(self.centery)
313                         z.append(self.centerz + 50)
314                         cr.append(0)
315                         cg.append(0)
316                         cb.append(0)
317 
318                 for i in range(-100, 100):
319                         x.append(self.centerx)
320                         y.append(self.centery + i)
321                         z.append(self.centerz + 50)
322                         cr.append(0)
323                         cg.append(0)
324                         cb.append(0)
325 
326                 points = np.vstack((x, y, z)).transpose()
```

```
327                 colors = np.vstack((cr, cg, cb)).transpose()
328 
329                 pcd = o3d.geometry.PointCloud()
330                 pcd.points = o3d.utility.Vector3dVector(points)
331                 pcd.colors = o3d.utility.Vector3dVector(colors/255)
332                 o3d.visualization.draw_geometries([pcd])
333 
334         def view_img(self):
335                 test_img = np.array([[list(i) for i in line] for line in
                    ↪  self.source_img])
336                 cv2.drawContours(test_img, [self.box], 0, (0, 255, 0), 1)
337                 show_img(test_img)
338 
339         def check_around(self, trinarised_img):
340                 img = np.array(trinarised_img, dtype=np.uint8)
341 
342                 box = np.array(self.box)
343                 A, B, C, D = box
344                 center = np.array([self.centery, self.centerx])
345 
346                 A += np.int0((A - center) * 1)
347                 B += np.int0((B - center) * 1)
348                 C += np.int0((C - center) * 1)
349                 D += np.int0((D - center) * 1)
350 
351                 img = img[min(A, B, C, D, key=lambda x: x[0])[0]:max(A, B, C,
                    ↪  D, key=lambda x: x[0])[0], min(A, B, C, D, key=lambda x:
                    ↪  x[1])[1]:max(A, B, C, D, key=lambda x: x[1])[1]]
352                 img = self.rotate_image(img, -degrees(self.angle))
353 
354                 contours, _ = cv2.findContours(cv2.cvtColor(img,
                    ↪  cv2.COLOR_RGB2GRAY), cv2.RETR_EXTERNAL,
                    ↪  cv2.CHAIN_APPROX_SIMPLE)
355                 cnt = sorted(contours, key=cv2.contourArea, reverse=True)[0]
356                 rect = cv2.minAreaRect(cnt)
357                 box = np.int0(np.round(cv2.boxPoints(rect)))
358 
359                 O = np.int0(np.round(sum(box) / 4))
360                 O = [O[1], O[0]]
361                 box_list = self.sort_clockwise([(y, x) for x, y in box], O)
362 
363                 y_low_coord = min(box_list, key=lambda x: self.dist(x, (0,
                    ↪  0)))
364 
365                 ind = box_list.index(y_low_coord) # find lowest point
366                 box_list = box_list[ind-1:] + box_list[:ind-1]
367 
368                 K, L, M, N = box_list
369                 height, width, _ = img.shape
370 
```

```
371                 # top and down
372                 top = np.count_nonzero(img[0:min(L[0], M[0]) - 5, L[1]:M[1]])
373                 down = np.count_nonzero(img[max(K[0], N[0]) + 5:height,
                    ↪  K[1]:N[1]])
374                 # print(top, down)
375                 if (top + down < 100):
376                         return 1
377
378                 # left and right
379                 left = np.count_nonzero(img[L[0]:K[0], 0:min(L[1], K[1]) - 5])
380                 right = np.count_nonzero(img[M[0]:N[0], max(M[1], N[1]) +
                    ↪  5:width])
381                 if (left + right < 100):
382                         return 2
383                 # print(left, right)
384                 return 0
385
386         def get_angle(self, trinarised_img):
387                 result = self.check_around(trinarised_img)
388
389                 if (result == 1):
390                         return (True, pi / 2 - self.angle, 1)
391                 elif (result == 2):
392                         if (self.angle >= pi / 2):
393                                 return (True, pi - self.angle, 2)
394                         else:
395                                 return (True, -self.angle, 2)
396                 return (False, pi / 2 - self.angle, 1)
397
398
399
400 class HeightMatrixProc:
401         """Functions for heights matrix processing.
402         """
403         def get_area_h(self, pixel):
404                 """pixel = (y, x)
405                 Returns area near the given point which differ from (y, x)
    ↪  coordinate not more than COEFF.
406                 """
407
408                 COEFF = 8
409                 height = self.mt_height[pixel[0]][pixel[1]]
410
411                 visited = set([pixel])
412                 queue = set([pixel])
413                 heights_area = [pixel]
414
415                 while queue:
416                         nextPnts = self.get_next_pnts(queue.pop())
417                         for next_point in nextPnts:
```

```
                                if (next_point not in visited):
                                        visited.add(next_point)
                                        # we can also do another one thing
                                        ↪ here: check if inverse of matcher
                                        ↪ is not
                                        ↪ self.mt_color[next_point[0]][next_point[1]]
                                        if
                                        ↪ abs(self.mt_height[next_point[0]][next_point
                                        ↪ - height) < COEFF and [255, 0, 0]
                                        ↪ ==
                                        ↪ self.mt_color[next_point[0]][next_point[1]]:
                                            queue.add(next_point)
                                            heights_area.append(next_point)


class RobotControl:
        def get_pic_from_robot():
                cam = OperateCamera()

                return cam.catch_frame()


class MatrixProcessing(Tools):
        def __init__(self, path=None, path_to_img=None, name=1):
                super().__init__()

                if (path):
                        self.PATH = path
                if (path_to_img):
                        self.PATH_TO_IMG = path_to_img

                self.mt_color = [[[255, 255, 255]]] # begin with this for
                ↪ convenience
                self.mt_height = [[0]]

                self.HEIGHT, self.WIDTH = None, None
                # top left and low right corners
                # we don't care about z
                self.tl_corner = [0, 0] # xl, yu | indexes!!
                self.lr_corner = [0, 0] # xr, yl

                self.objects = []

        def build_mt(self, matrix):
                def do_expend(number, axis):
                        result = [0, 0] # low not ok, high not ok
                        if (number < self.tl_corner[axis]):
                                result[0] = 1
                        elif (number > self.lr_corner[axis]):
                                result[1] = 1
```

```
460                         return result
461                 def add_colomns(number_colomns, where):
462                         if (where):
463                                 # add after
464                                 for i in range(len(self.mt_color)):
465                                         self.mt_color[i].extend([[0, 0, 0]] *
                                            ↪  number_colomns)
466                                         self.mt_height[i].extend([0] *
                                            ↪  number_colomns)
467                         else:
468                                 # add before
469                                 for _ in range(number_colomns):
470                                         for i in range(len(self.mt_color)):
471                                                 self.mt_color[i].insert(0, [0,
                                                    ↪  0, 0])
472                                                 self.mt_height[i].insert(0, 0)
473                 def add_lines(number_lines, where):
474                         l = len(self.mt_color[0])
475                         if (where):
476                                 # add after
477                                 for _ in range(number_lines):
478                                         self.mt_color.append([[0, 0, 0]] * l)
479                                         self.mt_height.append([0] * l)
480                         else:
481                                 # add before
482                                 for _ in range(number_lines):
483                                         self.mt_color.insert(0, [[0, 0, 0]] *
                                            ↪  l)
484                                         self.mt_height.insert(0, [0] * l)
485
486                 for line in matrix:
487                         x, y, z, *color = map(int, line)
488                         # y = -y # fliping over the mtx to do it look like a
                            ↪  real picture
489                         x = -x
490
491                         expand_x = do_expend(x, 0)
492                         if (expand_x[0]):
493                                 # we have point with coord tl_corner[0] and we
                                    ↪  need to have point with coord x
494                                 add_colomns(self.tl_corner[0] - x, 0)
495                                 self.tl_corner[0] = x
496                         elif (expand_x[1]):
497                                 add_colomns(x - self.lr_corner[0], 1)
498                                 self.lr_corner[0] = x
499
500                         expand_y = do_expend(y, 1)
501                         if (expand_y[0]):
502                                 add_lines(self.tl_corner[1] - y, 0)
503                                 self.tl_corner[1] = y
```

```
504                         elif (expand_y[1]):
505                                 add_lines(y - self.lr_corner[1], 1)
506                                 self.lr_corner[1] = y
507 
508                         self.mt_color[y - self.tl_corner[1]][x -
                            ↪  self.tl_corner[0]] = color
509                         self.mt_height[y - self.tl_corner[1]][x -
                            ↪  self.tl_corner[0]] = abs(z)
510                 self.HEIGHT, self.WIDTH = len(self.mt_color),
                    ↪  len(self.mt_color[0])
511 
512         def improve_mt(self, matrix):
513                 for line in matrix:
514                         x, y, z, *color = map(int, line)
515                         # y = -y # fliping over the mtx to do it look like a
                            ↪  real picture
516                         x = -x
517 
518                         mt_y = y - self.tl_corner[1]
519                         mt_x = x - self.tl_corner[0]
520                         if ((0 <= mt_y < self.HEIGHT) and (0 <= mt_x <
                            ↪  self.WIDTH) and self.mt_height[mt_y][mt_x] == 0):
521                                 self.mt_color[mt_y][mt_x] = color
522                                 self.mt_height[mt_y][mt_x] = abs(z)
523 
524         def read_img(self):
525                 self.mt_color = cv2.cvtColor(cv2.imread(self.PATH_TO_IMG),
                    ↪  cv2.COLOR_BGR2RGB)
526                 self.HEIGHT, self.WIDTH = len(self.mt_color),
                    ↪  len(self.mt_color[0])
527 
528         def trinarise_mtx(self, standart=True):
529                 """mtx to red, blue and black colors only.
530                 """
531                 def color(p, standart):
532                         if (not standart and p == [0, 0, 0]):
533                                 return [255, 255, 255]
534                         return [255 * self.is_red(p), 0, 255 *
                            ↪  self.is_blue(p)]
535                 self.mt_color = [[color(p, standart) for p in line] for line
                    ↪  in self.mt_color]
536 
537         def get_area(self, pixel, img=None):
538                 """
539                 - pixel = (y, x);
540                 - col = [r, g, b], (optional, by default, is determined
    ↪  relying on "pixel");
541                 """
542                 if (img is None):
543                         img = self.mt_color
```

```
544                 col = img[pixel[0]][pixel[1]]
545
546                 visited = set([pixel])
547                 queue = set([pixel])
548                 while (queue):
549                         next_pnts = self.get_next_pnts(queue.pop(),
                            ↪  short_set=False)
550                         for next_pnt in next_pnts:
551                                 if (next_pnt not in visited and
                                    ↪  img[next_pnt[0]][next_pnt[1]] == col):
552                                         visited.add(next_pnt)
553                                         queue.add(next_pnt)
554                 return visited
555
556         def clean_img(self, min_area=370):
557                 """Removes small areas from the image.
558                 """
559                 visited_total = set() # {(y, x), (y, x), ..}
560
561                 for y in range(self.HEIGHT):
562                         for x in range(self.WIDTH):
563
564                                 if (self.mt_color[y][x] != [0, 0, 0] and ((y,
                                    ↪  x) not in visited_total)):
565                                         cur_block_area = self.get_area((y, x))
566                                         visited_total = visited_total |
                                            ↪  cur_block_area
567
568                                         if (len(cur_block_area) <= min_area):
569                                                 for (y, x) in cur_block_area:
570                                                         self.mt_color[y][x] =
                                                            ↪  [0, 0, 0]
571
572         def get_objects(self):
573                 main_set = set() # {(y, x), (y, x), ..}
574                 areas = [] # [Object, Object, ...]
575
576                 for y in range(self.HEIGHT):
577                         for x in range(self.WIDTH):
578                                 if (self.mt_color[y][x] != [0, 0, 0] and ((y,
                                    ↪  x) not in main_set)):
579                                         visited, cur_block_area =
                                            ↪  self.get_area((y, x),
                                            ↪  self.mt_color[y][x])
580                                         main_set = main_set | visited
581                                         if len(cur_block_area) < 400:
582                                                 for (y, x) in visited:
583                                                         self.mt_color[y][x] =
                                                            ↪  [0, 0, 0]
584                                         else:
```

```
585                                                 h =
                                                    ↪  [self.mt_height[p[0]][p[1]]
                                                    ↪  for p in cur_block_area]
586                                                 img = [[[0, 0, 0] for _ in
                                                    ↪  line] for line in
                                                    ↪  self.mt_color]
587                                                 for p in cur_block_area:
588                                                         img[p[0]][p[1]] =
                                                            ↪  [255, 0, 0]
589                                                 areas.append(Object(cur_block_area,
                                                    ↪  h, img))
590             self.objects = areas
591             return areas
592
593     def get_corners(self, sorted_contour):
594             reds_list = []
595             for y, x in sorted_contour:
596                     area = self.create_area(y, x)
597                     reds = 0
598                     for y1, x1 in area:
599                             if self.mt_color[y1][x1][0] == 255:
600                                     reds += 1
601                     reds_list.append([reds, (y, x), set(area)])
602             reds_list.sort(key=lambda x: x[0])
603
604             remembered_points = set()
605             corners = []
606             i = 0
607             while len(corners) < 4:
608                     y, x = reds_list[i][1]
609                     while (y, x) in remembered_points:
610                             i += 1
611                             y, x = reds_list[i][1]
612
613                     corners.append([y, x])
614                     remembered_points = remembered_points |
                        ↪  reds_list[i][2]
615                     i += 1
616             return corners
617
618     def visualize(self, ignore_colors=[]):
619             import open3d as o3d
620             x, y, z = [], [], []
621             cr, cg, cb = [], [], []
622             for i in range(self.HEIGHT):
623                     for j in range(self.WIDTH):
624
625                             if(self.mt_height[i][j] == 0): continue
626                             if(self.mt_color[i][j] in ignore_colors):
                                ↪  continue
```

```
627
628                                 x.append(j)
629                                 y.append(i)
630                                 z.append(self.mt_height[i][j])
631
632                                 cr.append(self.mt_color[i][j][0])
633                                 cg.append(self.mt_color[i][j][1])
634                                 cb.append(self.mt_color[i][j][2])
635
636             points = np.vstack((x, y, z)).transpose()
637             colors = np.vstack((cr, cg, cb)).transpose()
638
639             pcd = o3d.geometry.PointCloud()
640             pcd.points = o3d.utility.Vector3dVector(points)
641             pcd.colors = o3d.utility.Vector3dVector(colors/255)
642             o3d.visualization.draw_geometries([pcd])
643
644         def visualize_points(self, points, ignore_colors=[]):
645             import open3d as o3d
646             x, y, z = [], [], []
647             cr, cg, cb = [], [], []
648             for i ,j in points:
649                 if(self.mt_height[i][j] == 0): continue
650                 if(self.mt_color[i][j] in ignore_colors): continue
651
652                 x.append(j)
653                 y.append(i)
654                 z.append(self.mt_height[i][j])
655
656                 cr.append(self.mt_color[i][j][0])
657                 cg.append(self.mt_color[i][j][1])
658                 cb.append(self.mt_color[i][j][2])
659
660             points = np.vstack((x, y, z)).transpose()
661             colors = np.vstack((cr, cg, cb)).transpose()
662
663             pcd = o3d.geometry.PointCloud()
664             pcd.points = o3d.utility.Vector3dVector(points)
665             pcd.colors = o3d.utility.Vector3dVector(colors/255)
666             o3d.visualization.draw_geometries([pcd])
667
668         def determine_blocks(self, image):
669             hgts = [[[0, 255-self.mt_height[y][x], 0] if
                     ↪  self.mt_color[y][x]!=[0,0,0] else [0,0,0] for x in
                     ↪  range(self.WIDTH)] for y in range(self.HEIGHT)]
670             src=np.array(hgts, dtype=np.uint8)
671             # src = np.array(image, dtype=np.uint8)
672             # src = cv2.cvtColor(src, cv2.COLOR_RGB2BGR) # optional
673
674             gray = cv2.cvtColor(src, cv2.COLOR_RGB2GRAY)
```

```
675
676                 bw = cv2.threshold(gray, 10, 255, cv2.THRESH_BINARY)[1]
677
678                 dist = cv2.distanceTransform(bw, cv2.DIST_L2, 3)
679                 cv2.normalize(dist, dist, 0, 1.0, cv2.NORM_MINMAX)
680
681                 _, dist = cv2.threshold(dist, 0.05, 1.0, cv2.THRESH_BINARY)
682
683                 kernel = np.ones((3, 3), dtype=np.uint8)
684                 dist = cv2.dilate(dist, kernel)
685
686                 dist_8u = dist.astype('uint8')
687                 # Find total markers
688                 contours, _ = cv2.findContours(dist_8u, cv2.RETR_EXTERNAL,
                    ↪  cv2.CHAIN_APPROX_SIMPLE)
689                 # Create the marker image for the watershed algorithm
690                 markers = np.zeros(dist.shape, dtype=np.int32)
691                 for i in range(len(contours)):
692                         cv2.drawContours(markers, contours, i, (i+1), -1)
693
694                 cv2.circle(markers, (5, 5), 3, (255, 255, 255), -1)
695
696                 cv2.watershed(src, markers)
697
698                 # result image
699                 blocks = {}
700                 for i in range(markers.shape[0]):
701                         for j in range(markers.shape[1]):
702                                 cur_index = markers[i, j]
703                                 if (0 < cur_index <= len(contours) and
                                    ↪  image[i][j] != [0, 0, 0]):
704                                         if (cur_index not in blocks):
705                                                 blocks[cur_index] = [(i, j)]
706                                         else:
707                                                 blocks[cur_index].append((i,
                                                    ↪  j))
708                 return blocks
709
710         def create_objects(self):
711                 obj = []
712                 blocks = self.determine_blocks(self.mt_color)
713                 for block in blocks.values():
714                         if len(block) < 370 : continue
715                         h = [self.mt_height[p[0]][p[1]] for p in block]
716                         img = [[[0, 0, 0] for _ in line] for line in
                            ↪  self.mt_color]
717                         color = self.most_common_color(block)
718                         for p in block:
719                                 img[p[0]][p[1]] = color
720                         obj.append(Object(block, h, img, color))
```

```
721                 return obj
722
723         def find_max_min_blocks(self, blocks):
724                 # Returns minobj, maxobj sorted by `y` coordinate
725                 miny, maxy = int(10e10), int(-10e10)
726                 minobj, maxobj = None, None
727                 for block in blocks:
728                         for p in block.area:
729                                 if p[0] < miny:
730                                         miny = p[0]
731                                         minobj = block
732                                 if p[0] > maxy:
733                                         maxy = p[0]
734                                         maxobj = block
735                 return minobj, maxobj
736
737         def get_rot_angle(self, obj):
738                 img = np.array(obj.source_img, dtype=np.uint8)
739                 img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
740                 contours = cv2.findContours(img, cv2.RETR_TREE,
                    ↪  cv2.CHAIN_APPROX_SIMPLE)[0]
741
742                 cnt = sorted(contours, key=cv2.contourArea, reverse=True)[0]
743                 rect = cv2.minAreaRect(cnt)
744                 box = cv2.boxPoints(rect)
745                 box = np.int0(box)
746
747                 center = np.round(sum(box) / 4)
748                 cx, cy = map(int, center)
749
750                 yx_box = [[y, x] for x, y in list(box)]
751                 sorted_corners = self.sort_clockwise(yx_box, [cy, cx])
752
753                 y_low_coord = max(sorted_corners, key=lambda x: x[0])
754
755                 ind = sorted_corners.index(y_low_coord) # find lowest point
756                 sorted_corners = sorted_corners[ind:] + sorted_corners[:ind]
757
758                 a = self.dist(sorted_corners[0], sorted_corners[1])
759                 b = self.dist(sorted_corners[0], sorted_corners[3])
760                 if (a > b):
761                         # tilted to the left
762                         angle = degrees(acos((sorted_corners[1][1] -
                            ↪  y_low_coord[1]) / a))
763                 else:
764                         # tilted to the right
765                         angle = degrees(acos((sorted_corners[3][1] -
                            ↪  y_low_coord[1]) / b))
766                 return angle
```

Файл src/main.py

```
from itertools import count
from dots3d import MatrixProcessing, Object
import numpy as np
import open3d as o3d
import cv2
import random as rng
import time
import sys
import time
from math import pi
from FrameProcessing import FrameProcessing
from OperateCamera import OperateCamera
from OperateRobot import OperateRobot
from rotation_vector_map import rotation_vector_map
from Movement import Movement

def way_to_remove_top_blocks(blocks):
    move_poses = []
    for i, obj in enumerate(blocks):
        move_poses.append((obj.globaly + 0.07, obj.globalx))
        move_poses.append((obj.globaly - 0.07, obj.globalx))
        move_poses.append((obj.globaly, obj.globalx + 0.07))
        move_poses.append((obj.globaly, obj.globalx - 0.07))
        #print('object'+str(i),obj.color, obj.globaly, obj.globalx,
        ↪  obj.globalz, obj.angle, len(obj.area), sep='    ')
    return move_poses


#
↪  ================================================================================
def main():

    ↪  #pathnpy=r'C:\Users\mreve\Desktop\final_solution\cols_640x480_test0h0.65.npy'
    #pathply=r'C:\Users\mreve\Desktop\final_solution\640x480_test0h0.65.ply'
    movement = Movement()

    movement.move_to_pcd()
    movement.rotate_tool_oz(np.radians(2.5))

    fp = FrameProcessing()
    fp.frameprocessing()

    move_poses = way_to_remove_top_blocks(fp.objects)

    movement.close_gripper()
    for pos in move_poses:
        movement.move_to_position({'x': pos[1], 'y': pos[0], 'z': 0.38, 'ang':
        ↪  0})
```

```
45
46
47
48      while True:
49
50          movement.move_to_pcd()
51          movement.rotate_tool_oz(np.radians(2.5))
52
53          fp = FrameProcessing()
54          fp.frameprocessing()
55
56          obj = fp.objects[0]
57
58          angle = ((90 - np.degrees(obj.angle)) // 5) * 5
59
60          movement.grip_block({'x' : obj.globalx, 'y': obj.globaly, 'z':
            ↪  obj.globalz, 'ang': angle})
61          if obj.color == [255, 0, 0]:
62              movement.to_red_box()
63          else:
64              movement.to_blue_box()
65          #print(obj.color, obj.globaly, obj.globalx, obj.globalz, angle,
            ↪  len(obj.area), sep='    ')
66
67
68
69
70  if __name__ == "__main__":
71      main()
```

Файл src/visualise.py

```
1   from itertools import count
2   from dots3d import MatrixProcessing, Object
3   import numpy as np
4   import open3d as o3d
5   import cv2
6   import random as rng
7   import time
8   import sys
9   import time
10  from math import pi
11  from FrameProcessing import FrameProcessing
12  from OperateCamera import OperateCamera
13  from OperateRobot import OperateRobot
14  from rotation_vector_map import rotation_vector_map
15  from Movement import Movement
16
17
```

```
#
    ↪ ==========================================================================================
def main():

    ↪ #pathnpy=r'C:\Users\mreve\Desktop\final_solution\cols_640x480_test0h0.65.npy'
    #pathply=r'C:\Users\mreve\Desktop\final_solution\640x480_test0h0.65.ply'
    movement = Movement()

    while True:

        movement.move_to_pcd()
        movement.rotate_tool_oz(np.radians(2.5))

        fp = FrameProcessing()
        fp.frameprocessing()

        obj = fp.objects[0]

        angle = ((90 - np.degrees(obj.angle)) // 5) * 5

        movement.grip_block({'x' : obj.globalx, 'y': obj.globaly, 'z':
        ↪ obj.globalz, 'ang': angle})
        if obj.color == [255, 0, 0]:
            movement.to_red_box()
        else:
            movement.to_blue_box()
        print(obj.color, obj.globaly, obj.globalx, obj.globalz, angle,
        ↪ len(obj.area), sep='    ')




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

Файл src/sim.py

```
import OperateCamera
import OperateRobot
import open3d as o3d
import cv2
import numpy as np
import time
import random as rng
from dots3d import MatrixProcessing, Object

def build_mtx(arr_cols, arr_dots):
        mtx_hgts=np.zeros((720, 1280,3), dtype=np.int32)
        mtx_cols=np.zeros((720, 1280,3), dtype=np.uint8)

```

```
14
15          for i in range(len(arr_dots)):
16
17                  x = arr_dots[i][0]
18                  y = arr_dots[i][1]
19                  z = np.abs(arr_dots[i][2])
20                  mtx_hgts[360-y][640+x] = [x,y,z]
21                  mtx_cols[360-y][640+x]=arr_cols[i]
22
23
24          sumx=np.sum(mtx_hgts, axis=0)
25          sumy=np.sum(mtx_hgts, axis=1)
26          up, down, left, right = -1,-1,-1,-1
27          for i in range(640):
28                  if up==-1 and sumy[i][2]>0:
29                          up=i
30                  if down==-1 and sumy[-i][2]>0:
31                          down=720-i
32                  if left==-1 and sumx[i][2]>0:
33                          left=i
34                  if right==-1 and sumx[-i][2]>0:
35                          right=1280-i
36          #print(up, down, left, right)
37          _, mtx_cols, _ =np.split(mtx_cols,[up, down],axis=0)
38          _, mtx_cols, _ =np.split(mtx_cols,[left,right],axis=1)
39          _, mtx_hgts, _ =np.split(mtx_hgts,[up, down],axis=0)
40          _, mtx_hgts, _ =np.split(mtx_hgts,[left,right],axis=1)
41
42
43          mtx_cols=cv2.cvtColor(mtx_cols, cv2.COLOR_RGB2BGR)
44          #cv2.imshow('c',mtx_cols)
45          #cv2.imwrite('mtx_cols.png', mtx_cols)
46          #cv2.imshow('p',np.split(mtx_hgts,3,axis=2)[2])
47
48          return mtx_cols, mtx_hgts
49
50  #======================красный синий черный
51  def trinarize(mtx_cols):
52          # удаление белого фона
53          lower = np.array([0, 110, 0])
54          upper = np.array([255, 255, 255])
55          hsv = cv2.cvtColor(mtx_cols, cv2.COLOR_RGB2HSV)
56          mask = cv2.inRange(hsv, lower, upper)
57          output = cv2.bitwise_and(mtx_cols, mtx_cols, mask=mask)
58          # cv2.imshow('iiii',output)
59          #cv2.imwrite('outtt.png', output)
60          # тринаризация изображения
61          hsv = cv2.cvtColor(output, cv2.COLOR_RGB2HSV)
62          bd = np.array([40, 0, 0])
63          bu = np.array([179, 255, 255])
```

```
64          rd = np.array([0, 0, 0])
65          ru = np.array([70, 255, 255])
66          for y in range(output.shape[0]):
67                  for x in range(output.shape[1]):
68                          # print(output[y][x])
69                          if (hsv[y][x] > bd).all() and (hsv[y][x] < bu).all():
70                                  output[y][x] = [0, 0, 255]
71                          elif (hsv[y][x] > rd).all() and (hsv[y][x] <
                            ↪  ru).all():
72                                  output[y][x] = [255, 0, 0]
73                          else:
74                                  output[y][x] = [0, 0, 0]
75          mtx_cols = output
76          #cv2.imshow('i',mtx_cols)
77
78          return mtx_cols
79  #======================красный синий черный
80
81  #======================выделение блоков по высоте - не используется
82  def determine_blocks_by_heght(image, hights):
83          _, _, hights = np.split(hights,3,axis=2)
84
85          minh=min([i for i in hights.flatten() if i!=0])
86          maxh=max([i for i in hights.flatten()])
87          #print(minh, maxh)
88          #hights[y][x]-minh<10 нижний уровень
89          #print(image.shape)
90          hgts = np.zeros(image.shape, dtype=np.uint8)
91          for y in range(image.shape[0]):
92                  for x in range(image.shape[1]):
93                          if not (image[y][x] == [0, 0, 0]).all() and
                            ↪  hights[y][x]-minh<5:
94                                  hgts[y][x][1]=hights[y][x]
95
96          if maxh-minh<22:
97                  return {}
98          else:
99                  blocks = determine_blocks(hgts)
100                 return blocks
101 #======================выделение блоков по высоте - не используется
102
103
104 def dilatation(src):
105         dilatation_size = 2
106         dilation_shape = cv2.MORPH_ELLIPSE#cv2.MORPH_RECT#
107         element = cv2.getStructuringElement(dilation_shape, (2 *
            ↪  dilatation_size + 1, 2 * dilatation_size + 1),
108
                                                                    ↪  (dilatati
                                                                    ↪  dilatatio
```

```
109         dilatation_dst = cv2.dilate(src, element)
110         return dilatation_dst
111 
112 
113 #====================выделение блоков по цвету
114 def determine_blocks(image):
115         image=dilatation(image)
116         #cv2.imwrite('out.png', image)
117         #cv2.imshow('original ', image)
118         kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]],
        ↪  dtype=np.float32)
119         imgLaplacian = cv2.filter2D(image, cv2.CV_32F, kernel)
120         sharp = np.float32(image)
121         imgResult = sharp - imgLaplacian
122         # convert back to 8bits gray scale
123         imgResult = np.clip(imgResult, 0, 255)
124         imgResult = imgResult.astype('uint8')
125         imgLaplacian = np.clip(imgLaplacian, 0, 255)
126         imgLaplacian = np.uint8(imgLaplacian)
127 
128         bw = cv2.cvtColor(imgResult, cv2.COLOR_BGR2GRAY)
129         #_, bw = cv.threshold(bw, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)
130         #cv2.imshow('Binary Image2', bw)
131         dist = cv2.distanceTransform(bw, cv2.DIST_L2, 3)
132         # Normalize the distance image for range = {0.0, 1.0}
133         # so we can visualize and threshold it
134         cv2.normalize(dist, dist, 0, 1.0, cv2.NORM_MINMAX)
135         #cv2.imshow('Distance Transform Image', dist)
136         _, dist = cv2.threshold(dist, 0.4, 1.0, cv2.THRESH_BINARY)
137         # Dilate a bit the dist image
138         kernel1 = np.ones((3,3), dtype=np.uint8)
139         dist = cv2.dilate(dist, kernel1)
140         #cv2.imshow('Peaks', dist)
141         dist_8u = dist.astype('uint8')
142         # Find total markers
143         contours, _ = cv2.findContours(dist_8u, cv2.RETR_EXTERNAL,
        ↪  cv2.CHAIN_APPROX_SIMPLE)
144         # Create the marker image for the watershed algorithm
145         markers = np.zeros(dist.shape, dtype=np.int32)
146         # Draw the foreground markers
147         for i in range(len(contours)):
148                 cv2.drawContours(markers, contours, i, (i+1), -1)
149         # Draw the background marker
150         cv2.circle(markers, (5,5), 3, (255,255,255), -1)
151         markers_8u = (markers * 10).astype('uint8')
152         #cv2.imshow('Markers', markers_8u)
153         cv2.watershed(imgResult, markers)
154 
155         # result image
156         colors = []
```

```
157         for contour in contours:
158                 colors.append((rng.randint(0,256), rng.randint(0,256),
                    ↪  rng.randint(0,256)))
159         # Create the result image
160         dst = np.zeros((markers.shape[0], markers.shape[1], 3),
            ↪  dtype=np.uint8)
161         # Fill labeled objects with random colors
162         for i in range(markers.shape[0]):
163                 for j in range(markers.shape[1]):
164                         index = markers[i,j]
165                         if index > 0 and index <= len(contours):
166                                 dst[i,j,:] = colors[index-1]
167         #cv2.imshow('dst', dst)
168         # ================================================
169         # in contours we have approximate contours of the blocks
170         #test_img = np.array([[list(i) for i in line] for line in src])
171         #for i in range(len(contours)):
172         #    cv2.drawContours(test_img, contours, i, (0, 255, 0), 1)
173         #show_img(test_img)
174         # ================================================
175         #print(len(contours))
176         blocks = {}
177         for i in range(markers.shape[0]):
178                 for j in range(markers.shape[1]):
179                         cur_index = markers[i, j]
180                         if 0 < cur_index <= len(contours):
181 
182                                 #print(cur_index)
183                                 if (cur_index not in blocks):
184                                         blocks[cur_index] = [(i, j)]
185                                 else:
186                                         blocks[cur_index].append((i, j))
187         return blocks
188 #======================выделение блоков по цвету
189 
190 def most_common_color( area, mtx_cols):
191         red = 0
192         blue = 0
193         for (y, x) in area:
194                 color = mtx_cols[y][x]
195                 if (color == [0, 0, 255]).all():
196                         red += 1
197                 elif (color == [255, 0, 0]).all():
198                         blue += 1
199         if (red > blue):
200                 return [255, 0, 0]
201         else:
202                 return [0, 0, 255]
203 
204 
```

```
205 def main():
206         np.set_printoptions(threshold=np.inf)
207 
208         #pcd =
            ↪  o3d.io.read_point_cloud(r"D:\Vadim\Documents\projects\NTI\2021-2022\IRS\STEP
209         pcd = o3d.io.read_point_cloud(r"input.ply")
210         arr_cols =(np.array(pcd.colors) * 255).astype(np.uint8)
211         arr_dots = (np.array(pcd.points) * 500).astype(np.int32)
212 
213         mtx_cols, mtx_hgts = build_mtx(arr_cols, arr_dots)
214         #print(np.split(mtx_hgts,3,axis=2)[2])
215         mtx_cols=trinarize(mtx_cols)
216         #blocks = determine_blocks(mtx_cols)
217         blocks = determine_blocks_by_heght(mtx_cols, mtx_hgts)
218 
219         #print(blocks.keys())
220 
221         objs = []
222         for block in blocks.values():
223                 if len(block) < 180:
224                         continue
225                 h = [mtx_hgts[p[0]][p[1]][2] for p in block]
226                 y = [mtx_hgts[p[0]][p[1]][0] for p in block]
227                 x = [mtx_hgts[p[0]][p[1]][1] for p in block]
228                 img = [[[0, 0, 0] for _ in line] for line in mtx_cols]
229                 color = most_common_color(block, mtx_cols)
230                 for p in block:
231                         img[p[0]][p[1]] = color
232 
233                 objs.append(Object(block, h, img, color))
234 
235         #for i, obj in enumerate(objs):
236         #        print('object'+str(i),obj.color, obj.centerz, len(obj.area),
            ↪  sep='    ')
237 
238         #cv2.waitKey()
239         print(len(objs))
240         pass
241 
242 
243 if __name__ == "__main__":
244         main()
```

Файл src/test-red.py

```
1 from itertools import count
2 from dots3d import MatrixProcessing, Object
3 import numpy as np
4 import open3d as o3d
5 import cv2
```

```python
import random as rng
import time
import sys
import time
from math import pi
from FrameProcessing import FrameProcessing
from OperateCamera import OperateCamera
from OperateRobot import OperateRobot
from rotation_vector_map import rotation_vector_map
from Movement import Movement


def get_last_put(area):
    if (1 not in area):
        last_put_place = -1
    else:
        last_put_place = area.index(1)
    return last_put_place

# ================================================================================
def main():

    red_area = [0, 0, 0, 0]
    take_uncomfortable = False
    movement = Movement()

    while (True):
        movement.move_to_pcd()
        movement.rotate_tool_oz(np.radians(2.5))
        fp = FrameProcessing()
        fp.frameprocessing()
        objects = fp.objects

        objects_queue = list(objects)
        queue_len = len(objects_queue)

        if (queue_len == 0):
            print("There is no blocks in the field.")
            break

        for _ in range(queue_len):
            obj = objects_queue.pop(0)

            if (obj.color == [255, 0, 0] and len(obj.area) < 550):
                is_available, angle, side = obj.get_angle(fp.mtx_cols)
                print("You should better touch edges:", side)
                # side == 1 -- с длинной стороны, 2 -- за короткую
                if (is_available or take_uncomfortable):
                    # take and put to the destination area
```

```
                    last_put_place = get_last_put(red_area)

                    if (last_put_place in [-1, 0, 1, 2]):
                        ang = (np.degrees(angle) // 5) * 5
                        movement.grip_block({'x' : obj.globalx, 'y':
                         ↪  obj.globaly, 'z': obj.globalz, 'ang': ang})

                        if last_put_place + 2 == 1:
                            movement.to_first_red_box()
                        elif last_put_place + 2 == 2:
                            movement.to_second_red_box()
                        elif last_put_place + 2 == 3:
                            movement.to_therd_red_box()
                        else:
                            movement.to_forth_red_box()

                        red_area[last_put_place + 1] = 1

                        break
                    else: # last_put_place == 3
                        print("I found 5th red block!")
                        pass
                else:
                    objects_queue.append(obj)
            else:
                take_uncomfortable = True
    print("Finished.")

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

Файл src/test-test.py

```
from itertools import count
from dots3d import MatrixProcessing, Object
import numpy as np
import open3d as o3d
import cv2
import random as rng
import time
import sys
import time
from math import pi
from FrameProcessing import FrameProcessing
from OperateCamera import OperateCamera
from OperateRobot import OperateRobot
from rotation_vector_map import rotation_vector_map
from Movement import Movement

```

```
#
↪ ==========================================================================================
def main():

    ↪ #pathnpy=r'C:\Users\mreve\Desktop\final_solution\cols_640x480_test0h0.65.npy'
    #pathply=r'C:\Users\mreve\Desktop\final_solution\640x480_test0h0.65.ply'
    movement = Movement()
    movement.move_to_pcd()
    movement.rotate_tool_oz(np.radians(2.5))

    fp = FrameProcessing()
    start=time.time()
    fp.frameprocessing()
    print('time: ', time.time()-start)


    for i, obj in enumerate(fp.objects):

        angle = ((90 - np.degrees(obj.angle)) // 5) * 5
        movement.move_to_pcd()  # Move to pcd position
        #_ = movement.get_pcd()  # Getting pcd
        #movement.set_perpenicular_on_pcd_position()  # Set perpenicular on
        ↪ pcd position

        movement.grip_block({'x' : obj.globalx, 'y': obj.globaly, 'z':
        ↪ obj.globalz, 'ang': angle})
        if obj.color == [255, 0, 0]:
            movement.to_red_box()
        else:
            movement.to_blue_box()
        print('object'+str(i),obj.color, obj.globaly, obj.globalx,
        ↪ obj.globalz, angle, len(obj.area), sep='    ')



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

Файл src/test-moves.py

```
from shutil import move
from dots3d import MatrixProcessing, Object, show_img, Object
import numpy as np
import open3d as o3d
import time
from math import pi
from OperateCamera import OperateCamera
from OperateRobot import OperateRobot
from rotation_vector_map import rotation_vector_map
```

```python


class Movement():
    def __init__(self):
        self.ip = "172.31.1.25"
        self.rob = OperateRobot(self.ip)
        self.cam = OperateCamera(resolution=1)

    def get_pcd(self):
        self.rob.movel(self.cord_for_pcd)
        return self.cam.catch_frame_without_pcd()

    def move_to_pcd(self):
        self.rob.movel({"x": -0.80, "y": -0.20, "z": 0.65,
                        "rx": 1.487, "ry": 3.536, "rz": -0.669})

    def set_perpenicular_on_pcd_position(self):
        self.rob.movel({"x": -0.80, "y": -0.20, "z": 0.80,
                        "rx": 1.217, "ry": 2.882, "rz": -0.013})

    def grip_block(self, pos):
        rx, ry, rz = rotation_vector_map[pos["ang"]]
        to_block = [{"x": pos['x'], "y": pos['y'], "z": pos['z'] + 0.1,  "rx":
        ↪  rx, "ry": ry, "rz": rz},
                    {"x": pos['x'], "y": pos['y'], "z": pos['z'],        "rx":
                    ↪  rx, "ry": ry, "rz": rz}]

        self.rob.open_gripper()
        self.rob.movel(to_block[0])
        time.sleep(0.2)
        self.rob.movel(to_block[1])
        time.sleep(0.2)
        self.rob.close_gripper()

    def move_to_position(self, pos):
        rx, ry, rz = rotation_vector_map[pos["ang"]]
        self.rob.movel({"x": pos['x'], "y": pos['y'], "z": pos['z'], "rx": rx,
        ↪  "ry": ry, "rz": rz})

    def to_red_box(self):
        to_red_box = {"x": -0.76, "y": 0.25, "z": 0.55,
                      "rx": 1.487, "ry": 3.536-pi/4, "rz": 0}
        self.rob.movel(to_red_box)
        time.sleep(0.2)
        self.rob.open_gripper()  # открыть

    def to_blue_box(self):
        to_blue_box = {"x": -0.94, "y": 0.25, "z": 0.55,
                       "rx": 1.487, "ry": 3.536-pi/4, "rz": 0}
        self.rob.movel(to_blue_box)
```

```
57          time.sleep(0.2)
58          self.rob.open_gripper()  # открыть
59
60
61  def test():
62      movement = Movement()
63
64      blocks = [{"x": -0.7276, "y": -0.1069, "z": 0.3807, "ang": 45},
65                {"x": -0.9092, "y": -0.0915, "z": 0.3807, "ang": -45}]
66
67      for i in range(2):
68          movement.move_to_pcd()  # Move to pcd position
69          time.sleep(3)  # Getting pcd
70          movement.set_perpenicular_on_pcd_position()  # Set perpenicular on pcd
            ↪  position
71
72          movement.grip_block(blocks[i])
73          if i == 0:
74              movement.to_red_box()
75          else:
76              movement.to_blue_box()
77
78
79  test()
```

Файл src/remove-top-blocks.py

```
1   import cv2
2   import time
3   from math import pi
4   import time
5   from FrameProcessing import FrameProcessing
6   from rotation_vector_map import rotation_vector_map
7   from Movement import Movement
8   import numpy as np
9
10
11  def way_to_remove_top_blocks(blocks):
12      move_poses = []
13      for i, obj in enumerate(blocks):
14          move_poses.append((obj.globaly + 0.07, obj.globalx + 0.01))
15          move_poses.append((obj.globaly - 0.07, obj.globalx - 0.01))
16          print('object'+str(i),obj.color, obj.globaly, obj.globalx,
            ↪  obj.globalz, obj.angle, len(obj.area), sep='    ')
17      return move_poses
18
19  #
    ↪  ========================================================================
20  def main():
21
```

```
    movement = Movement()
    movement.move_to_pcd()
    movement.rotate_tool_oz(np.radians(2.5))

    start=time.time()
    fp = FrameProcessing()
    fp.frameprocessing()
    print('time: ', time.time()-start)

    move_poses = way_to_remove_top_blocks(fp.objects)

    movement.close_gripper()
    for pos in move_poses:
        movement.move_to_position({'x': pos[1], 'y': pos[0], 'z': 0.38, 'ang':
        ↪  0})


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

Файл src/FrameProcessing.py

```
import bdb
from itertools import count
from dots3d import MatrixProcessing, Object
from OperateCamera import OperateCamera
import numpy as np
import open3d as o3d
import cv2
import random as rng
import time
import sys

import time


class FrameProcessing():
        def __init__(self, cameraon=True, pathnpy=r'', pathply=r''):
                """
                        camera=True                         брать изображение с
    ↪  камеры
                        camera=False                брать изображение из
    ↪  pathnpy и pathply
                        ------------------------------------------------------------
                        arr_cols                        необработанная фотка
    ↪  прям с камеры
                        arr_dots                        список точек в СО
    ↪  камеры x,y,z
                        mtx_cols                        уменьшенная фотка
    ↪  (красный синий черный) (высота, ширина, rgb)
```

```
                        mtx_global_hgts                 матрица глобальных
    ↪ координат  (высота, ширина, yxz) в мм
                        objects                                 список из
    ↪ Object
                        dst                                             уменьшенная
    ↪ цветная фотка с сегментированными блоками
                """
                self.mtx_cols=np.zeros((1), dtype=np.uint8)
                self.mtx_global_hgts=np.zeros((1), dtype=np.int32)
                self.objects=[]
                self.dst=np.zeros((1), dtype=np.uint8)
                if cameraon:
                        self.cam = OperateCamera(resolution=1)
                        pcd, color_image, _ = self.cam.catch_frame()
                        self.cam.stop()
                        self.arr_cols = cv2.cvtColor(np.array(color_image),
                        ↪  cv2.COLOR_BGR2RGB)
                        self.arr_dots = (np.array(pcd.points) *
                        ↪  1000).astype(np.int32)        #сырые значения
                else:
                        self.arr_cols = cv2.cvtColor(np.load(pathnpy),
                        ↪  cv2.COLOR_BGR2RGB)
                        self.arr_dots =
                        ↪  (np.array(o3d.io.read_point_cloud(pathply).points)*1000).ast

        #====================матрица из облака точек [y,x,z] глобальные
        def build_mtx_hgts(self, arr_dots):
                mtx_hgts = np.zeros((720, 1280, 3), dtype=np.int32)

                for i in range(len(arr_dots)):
                        x = arr_dots[i][0]
                        y = arr_dots[i][1]
                        z = np.abs(arr_dots[i][2])
                        mtx_hgts[360-y][640+x] = [-200-x, -800+y, z]

                sumx = np.sum(mtx_hgts, axis=0)
                sumy = np.sum(mtx_hgts, axis=1)
                up, down, left, right = -1, -1, -1, -1
                for i in range(640):
                        if up == -1 and sumy[i][2] > 0:
                                up = i
                        if down == -1 and sumy[-i][2] > 0:
                                down = 720-i
                        if left == -1 and sumx[i][2] > 0:
                                left = i
                        if right == -1 and sumx[-i][2] > 0:
                                right = 1280-i

                _, mtx_hgts, _ = np.split(mtx_hgts, [up, down], axis=0)
                _, mtx_hgts, _ = np.split(mtx_hgts, [left, right], axis=1)
```

```
67
68              #cv2.imshow('p', np.split(mtx_hgts.astype(np.uint8), 3,
                ↪  axis=2)[2])
69
70              return mtx_hgts
71      #====================матрица из облака точек [y,x,z] глобальные
72
73
74      #====================красный синий черный
75      def trinarize(self, mtx_cols):
76              # удаление белого фона
77              lower = np.array([0, 110, 0])
78              upper = np.array([255, 255, 255])
79              hsv = cv2.cvtColor(mtx_cols, cv2.COLOR_RGB2HSV)
80              mask = cv2.inRange(hsv, lower, upper)
81              output = cv2.bitwise_and(mtx_cols, mtx_cols, mask=mask)
82              # cv2.imshow('iiii',output)
83              #cv2.imwrite('outtt.png', output)
84              # тринаризация изображения
85              hsv = cv2.cvtColor(output, cv2.COLOR_RGB2HSV)
86              bd = np.array([40, 0, 0])
87              bu = np.array([179, 255, 255])
88              rd = np.array([0, 0, 0])
89              ru = np.array([70, 255, 255])
90              for y in range(output.shape[0]):
91                      for x in range(output.shape[1]):
92                              # print(output[y][x])
93                              if (hsv[y][x] > bd).all() and (hsv[y][x] <
                                ↪  bu).all():
94                                      output[y][x] = [0, 0, 255]
95                              elif (hsv[y][x] > rd).all() and (hsv[y][x] <
                                ↪  ru).all():
96                                      output[y][x] = [255, 0, 0]
97                              else:
98                                      output[y][x] = [0, 0, 0]
99              mtx_cols = output
100             #cv2.imshow('i',mtx_cols)
101
102             return mtx_cols
103     #====================красный синий черный
104
105     #====================выделение блоков по высоте - не используется
106     def determine_blocks_by_heght(self, image, hights):
107             hgts = [[[0, 650-hights[y][x], 0] if not (image[y][x] == [0,
                ↪  0, 0]).all(
108             ) else [0, 0, 0] for x in range(image.shape[1])] for y in
                ↪  range(image.shape[0])]
109             hgts = np.array(hgts, dtype=np.uint8)
110
111             blocks = self.determine_blocks(hgts)
```

```
112             return blocks
113     #=====================выделение блоков по высоте - не используется
114
115     #=====================выделение блоков по цвету
116     def determine_blocks(self, image):
117             #cv2.imshow('original ', image)
118             kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]],
                ↪  dtype=np.float32)
119             imgLaplacian = cv2.filter2D(image, cv2.CV_32F, kernel)
120             sharp = np.float32(image)
121             imgResult = sharp - imgLaplacian
122             # convert back to 8bits gray scale
123             imgResult = np.clip(imgResult, 0, 255)
124             imgResult = imgResult.astype('uint8')
125             imgLaplacian = np.clip(imgLaplacian, 0, 255)
126             imgLaplacian = np.uint8(imgLaplacian)
127
128             bw = cv2.cvtColor(imgResult, cv2.COLOR_BGR2GRAY)
129
130             dist = cv2.distanceTransform(bw, cv2.DIST_L2, 3)
131             # Normalize the distance image for range = {0.0, 1.0}
132             # so we can visualize and threshold it
133             cv2.normalize(dist, dist, 0, 1.0, cv2.NORM_MINMAX)
134
135             _, dist = cv2.threshold(dist, 0.4, 1.0, cv2.THRESH_BINARY)
136             # Dilate a bit the dist image
137             kernel1 = np.ones((3, 3), dtype=np.uint8)
138             dist = cv2.dilate(dist, kernel1)
139
140             dist_8u = dist.astype('uint8')
141             # Find total markers
142             contours, _ = cv2.findContours(
143                     dist_8u, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
144             # Create the marker image for the watershed algorithm
145             markers = np.zeros(dist.shape, dtype=np.int32)
146             # Draw the foreground markers
147             for i in range(len(contours)):
148                     cv2.drawContours(markers, contours, i, (i+1), -1)
149             # Draw the background marker
150             cv2.circle(markers, (5, 5), 3, (255, 255, 255), -1)
151
152             cv2.watershed(imgResult, markers)
153
154             # result image
155             colors = []
156             for contour in contours:
157                     colors.append(
158                             (rng.randint(0, 256), rng.randint(0, 256),
                            ↪  rng.randint(0, 256)))
159             # Create the result image
```

```
160             dst = np.zeros((markers.shape[0], markers.shape[1], 3),
                ↪  dtype=np.uint8)
161             # Fill labeled objects with random colors
162             for i in range(markers.shape[0]):
163                     for j in range(markers.shape[1]):
164                             index = markers[i, j]
165                             if index > 0 and index <= len(contours):
166                                     dst[i, j, :] = colors[index-1]
167             self.dst=dst
168
169             blocks = {}
170             for i in range(markers.shape[0]):
171                     for j in range(markers.shape[1]):
172                             cur_index = markers[i, j]
173                             if 0 < cur_index <= len(contours):
174
175                                     # print(cur_index)
176                                     if (cur_index not in blocks):
177                                             blocks[cur_index] = [(i, j)]
178                                     else:
179                                             blocks[cur_index].append((i,
                                        ↪  j))
180             return blocks
181     #======================выделение блоков по цвету
182
183     def most_common_color(self, area, mtx_cols):
184             red = 0
185             blue = 0
186             for (y, x) in area:
187                     color = mtx_cols[y][x]
188                     if (color == [0, 0, 255]).all():
189                             red += 1
190                     elif (color == [255, 0, 0]).all():
191                             blue += 1
192             if (red > blue):
193                     return [255, 0, 0]
194             else:
195                     return [0, 0, 255]
196
197     def frameprocessing(self):
198             mtx_hgts_ply = self.build_mtx_hgts(self.arr_dots)
199             mtx_cols = cv2.resize(
200                     self.arr_cols, (mtx_hgts_ply.shape[1],
                        ↪  mtx_hgts_ply.shape[0]))
201
202             #======================сжатие матриц в 4 раза
203             for i in range(mtx_cols.shape[0]//2):
204                     mtx_cols = np.delete(mtx_cols, (i), axis=0)
205                     mtx_hgts_ply = np.delete(mtx_hgts_ply, (i), axis=0)
206             for i in range(mtx_cols.shape[1]//2):
```

```
207                     mtx_cols = np.delete(mtx_cols, (i), axis=1)
208                     mtx_hgts_ply = np.delete(mtx_hgts_ply, (i), axis=1)
209             #====================сжатие матриц в 4 раза
210
211             mtx_cols = self.trinarize(mtx_cols)
212
213             self.mtx_cols = mtx_cols
214             self.mtx_global_hgts =mtx_hgts_ply
215
216             blocks = self.determine_blocks(mtx_cols)
217             #blocks = determine_blocks_by_heght(mtx_cols,
                ↪  np.split(mtx_hgts_ply,3,axis=2)[2])
218             #print(blocks.keys())
219
220             obj = []
221             for block in blocks.values():
222                     if len(block) < 100:
223                             continue
224                     h = [mtx_hgts_ply[p[0]][p[1]][2] for p in block]
225                     y = [mtx_hgts_ply[p[0]][p[1]][0] for p in block]
226                     x = [mtx_hgts_ply[p[0]][p[1]][1] for p in block]
227                     img = [[[0, 0, 0] for _ in line] for line in mtx_cols]
228                     color = self.most_common_color(block, mtx_cols)
229                     for p in block:
230                             img[p[0]][p[1]] = color
231
232                     obj.append(Object(block, h, img, color))
233
234                     obj[-1].globaly = (sum(y) / len([i for i in y if i !=
                        ↪  0]))/1000
235                     obj[-1].globalx = (sum(x) / len([i for i in x if i !=
                        ↪  0]))/1000-0.035
236                     obj[-1].globalz =
                        ↪  0.345        #mtx_hgts_ply[y][x][2]/1000
237
238             self.objects=obj
239
240     def calibration(self):
241             print('create calibration')
242             dots = []
243             for i in range(4):
244                     dots.append([self.objects[i].globaly,
                        ↪  self.objects[i].globalx])
245                     dots.append([self.objects[i].globaly+0.02,
                        ↪  self.objects[i].globalx-0.02])
246                     dots.append([self.objects[i].globaly+0.03,
                        ↪  self.objects[i].globalx-0.03])
247                     dots.append([self.objects[i].globaly+0.04,
                        ↪  self.objects[i].globalx-0.04])
```

```
                        dots.append([self.objects[i].globaly+0.05,
                        ↪  self.objects[i].globalx-0.05])
                print('complete')
                return dots
```

Файл src/calibration.py

```
import cv2
import time
from math import pi
import time
from FrameProcessing import FrameProcessing
from rotation_vector_map import rotation_vector_map
from Movement import Movement


#
↪  ================================================================================
def main():

    movement = Movement()
    movement.move_to_pcd()

    start=time.time()
    fp = FrameProcessing()
    fp.frameprocessing()
    print('time: ', time.time()-start)

    move_poses = fp.calibration()
    movement.close_gripper()

    for i, pos in enumerate(move_poses):
        print(f'Moving to pose: {pos}, {i}')
        movement.move_to_position({'x': pos[1], 'y': pos[0], 'z': 0.38, 'ang':
        ↪  0})
        time.sleep(1)


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

Файл src/OperateRobot.py

```
import urx
# pip install git+https://github.com/jkur/python-urx


class OperateRobot:

```

```
    def __init__(self, ip):
        self.rob = urx.Robot(ip)

    def set_tcp(self, tcp):
        self.rob.set_tcp(tcp)

    def movel(self, point: dict):
        self.rob.movel((point["x"], point["y"], point["z"],
                        point["rx"], point["ry"], point["rz"]), 0.2, 0.2)

    def movej(self, point: dict):
        self.rob.movej((point["w1"], point["w2"], point["w3"],
                        point["w4"], point["w5"], point["w6"]), 0.2, 0.2)

    def getl(self):
        return self.rob.getl()

    def getj(self):
        return self.rob.getj()

    def close(self):
        self.rob.close()

    def open_gripper(self):
        self.rob.send_program('set_tool_digital_out(0, True)')
        self.rob.send_program('set_tool_digital_out(1, False)')

    def close_gripper(self):
        self.rob.send_program('set_tool_digital_out(0, False)')
        self.rob.send_program('set_tool_digital_out(1, True)')

    def set_tcp(self, tcp):
        self.rob.set_tcp(tcp)
```

Файл src/get-rotation-vector.py

```
import time
import numpy as np
import os
from OperateCamera import OperateCamera
from OperateRobot import OperateRobot

rob = OperateRobot("172.31.1.25")
cam = OperateCamera()

# Едем  изначальную позицию для фотографироания, под 45 градусо к столу
print('Frame capture pos')
rob.movel({"x": -0.80, "y": -0.20, "z": 0.80,
           "rx": 1.487, "ry": 3.536, "rz": -0.669})
time.sleep(5)
```

```
15
16  # Поворачиаемся перпендикулярно плоскости стола
17  print('Move to st pos...')
18  rob.movel({"x": -0.80, "y": -0.20, "z": 0.80,
19             "rx": 1.217, "ry": 2.882, "rz": -0.013})
20  time.sleep(5)
21
22  # Функия для поврота последнего шарнира
23  def rotate_tool_oz(radians):
24      w1, w2, w3, w4, w5, w6 = rob.getj()
25      print(w1, w2, w3, w4, w5, w6)
26      w6 = radians
27      rob.movej({"w1": w1, "w2": w2, "w3": w3, "w4": w4, "w5": w5, "w6": w6})
28
29
30  # Поворачиваемся на каждые 5 градусов и сохраняем значения rx, ry, rz
31  positions = []
32  _, _, _, _, _, w = rob.getj()
33  for i in range(-180, 181, 5):
34      angle_now = w + np.radians(i)
35      rotate_tool_oz(angle_now)
36      _, _, _, rx, ry, rz = rob.getl()
37      positions.append([i, rx, ry, rz])
38      print(i, rx, ry, rz)
39  path = os.path.join(os.getcwd(), 'output', 'position_vector.npy')
40  np.save(path, np.array(positions))
41
42
43  rob.close()
44  cam.stop()
```

Файл src/Movement.py

```
1   import time
2   import time
3   from math import pi
4   from OperateRobot import OperateRobot
5   from rotation_vector_map import rotation_vector_map
6
7
8   class Movement():
9       def __init__(self):
10          self.ip = "172.31.1.25"
11          self.rob = OperateRobot(self.ip)
12
13      def move_to_pcd(self):
14          self.rob.movel({"x": -0.80, "y": -0.20, "z": 0.65,
15                          "rx": 1.487, "ry": 3.536, "rz": -0.669})
16
17      def set_perpenicular_on_pcd_position(self):
```

```
18          self.rob.movel({"x": -0.80, "y": -0.20, "z": 0.80,
19                          "rx": 1.217, "ry": 2.882, "rz": -0.013})
20
21      def grip_block(self, pos):
22          rx, ry, rz = rotation_vector_map[pos["ang"]]
23          to_block = [{"x": pos['x'], "y": pos['y'], "z": pos['z'] + 0.1, "rx":
            ↪  rx, "ry": ry, "rz": rz},
24                      {"x": pos['x'], "y": pos['y'], "z": pos['z'],       "rx":
                        ↪  rx, "ry": ry, "rz": rz}]
25
26          self.rob.open_gripper()
27          self.rob.movel(to_block[0])
28          time.sleep(0.2)
29          self.rob.movel(to_block[1])
30          time.sleep(0.2)
31          self.rob.close_gripper()
32
33          # Функия для поврота последнего шарнира
34      def rotate_tool_oz(self, radians):
35          w1, w2, w3, w4, w5, w6 = self.rob.getj()
36          print(w1, w2, w3, w4, w5, w6)
37          w1 += radians
38          self.rob.movej({"w1": w1, "w2": w2, "w3": w3, "w4": w4, "w5": w5,
            ↪  "w6": w6})
39          print(self.rob.getl())
40
41      def move_to_position(self, pos):
42          rx, ry, rz = rotation_vector_map[pos["ang"]]
43          self.rob.movel({"x": pos['x'], "y": pos['y'], "z": pos['z'], "rx": rx,
            ↪  "ry": ry, "rz": rz})
44
45      def to_red_box(self):
46          to_red_box = {"x": -0.76, "y": 0.25, "z": 0.40,
47                        "rx": 1.217, "ry": 2.882, "rz": -0.013}
48          self.rob.movel(to_red_box)
49          time.sleep(0.2)
50          self.rob.open_gripper()  # открыть
51
52      def to_blue_box(self):
53          to_blue_box = {"x": -0.94, "y": 0.25, "z": 0.40,
54                         "rx": 1.217, "ry": 2.882, "rz": -0.013}
55          self.rob.movel(to_blue_box)
56          time.sleep(0.2)
57          self.rob.open_gripper()  # открыть
58
59      def close_gripper(self):
60          self.rob.close_gripper()
61
62      def open_gripper(self):
63          self.rob.open_gripper()
```

Файл src/tutorial.py

```python
import time
from math import pi

from OperateCamera import OperateCamera
from OperateRobot import OperateRobot

# Connection to the robot
rob = OperateRobot("172.31.1.25")

# Taking global linear position of arm
pos = rob.getl()
pos[0] += 0.1

moving_coordinates = {"x": pos[0], "y": pos[1], "z": pos[2], "rx": pos[3], "ry": pos[4], "rz": pos[5]}

# Moving to new coordinates. X + 10 mm
rob.movel(moving_coordinates)
time.sleep(2)

# Moving to position for taking frame from camera
rob.movel({"x": -0.82, "y": -0.1723, "z": 0.68, "rx": 1.487, "ry": 3.536, "rz": -0.669})
time.sleep(4)

# Open gripper of arm
rob.open_gripper()
time.sleep(2)

# Close gripper of arm
rob.close_gripper()
time.sleep(2)

cam = OperateCamera()

# Taking data frame from camera (RGBD format)
frame = cam.catch_frame()

# Printing x y z of some point
print(frame.points[0])
# Printing r g b color of some point
print(frame.colors[0])

# Saving test data frame from camera (RGBD format)
cam.save("test.ply")

# Loading test data frame from file (RGBD format)
pcd = cam.open("test.ply")
```

```

# Visualizing test data frame
cam.visualization_of_points(pcd)
```

Файл src/rotation-vector-map.py

```
#IN DEGREES!
# degree: [rx, ry, rz]
rotation_vector_map = {}


import numpy as np
import os
path = os.path.join(os.getcwd(), 'position_vector.npy')
v = np.load(path)
for i in v:
    rotation_vector_map[i[0]] = [i[1], i[2], i[3]]
```

Файл src/OperateCamera.py

```
import pyrealsense2 as rs
from enum import IntEnum
import open3d as o3d
import numpy as np


class OperateCamera:
    class Preset(IntEnum):
        Custom = 0
        Default = 1
        Hand = 2
        HighAccuracy = 3
        HighDensity = 4
        MediumDensity = 5

    def __init__(self, clipping_distance=2, resolution=0):
        """
            resolution = 0(1280x720), 1(640x480)
        """
        # Configure depth and color streams
        # Create a pipeline
        self.pipeline = rs.pipeline()

        # Create a config and configure the pipeline to stream
        #  different resolutions of color and depth streams
        self.config = rs.config()


        #resolution = 0(1280x720), 1(640x480)
```

```
30          if resolution==1:
31              self.config.enable_stream(rs.stream.depth, 640, 480,
                ↪  rs.format.z16, 30)
32              self.config.enable_stream(rs.stream.color, 640, 480,
                ↪  rs.format.rgb8, 30)
33          else:
34              self.config.enable_stream(rs.stream.depth, 1280, 720,
                ↪  rs.format.z16, 30)
35              self.config.enable_stream(rs.stream.color, 1280, 720,
                ↪  rs.format.rgb8, 30)
36
37          # Start streaming
38          profile = self.pipeline.start(self.config)
39          self.depth_sensor = profile.get_device().first_depth_sensor()
40
41          # Using preset HighAccuracy for recording
42          self.depth_sensor.set_option(rs.option.visual_preset,
            ↪  self.Preset.HighAccuracy)
43
44          # Getting the depth sensor's depth scale (see rs-align example for
            ↪  explanation)
45          self.depth_scale = self.depth_sensor.get_depth_scale()
46
47          # We will not display the background of objects more than
48          #  clipping_distance_in_meters meters away
49          self.clipping_distance_in_meters = clipping_distance  # 3 meter
50
51          # Create an align object
52          # rs.align allows us to perform alignment of depth frames to others
            ↪  frames
53          # The "align_to" is the stream type to which we plan to align depth
            ↪  frames.
54          align_to = rs.stream.color
55          self.align = rs.align(align_to)
56
57      def catch_frame(self):
58          print('Started catching realsence data')
59          for i in range(5):
60              # Get frameset of color and depth
61              frames = self.pipeline.wait_for_frames()
62
63              # Align the depth frame to color frame
64              aligned_frames = self.align.process(frames)
65
66              # Get aligned frames
67              aligned_depth_frame = aligned_frames.get_depth_frame()
68              color_frame = aligned_frames.get_color_frame()
69              intrinsic = o3d.camera.PinholeCameraIntrinsic(
70                  self.__get_intrinsic_matrix(color_frame))
71
```

```
72              # if aligned_depth_frame and color_frame:
73              #     break
74
75          depth_image = o3d.geometry.Image(
76              np.array(aligned_depth_frame.get_data()))
77          color_temp = np.asarray(color_frame.get_data())
78          color_image = o3d.geometry.Image(color_temp)
79
80          rgbd_image = o3d.geometry.RGBDImage.create_from_color_and_depth(
81              color_image,
82              depth_image,
83              depth_scale=1.0 / self.depth_scale,
84              depth_trunc=self.clipping_distance_in_meters,
85              convert_rgb_to_intensity=False)
86          temp = o3d.geometry.PointCloud.create_from_rgbd_image(
87              rgbd_image, intrinsic)
88          flip_transform = [[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, 0], [0, 0,
            ↪  0, 1]]
89          temp.transform(flip_transform)
90
91          self.pcd = o3d.geometry.PointCloud()
92          self.pcd.points = temp.points
93          self.pcd.colors = temp.colors
94
95          # o3d.io.write_point_cloud(f'{add_path}dataset/out.ply', pcd)
96          print('Stopped catching realsence data')
97          return self.pcd, color_image, depth_image
98
99
100     def catch_frame_without_pcd(self):
101         #resolution = 0(1280x720), 1(640x480)
102         print('Started catching realsence data; resolution mode ')
103
104         for i in range(5):
105             # Get frameset of color and depth
106             frames = self.pipeline.wait_for_frames()
107
108             # Align the depth frame to color frame
109             aligned_frames = self.align.process(frames)
110
111             # Get aligned frames
112             aligned_depth_frame = aligned_frames.get_depth_frame()
113             color_frame = aligned_frames.get_color_frame()
114             intrinsic = o3d.camera.PinholeCameraIntrinsic(
115                 self.__get_intrinsic_matrix(color_frame))
116
117             # if aligned_depth_frame and color_frame:
118             #     break
119
120         depth_image = o3d.geometry.Image(
```

```
121             np.array(aligned_depth_frame.get_data()))
122         color_temp = np.asarray(color_frame.get_data())
123         color_image = o3d.geometry.Image(color_temp)
124 
125         return color_image, depth_image
126 
127 
128     def save(self, filename):
129         o3d.io.write_point_cloud(filename, self.pcd)
130 
131     def __get_intrinsic_matrix(self, frame):
132         intrinsics = frame.profile.as_video_stream_profile().intrinsics
133         out = o3d.camera.PinholeCameraIntrinsic(640, 480, intrinsics.fx,
134                                                 intrinsics.fy, intrinsics.ppx,
135                                                 intrinsics.ppy)
136         return out
137 
138     @staticmethod
139     def open(filename):
140         return o3d.io.read_point_cloud(filename)
141 
142     @staticmethod
143     def visualization_of_points(points):
144         o3d.visualization.draw_geometries([points], 'Demonstration', 1080,
        ↪  720)
145 
146     def stop(self):
147         self.pipeline.stop()
```

Файл src/requirements.txt

```
1 pyrealsense2
2 numpy
3 opencv-python
4 open3d
5 git+https://github.com/jkur/python-urx
```