Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовой проект

Тема: Создание программы для автоматизированного сканирования с помощью робота

Дисциплина: Объектно-ориентированное программирование

Студент гр. 3331506/00401 Орехов А.М.
Преподаватель Ананьевский М.С.

«__»_____2023 г.

Санкт-Петербург
2023

1 Введение

3D сканирование – это систематический процесс определения координат точек, принадлежащих поверхностям сложных физических объектов (в частности, деталей) с целью последующего получения их пространственных математической моделей, которые могут анализироваться, модифицироваться с помощью CAD-систем, или воспроизведены с помощью методов цифрового производства.[1] Устройства, с помощью которых осуществляется сканирование объектов, называют 3D-сканерами.

Базовая задача 3D-сканера при работе – внутри каждого кадра решение задачи определения характерных точек на изображении и определение их положения относительно локальной системы координат сканера. В случае, если скан состоит из нескольких снимков, дополнительно должна быть решена задача определения смещения между положениями сканера между кадрами. Традиционно для этой задачи применяются математические методы, подкрепляемые дополнительной информацией с инерциальных датчиков или внешней системы локального позиционирования. Это позволяет использовать сканер «вручную» человеком, но серьезно усложняет и увеличивает стоимость всей системы. Однако, в случае установки сканера на промышленном роботе, что является относительно распространенным сценарием в промышленности, необходимость во внешней системе позиционирования отпадает в силу того, что робот способен перемещать сканер с повторяемостью, измеряемой десятками-сотнями микрон, что зачастую превосходит точность работы внешних систем позиционирования. На рисунке 1.1 показан пример подобного система локального позиционирования из инфракрасных меток на роботизированном сканере Creaform Metrascan.

Рисунок 1.1 – Сканер Creaform Metrascan

Таким образом, целью работы стало создание роботизированной системы сканирования, где задача позиционирования сканера в глобальной системе координат решается за счет данных с робота.

2 Описание задачи

В данной работе в качестве сканирующего устройства была использована времяпролетная (ToF) камера глубины LucidLabs Helios2 (Рисунок 2.1). Подход, использыемый этим классом сканеров для получения координат точек состоит в измерении расстояния до множества точек за счет измерения времени, затраченном лазерным лучом на полет до соответствующей точки, отражения и детектирования сенсором сканера. Данный подход нечасто используется в промышленных сканерах ввиду ограничений, связанных с природой используемого света – возможны эффекты множественного отражения и потери фазы сигнала при отражении от металлических объектов, однако, для рассматриваемой экспериментальной задачи решающими такие плюсы, как компактность, простота подключения и получения данных, а также скорость работы.

Рисунок 2.1 - ToF камера LucidLabs Helios 2

В качестве робота, на котором будет отрабатываться алгоритм, был выбран Kawasaki BA006 (Рисунок 2.2) из семейства сварочных роботов. Их преимуществами является относительно высокая повторяемость (± 0.05мм) при большом радиусе работы (1.5м).

Рисунок 2.2 - Kawasaki BA006L

Таким образом, можно сформулировать задачи, которые требовалось решить на пути создания прототипа устройства:

1. Планирование архитектуры программы
2. Получение данных с камеры при помощи предоставленного производителем SDK.
3. Создание библиотеки для обработки сырых данных, «сшивания» полученных с разных ракурсов. облаков точек, фильтрации и сохранения результатов.
4. Создание библиотеки для удаленного управления роботом.
5. Объединение этапов в ПО для сканирования, тесты.

3 Ход работы

3.1 Планирование архитектуры программы:

При планировании архитектуры разрабатываемой программы принимались во внимание уже имеющиеся наработки лаборатории, на базе которой я выполнял свой проект. В частности, для дистанционного управления роботом существует готовый и оттестированный модуль, осуществляющий связь с роботом по протоколу Telnet, написанный на Python. Для ускорения разработки на финальных этапах было решено использовать именно Python для взаимодействия с роботом из-за высокой скорости прототипирования, которую предоставляет этот язык. Поскольку взаимодействия с роботом происходят фактически по текстовому каналу связи, скорость более низкоуровневых языков для данного сценария применения не была бы решающим преимуществом.

В случае же с обработкой данных камеры скорость алгоритма является важнейшим требованием. В идеальной ситуации большой объем получаемых с камеры данных хочется обрабатывать в реальном времени, что дает временную оценку в 33 миллисекунды на кадр с матрицы разрешением 640 на 480. По этой причине было принято решение написания отдельной библиотеки на C++ с передачей описанных в ней функций в код на Python через механизм динамически связываемых библиотек (.dll в Windows и .so в LINUX). Данный подход является распространенным в библиотеках для Python.

3.2 Получение данных с камеры при помощи предоставленного производителем SDK.

Для взаимодействия со своими камерами компания Lucid Labs предоставляет комплекс инструментов для разработки ПО (software development kit, SDK) Arena SDK. Примечательно, что данный комплекс имеет схожие методы как для работы с ToF, так и с обычными и тепловизионными камерами компании, поскольку все они используют распространенный в промышленности интерфейс GenICam [2].

Основные операции, предоставляемые SDK представляют собой функции установки настроек камеры, начала сеанса получения данных и его остановки, получения кадра, а также обнуления очереди кадров, что полезно, если получение данных происходит быстрее, чем пользователь способен их обработать и всегда обеспечивает получение пользователем актуальных данных. В ходе работы было решено все методы, управляющие работой устройств в виде робота и камеры оставить на высокоуровневой стороне, написанной на Python. В библиотеку для обработки же передаются данные о координатах робота, на которых был получен кадр и указатель на область памяти, в которой SDK хранит первый в очереди отснятый кадр.

При нормальной работе камеры каждый кадр представляет собой массив из 921600 переменных типа unsigned int, в котором последовательно записаны по 3 координаты каждой точки. Производителем, с целью уменьшения размера пакета отправляемых данных, используется прием, при котором координаты каждой точки умножаются на константу (по умолчанию – 4) и переводятся из формата с плавающей точкой в целое число. С точностью камеры в районе четверти миллиметра это является допустимым. После чего к данным прибавляется величина максимального значения данных типа int с целью избавиться от возможных отрицательных точек и записать координату в формат unsigned int. Также, возможна ситуация отсутствия данных в точке, что камера отмечает значением 65535.

Поскольку скорость получения данных камерой существенно отличается от скорости работы робота, было решено отделить действия, связанные с этим в параллельный поток с помощью модуля threading. Класс камеры был создан с использованием функционала менеджера контекста Python для корректного завершения работы с камерой даже при возникновении исключений в процессе работы. Код модуля для работы с камерой представлен в приложении 1.

3.3 Создание библиотеки для обработки сырых данных, «сшивания» полученных с разных ракурсов. облаков точек, фильтрации и сохранения результатов.

Первым этапом после получения массива данных с камеры является фильтрация «валидных» точек и обратное преобразование в формат трехмерных векторов float. Данные для этого в виде масштаба и смещения данных могут быть получены с камеры. Далее следует осуществление преобразования на точках с целью перевода их из локальной системы координат связанной с корпусом камеры в глобальную, за счет данных о координате эффектора робота, заранее известных пользователю. Для поворота точек было решено применить математику кватернионов – системы гиперкомплексных чисел, образующий четырехмерное векторное пространство. Кватернионы получили распространение в механике и компьютерной графике для описания трех и четырехмерных объектов, и задаются относительно простым набором свойств и законов [3].

Для задачи поворота точек облака точек были созданы отдельные структуры вектора и кватерниона для удобной работы в коде. При обработке, каждая из точек преобразуется в кватернион, над ней производятся операции поворота последовательно по трем осям. Далее кватернион преобразуется обратно в трехмерный вектор, к которому прибавляется вектор, описывающий параллельный перенос СК камеры относительно СК робота.

Полученные точки записываются во временный массив. Цель этого – дать пользователю возможность в будущих версиях программы выбирать, следует ли сохранить кадр в основное хранилище или нет. При сохранении в хранилище, на основании точек массива изменяются значения максимальной и минимальной точки, образующей область пространства, вмещающей целиком все облако точек. Это необходимо для следующего этапа – фильтрации облака точек.

Фильтрация облака точек является ключевой задачей в процессе обработки скана, поскольку, при выборе соответствующего алгоритма, способна удалять лишние, не связанные с объектом съемки точки, сглаживать геометрию, упрощая

следующий логически процесс сшивания точек в полигональную сетку, а также ускорять этот процесс за счет уменьшения объема информации [4]. В текущей работе был выдвинут оригинальный алгоритм, появившийся вследствие упрощения первоначальной идеи.

Задуманный изначально алгоритм предусматривал вычисление расстояния между соседними точками и удаление точек без соседей, как и точек, которые находятся слишком близко к соседям. Однако, вычисление расстояния между точками в облаке из десятков миллионов точек без предварительной подготовки может быть решено с помощью N измерений по формуле

$$N = \frac{V \cdot (V - 1)}{2}$$

Где V – число точек в облаке. Соответственно, временная сложность такого алгоритма будет квадратичной, что неприемлемо для таких объемов данных. Решением этого будет разбиение пространства на элементарные ячейки и проверка расстояния между точками только внутри одной из нескольких соседних ячеек.

На этапе реализации описанного алгоритма концепция была упрощена до того, что все точки, попавшие внутрь одной элементарной ячейки, усредняют свои координаты в одну результирующую точку. При таком подходе теряется часть информации, что нельзя назвать идеальным, но на этапе тестирования было решено оставить этот вариант. Если точек в ячейке меньше порогового значения, то точки удаляются, за счет этого обеспечивается фильтрация «летающих точек»

Также, для сохранения результатов и их анализа были реализованы два метода для сохранения фильтрованного и оригинального облака точек в файл формата .ply, являющегося распространенным типом данных для подобного типа геометрий. Для этого создается бинарный текстовый файл, в который записывается заголовок в соответствии с описанием формата .ply [5] и далее массив точек.

Код модуля был скомпилирован в виде динамической библиотеки с соответствующей ответной частью на Python.

Исходный код библиотеки, полученной на этом этапе представлен в приложении 2.

3.4 Создание библиотеки для удаленного управления роботом.

Как было сказано ранее, описываемый в этой части модуль был написан на Python, с использованием готового кода для общения с роботом по протоколу Telnet. Модуль обеспечивает такой функционал, как:

- Корректная установка и завершения соединения с роботом через sockets
- Запись/чтение переменных в памяти робота с контролем ошибок
- Запуск программ из памяти робота
- Движение робота в точку

К описанному модулю были добавлены также класс векторов для записи положения робота в программе, и класс Position, описывающий позицию робота в виде шести пространственных координат -трех параллельных X, Y, Z и трех угловых, задаваемых углами Эйлера O, A, T. Для последнего были реализованы методы для конвертации позиции, задаваемой вектором положения и вектором направления в вышеописанные 6 координат.

Код модуля управления роботом представлен в приложении 3.

3.5 Объединение этапов в ПО для сканирования, тесты.

Для тестирования работы программы был написан код, показанный в приложении 4. Он включает в себя функцию, генерирующую точки окружности с заданной частотой и главную функцию, создающую массив точек для «облета» вокруг точки интереса по создаваемой траектории, удерживая объект съемки в центре кадра. В результате работы программы были получены облака точек, показанные на рисунках 3.1-3.3. Для обзора полученных точек использовалось ПО FreeCAD.

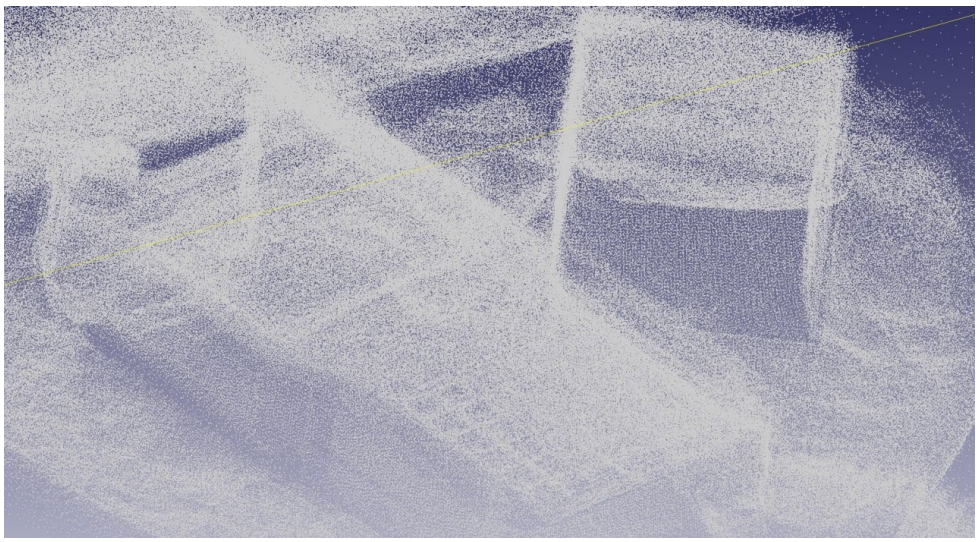

Рисунок 3.1 – Скан пульта робота без фильтрации

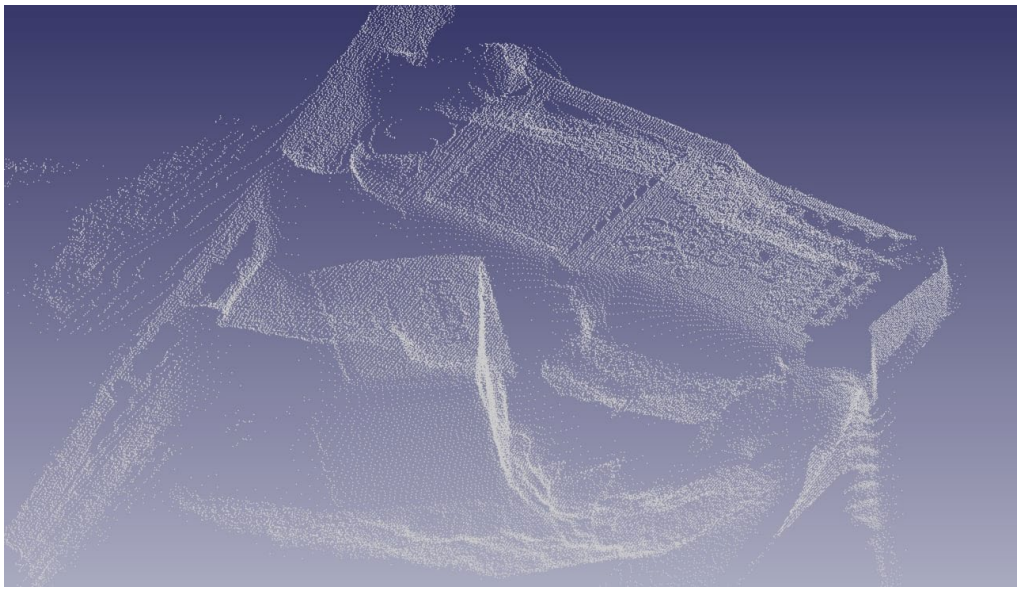

Рисунок 3.2 – Скан пульта робота, с фильтрацией

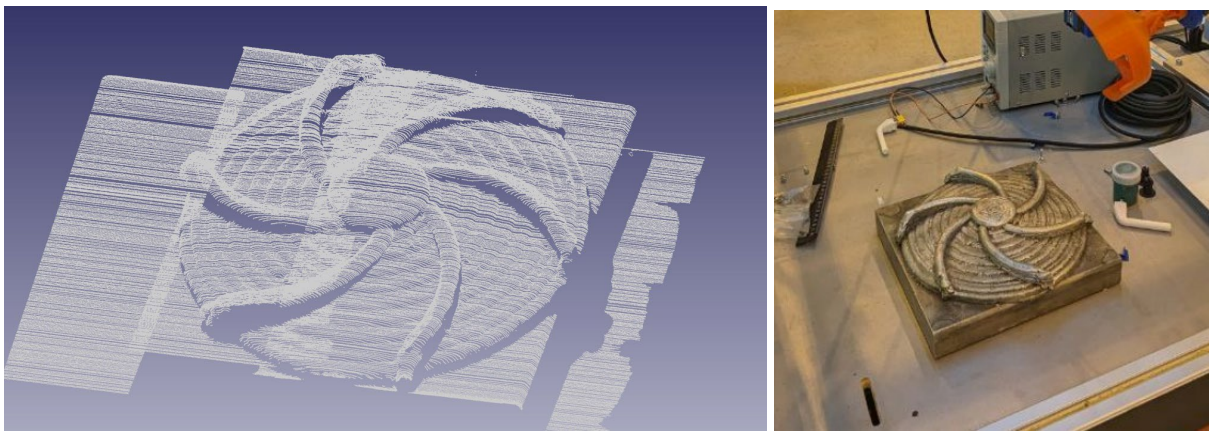

Рисунок 3.3 – Скан напечатанного из металла импеллера

4 Список литературы

4.1 Гордлеев С.Д. ПРОБЛЕМЫ И ПЕРСПЕКТИВЫ ТЕХНОЛОГИИ СКАНИРОВАНИЯ 3D ПОВЕРХНОСТЕЙ // Научные проблемы водного транспорта. 2014. №41. URL: https://cyberleninka.ru/article/n/problemy-i-perspektivy-tehnologii-skanirovaniya-3d-poverhnostey (дата обращения: 02.06.2022).

4.2 European Machine Vision Association, GenICam Standard description. URL: https://www.emva.org/wp-content/uploads/GenICam_Standard_v2_0.pdf (Дата обращения 29.05.2022)

4.3 Амелькин Н.И. Кинематика и динамика твердого тела // Московский физико-технический институт (государственный университет), Москва 2000г. URL: https://mipt.ru/dasr/upload/c8c/amelkin_rigidbody-arphh81ii9w.pdf (Дата обращения: 15.06.2022).

4.4 Xian-Feng Han, Jesse S. Jin, Ming-Jie Wang, Wei Jiang, Lei Gao, Liping Xiao, A review of algorithms for filtering the 3D point cloud, Signal Processing: Image Communication, Volume 57, 2017, Pages 103-112, URL: https://www.sciencedirect.com/science/article/abs/pii/S0923596517300930 (Дата обращения 28.07.2022)

4.5 Описание устройства формата .PLY (Stanford Triangle Format). URL: http://paulbourke.net/dataformats/ply/ (Дата обращения 6.07.2022)

Приложение 1 - Код модуля для работы с камерой

```python
from arena_api.enums import PixelFormat
from arena_api.system import system
import threading
import time


aorehov
class CameraHandler(threading.Thread):
    aorehov
    def __init__(self):
        super(CameraHandler, self).__init__()
        self.daemon = True

        self._stopped = threading.Event()
        self.new_frame = threading.Event()
        self.ready = threading.Barrier(2, timeout=5)
        self._running = True

        self.device = self.create_devices()[0]
        print(f"Found device:\n\t{self.device}")

        self.device_initial_state = None
        self.nodemap = None
        self.buffer = None

        self.setup()
        self.start()

    aorehov
    def __enter__(self):
        return self

    aorehov
    def run(self):
        with self.device.start_stream(1):  # Запуск потока изображений создает буфер и заполняет его первым кадром
            print("\nStream started with 1 buffer")
            time.sleep(0.5)
            self.buffer = self.device.get_buffer()
            print("buffer received")
            self.ready.wait()
            print("Acquisition thread started")
            while self._running:
                self.new_frame.wait()
                self.device.requeue_buffer(self.buffer)
                self.buffer = self.device.get_buffer()
```

```python
                self.new_frame.clear()
            self.device.requeue_buffer(self.buffer)
        self._stopped.set()

    aorehov
    @staticmethod
    def create_devices():
        tries = 0
        tries_max = 5
        sleep_time = 10
        while tries < tries_max:
            devices = system.create_device()
            if not devices:
                print(f"Try {tries+1} of {tries_max}: waiting for {sleep_time} secs for a device to be connected!")
                for sec in range(sleep_time):
                    time.sleep(1)
                    print(f"{sec + 1} seconds passed ." * sec, end="\r")
                tries += 1
            else:
                print(f"Created {len(devices)} device(s)")
                return devices
        else:
            raise Exception("No device found! Please connect a device and run the example again.")

    aorehov
    def setup(self):
        tl_stream_nodemap = self.device.tl_stream_nodemap  # Get device stream nodemap
        tl_stream_nodemap['StreamAutoNegotiatePacketSize'].value = True  # Enable stream auto negotiate packet size
        tl_stream_nodemap['StreamPacketResendEnable'].value = True  # Enable stream packet resend

        self.nodemap = self.device.nodemap  # Store nodes' initial values

        # get node values that will be changed in order to return their values at
        # the end of the example
        self.device_initial_state = self.nodemap['PixelFormat'].value, self.nodemap['Scan3dOperatingMode'].value

        # Set nodes ------------------------------------------------------------
        print('\nSettings nodes:')
        pixel_format = PixelFormat.Coord3D_ABC16
        print(f'Setting pixel format to { pixel_format.name}')
        self.nodemap.get_node('PixelFormat').value = pixel_format

        print('\tSetting 3D operating mode')
        self.nodemap['Scan3dAmplitudeGain'].value = 0.4
        #self.nodemap['Scan3dOperatingMode'].value = 'Distance4000mmSingleFreq'
```

```
        self.nodemap['Scan3dOperatingMode'].value = 'Distance8300mmMultiFreq'

        self.nodemap["Scan3dImageAccumulation"].value = 30
        self.nodemap["Scan3dFlyingPixelsRemovalEnable"].value = True
        self.nodemap["Scan3dFlyingPixelsDistanceThreshold"].value = 20

    aorehov
    def get_stream_data(self):
        return self.buffer.pdata, 640, 480, 3

    aorehov
    def stop(self):
        self._running = False  # Останавливаем цикл
        self._stopped.wait()  # Ожидаем окончания цикла и остановки потока с камеры
        self.nodemap['PixelFormat'].value = self.device_initial_state[0]
        self.nodemap['Scan3dOperatingMode'].value = self.device_initial_state[1]  # Возвращаем первоначальные значения
        system.destroy_device()
        print("Destroyed all created devices")

    aorehov
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.stop()
```

Приложение 2 - Исходный код библиотеки обработки облаков точек

```
#include <iostream>
#include <chrono>
#include <vector>
#include <unordered_map>
#include "vec.h"
#include "position.h"
#include "quaternion.h"

using vector_map = std::unordered_map<int, std::pair<int, VecXYZ>>;
const float PI = 3.141592;

class pointCloud {
private:
    const uint16_t *buffer = nullptr;  // Бубер, откуда получаем данные
    const int capacity = 20000000;  // Максимальный объем облака
    int size, pixel_size;  // Количество точек в облаке и размер каждой точки
    int cell_size = 3;  // Размер элементарной ячейки для фильтрации
    const int filter_threshold = 1; // Минимальное количество точек внутри элементарной ячейки, которое будет
    // пропускаться при фильтрации (удаляет "летающие" точки)

    VecXYZ* cloud = (VecXYZ*) malloc( Size: sizeof(VecXYZ) * capacity);  // Основной массив для хранения точек
    VecXYZ* filtered_cloud = (VecXYZ*) malloc( Size: sizeof(VecXYZ) * capacity);
    vector_map filtered_map;

    VecXYZ* temp_frame;  // Массив для временного хранения только что обработанного кадра с камеры
    int temp_frame_size = 0;
    float* vis_array = (float*) malloc( Size: sizeof(float) * 3 * capacity);

    float min_x = 0, max_x = 0, min_y = 0, max_y = 0, min_z = 0, max_z = 0;
    int point_number = 0;
    int point_number_f = 0;

    // Константы, нормально получаемые с камеры
    int min_z_threshold = 0 * 4;
    int max_z_threshold = 8000 * 4;
    Position offset = {.x: 0, .y: 0, .z: 0, .roll: 0, .pitch: 0, .yaw: 0};
    float scale = 0.25f;

    static inline VecXYZ q_rotate(VecXYZ& v, Quaternion& q) {
        return (q * v * q.conjugate()).vector();
    }

    void addPoint(VecXYZ v) {
        cloud[point_number] = v;
        if (point_number == 0) {
            min_x = v.x;
            min_y = v.y;
            min_z = v.z;
            max_x = v.x;
            max_y = v.y;
            max_z = v.z;
        }
        point_number += 1;
        if (v.x < min_x) min_x = v.x;
        else if (v.x > max_x) max_x = v.x;

        if (v.y < min_y) min_y = v.y;
        else if (v.y > max_y) max_y = v.y;

        if (v.z < min_z) min_z = v.z;
        else if (v.z > max_z) max_z = v.z;
    }

public:
    pointCloud(const uint16_t *new_buffer, int width, int height, int pixelSize) {
        buffer = new_buffer;
        size = width * height;
        temp_frame = (VecXYZ*) malloc( Size: sizeof(VecXYZ) * size);
        pixel_size = pixelSize;
    }


    void loadFromRaw(const char* filename) {
        FILE *f;
        fopen_s( Stream: &f, FileName: filename, Mode: "rb");
        if (f) std::cout << "file "<< filename << " found\n";
        else {
            std::cout << "file " << filename << " not found\n";
            return;
        }
        auto data = (uint16_t*) malloc( Size: size * pixel_size * sizeof(uint16_t));
        fread( Buffer: data, ElementSize: sizeof(uint16_t), ElementCount: size * pixel_size, Stream: f);
        buffer = data;
        fclose( Stream: f);
    }
```

```
int processFrame(float pos_x, float pos_y, float pos_z, float roll, float pitch, float yaw) {
    VecXYZ shift = { .x: pos_x,  .y: pos_y,  .z: pos_z};
    float o = roll - PI / 2 + offset.roll;
    float a = -pitch + offset.pitch;
    float t = yaw - PI / 2 + offset.yaw;
    Quaternion q_roll = { axis: { .x: 0,  .y: 0,  .z: 1},  &: t};
    Quaternion q_pitch = { axis: { .x: 1,  .y: 0,  .z: 0},  &: a};
    Quaternion q_yaw = { axis: { .x: 0,  .y: 0,  .z: 1},  &: o};
    Quaternion roll_c = q_yaw.conjugate();
    Quaternion pitch_c = q_pitch.conjugate();
    Quaternion yaw_c = q_roll.conjugate();
    temp_frame_size = 0;
    for (unsigned int idx = 0; idx < size * pixel_size - 2; idx += pixel_size) {
        if (min_z_threshold < buffer[idx + 2] && buffer[idx + 2] < max_z_threshold) {
            VecXYZ point = { .x: (float)buffer[idx],  .y: (float)buffer[idx + 1],  .z: (float)buffer[idx + 2]};
            Quaternion q_point = { v: (point + offset.xyz()) * scale};
            q_point = q_roll * q_point * roll_c;
            q_point = q_pitch * q_point * pitch_c;
            q_point = q_yaw * q_point * yaw_c;
            temp_frame[temp_frame_size++] = q_point.vector() + shift;
            vis_array[(point_number + temp_frame_size) * 3 - 2] = point.x;
            vis_array[(point_number + temp_frame_size) * 3 - 1] = point.y;
            vis_array[(point_number + temp_frame_size) * 3]     = point.z;
        }
    }
    return temp_frame_size;
}

void saveFrame(){
    for (int i = 0; i < temp_frame_size; i++) addPoint( v: temp_frame[i]);
}

void filter(){
    std::cout << "filtering started\n";
    int size_y = (int(max_y) / cell_size) + 1;
    int size_z = (int(max_z) / cell_size) + 1;
    int index = 0;
    for (int idx = 0; idx < point_number; idx++){
        index = int(cloud[idx].z) / cell_size + int(cloud[idx].y) / cell_size * size_z +
                int(cloud[idx].x) / cell_size * size_y * size_z;
        if (filtered_map.contains( Keyval: index)) {
            filtered_map[index].first++;
            filtered_map[index].second += cloud[idx];
        }
        else {

            filtered_map.insert( Val: std::make_pair( &: index,  Val2: std::make_pair( Val1: 1,  &: cloud[idx])));
        }
    }

    for (auto &el :pair<...> & : filtered_map) {
        auto point_sum :pair<int, VecXYZ> = el.second;
        if (point_sum.first > filter_threshold) {  // Удаление "летающих" точек
            filtered_cloud[point_number_f++] = point_sum.second / (float)point_sum.first;
        }
    }
}

void save(const char* filename){
    FILE *file_out;
    fopen_s( Stream: &file_out,  FileName: filename,  Mode: "wb");
    std::string header = "ply\n";
    header += "format binary_little_endian 1.0\n";
    header += "element vertex " + std::to_string( Val: point_number) + "\n";
    std::cout << "number of points - " << point_number << "\n";
    header += "property float32 x\n";
    header += "property float32 y\n";
    header += "property float32 z\n";
    header += "end_header\n";
    fwrite( Buffer: header.c_str(),  ElementSize: header.length(),  ElementCount: 1,  Stream: file_out);
    for(unsigned long i = 0; i<point_number; i++){
        float point[3] = { [0]: cloud[i].x,  [1]: cloud[i].y,  [2]: cloud[i].z};
        fwrite( Buffer: &point,  ElementSize: sizeof(point),  ElementCount: 1,  Stream: file_out);
    }
    fclose( Stream: file_out);
}

void save_filtered(const char* filename){
    FILE *file_out;
    fopen_s( Stream: &file_out,  FileName: filename,  Mode: "wb");
    std::string header = "ply\n";
    header += "format binary_little_endian 1.0\n";
    header += "element vertex " + std::to_string( Val: point_number_f) + "\n";
    std::cout << "number of points - " << point_number_f << "\n";
    header += "property float32 x\n";
    header += "property float32 y\n";
    header += "property float32 z\n";
    header += "end_header\n";
    fwrite( Buffer: header.c_str(),  ElementSize: header.length(),  ElementCount: 1,  Stream: file_out);
    for(unsigned long i = 0; i<point_number_f; i++){
        float point[3] = { [0]: filtered_cloud[i].x,  [1]: filtered_cloud[i].y,  [2]: filtered_cloud[i].z};
```

```
            fwrite( Buffer: &point, ElementSize: sizeof(point), ElementCount: 1, Stream: file_out);
        }
        fclose( Stream: file_out);
    }

    float* getVisBuffer(){
        return vis_array;
    }

    void set_cell_size(int new_size){
        cell_size = new_size;
    }

    void set_buffer(const uint16_t* new_buffer){
        buffer = new_buffer;
    }

    void clear(){
        free( Block: cloud);
        free( Block: filtered_cloud);
        free( Block: temp_frame);
    }
};


extern "C"{
    __declspec(dllexport) pointCloud* init(const uint16_t *buffer, int width, int height, int pixelSize)
    {return new pointCloud( new_buffer: buffer, width, height, pixelSize);}

    __declspec(dllexport) float* getVisBuffer(pointCloud *self) {return self->getVisBuffer();}

    __declspec(dllexport) int processFrame(pointCloud *self, float x, float y, float z, float o, float a, float t)
    {return self->processFrame( pos_x: x, pos_y: y, pos_z: z, roll: o, pitch: a, yaw: t);}

    __declspec(dllexport) void loadFromRaw(pointCloud *self, const char* filename) {return self->loadFromRaw(filename);}

    __declspec(dllexport) void saveFrame(pointCloud *self) {self->saveFrame();}

    __declspec(dllexport) void set_cell_size(pointCloud *self, int new_size) {self->set_cell_size(new_size);}

    __declspec(dllexport) void set_buffer(pointCloud *self, const uint16_t *buffer) {self->set_buffer( new_buffer: buffer);}

    __declspec(dllexport) void filter(pointCloud *self) {self->filter();}

    __declspec(dllexport) void save(pointCloud *self, const char* filename) {self->save(filename);}

    __declspec(dllexport) void save_filtered(pointCloud *self, const char* filename) {self->save_filtered(filename);}

    __declspec(dllexport) void clear(pointCloud *self) {self->clear();}
}
```

Приложение 3 – Исходный код модуля взаимодействия с роботом

```python
import socket
from dataclasses import dataclass
from math import pi, acos, sqrt, pow, degrees, radians, atan2, asin
import time
import re


aorehov *
class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'


aorehov
def sign(x):
    return 1 - 2 * (x < 0)


aorehov
def angle(dx, dy):
    return 2 * pi - acos(dx / (dx ** 2 + dy ** 2) ** 0.5) if dy < 0 else acos(dx / (dx ** 2 + dy ** 2) ** 0.5)


aorehov
def angle_dist(a1, a2):
    """ Функция определяет угловое расстояние между двумя углами по окружности """
    diff = (a2 - a1 + pi) % (2 * pi) - pi
    return diff if diff > -pi else diff + 2 * pi


aorehov
@dataclass
class VecXYZ:
    x: float
    y: float
    z: float

    aorehov
    def as_tuple(self):
        return [self.x, self.y, self.z]
```

```python
    def length(self):
        return sqrt(pow(self.x, 2) + pow(self.y, 2) + pow(self.z, 2))

    aorehov
    def normalize(self):
        factor = self.length()
        self.x /= factor
        self.y /= factor
        self.z /= factor
        return self

    aorehov
    def __getitem__(self, idx):
        return self.__getattribute__("xyz"[idx])

    aorehov
    def __add__(self, other):
        return VecXYZ(self.x + other.x,
                      self.y + other.y,
                      self.z + other.z)

    aorehov
    def __sub__(self, other):
        return VecXYZ(self.x - other.x,
                      self.y - other.y,
                      self.z - other.z)

    aorehov
    def __mul__(self, other):
        return VecXYZ(self.x * other,
                      self.y * other,
                      self.z * other)
```

```python
@dataclass
class Position:
    """Класс точек для робота Kawasaki. Инициализируется с помощью прямого указания координат или по вектору положения
     и направления"""

    x: float
    y: float
    z: float
    o: float
    a: float
    t: float

    aorehov
    def __init__(self, start, end, rot=0):
        pos = VecXYZ(*start)
        lookat = VecXYZ(*end)
        self.x, self.y, self.z = pos
        self.o, self.a, self.t = self.get_euler((lookat - pos).normalize(), rot)

    aorehov
    def __getitem__(self, idx):
        return self.__getattribute__("xyzoat"[idx])

    aorehov
    def get_radians(self):
        return [self.x, self.y, self.z] + list(map(radians, list(self)[3:]))

    aorehov
    @staticmethod
    def get_euler(vec: VecXYZ, rot: float):
        x, y, z = vec
        o_angle = atan2(y, x)
        a_angle = pi / 2 - asin(z)
        angles = [o_angle, a_angle, rot]
        return map(lambda ang: round(degrees(ang), 2), angles)


aorehov
class RobotCommunication:
    aorehov
    def __init__(self, ip, port):
        self.ip_address = ip
        self.port_number = port
        self.telnet_delay = 0.2
        self.telnet_connection_timeout = 1
        self.error_counter_limit = 10  # telnet_delay * error_counter_limit = time_in_second

        self.server = None

        self.robot_is_busy = False
        self.abort_operation = False

        # self.telnet = telnetlib.Telnet()

    aorehov
    def upload_variables(self, variables_text):
        # Return
        # 1 - everything is ok
        # -1 - not all variables uploaded correct
        # -1000 - communication error

        self.robot_is_busy = True
        self.abort_operation = False

        if self.connect() == -1:
            print("Can't establish connection with robot")
            return -1000

        list_of_strings = variables_text.split('\n')
        list_of_strings = [i for i in list_of_strings if i]  # Delete empty elements from string (\n in the end problem)
        num_variables = len(list_of_strings)

        for line in list_of_strings:
            counter = 0
            self.server.sendall((line + "\r\n").encode())
            self.server.sendall("\r\n".encode())
            while True:
                time.sleep(self.telnet_delay)
                receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n",
                                                                                                           " ")
                counter += 1
                if receive_string.find(">") > 0:
                    _ = self.server.recv(4096)
                    break
                if counter > self.error_counter_limit:
                    print("Transmission timeout is over")
                    return -1000
                if self.abort_operation:
                    self.close_connection()
                    self.robot_is_busy = False
```

```
    counter_variables = 0
    for line in list_of_strings:
        test_string = line.split(" ")
        if test_string != ['']:
            if test_string[0] == "POINT":
                var_name, var_value = self.read_variable(test_string[1], variable_type='position')

                var_in_array_strings = test_string[4:]
                var_in_array_floats = [None] * len(var_in_array_strings)
                for i in range(len(var_in_array_strings)):
                    var_in_array_floats[i] = float(var_in_array_strings[i][0:-1])

                # var_value += [0.0, 0.0, 0.0]

                if (test_string[1].lower() == var_name) and (var_in_array_floats == var_value):
                    counter_variables += 1
                else:
                    # pass
                    print("Warning: variable " + test_string[1] + " may be incorrect!!!")
                    print("Write: ", var_in_array_floats)
                    print("Read: ", var_value)
                    print("--------------------")
            else:
                var_name, var_value = self.read_variable(test_string[0], variable_type='real')
                var_value = float(var_value)
                var_in_array_float = float(test_string[2])
                if (test_string[0].lower() == var_name) and (var_in_array_float == var_value):
                    counter_variables += 1
                else:
                    # pass
                    print("Warning: variable " + test_string[0] + " may be incorrect!!!")
                    print("Write: ", var_in_array_float)
                    print("Read: ", var_value)
                    print("--------------------")

    self.close_connection()
    self.robot_is_busy = False

    if counter_variables != num_variables:
        print(f"{bcolors.WARNING}WARNING: Not all points is correct in robot memory{bcolors.ENDC}")
        errors_counter = str(counter_variables)
        print("Points correct:", f"{bcolors.FAIL}" + errors_counter + f"{bcolors.ENDC}", "/", num_variables)
        return -1
```

```
def read_variable(self, variable_name, variable_type='real'):
    error_counter = 0
    if variable_type == 'position':
        self.server.sendall(("list /l " + variable_name + "\r\n").encode())
        self.server.sendall("\r\n".encode())
        while True:
            time.sleep(self.telnet_delay)
            error_counter += 1
            receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n", " ")
            if receive_string.find(">") > 0:
                _ = self.server.recv(4096)
                position_string_list = re.split(' +', receive_string)
                read_point_name = position_string_list[4]
                read_point_position_strings = position_string_list[5:-2]
                read_point_position_floats = list(map(float, read_point_position_strings))
                return read_point_name, read_point_position_floats
            if error_counter > self.error_counter_limit:
                print("Receive timeout is over")
                return "", -1000

    if variable_type == 'real':
        self.server.sendall(("list /r " + variable_name + "\r\n").encode())
        self.server.sendall("\r\n".encode())
        while True:
            time.sleep(self.telnet_delay)
            error_counter += 1
            receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n", " ")
            if receive_string.find(">") > 0:
                _ = self.server.recv(4096)
                real_string_list = re.split(' +', receive_string)
                read_real_name = real_string_list[4]
                read_real_value = float(real_string_list[6])
                return read_real_name, read_real_value
            if error_counter > self.error_counter_limit:
                print("Receive timeout is over")
                return "", -1000

    aorehov
def read_variable_real(self, variable_name):
    self.robot_is_busy = True
    self.abort_operation = False
```

```
    if self.connect() == -1:
        print("Can't establish connection with robot")
        return -1

    error_counter = 0

    self.server.sendall(("list /r " + variable_name + "\r\n").encode())
    self.server.sendall("\r\n".encode())

    while True:
        time.sleep(self.telnet_delay)
        error_counter += 1
        receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n", " ")
        if receive_string.find(">") > 0:
            _ = self.server.recv(4096)
            real_string_list = re.split(' +', receive_string)
            read_real_value = float(real_string_list[6])
            self.close_connection()
            self.robot_is_busy = False
            return read_real_value

        if error_counter > self.error_counter_limit:
            self.close_connection()
            self.robot_is_busy = False
            self.abort_operation = False
            print("Receive timeout is over")
            return -1000

        if self.abort_operation:
            self.close_connection()
            self.robot_is_busy = False
            self.abort_operation = False
            return -1000

aorehov
def execute_program(self, program_name):
    #  Return:
    # 1 - everything is ok
    # 2 - XAC error detected
    # 3 - program HALT error detected
    # -1000 - communication with robot was aborted

    self.robot_is_busy = True
    self.abort_operation = False
```

```
    if self.connect() == -1:
        print("Can't establish connection with robot")
        return -1000

    error_counter = 0

    self.server.sendall(("exe " + program_name + "\r\n").encode())
    self.server.sendall("\r\n".encode())
    time.sleep(self.telnet_delay)

    if self.telnet_delay != 0:
        while True:
            time.sleep(self.telnet_delay)
            error_counter += 1
            receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n", " ")
            if receive_string.find("completed.No = 1") > 0:
                _ = self.server.recv(4096)
                print(f"{bcolors.WARNING}Program complete{bcolors.ENDC}")
                self.close_connection()
                self.robot_is_busy = False
                return 1

            if receive_string.find("(E6509) No work detected") > 0:
                _ = self.server.recv(4096)
                print(f"{bcolors.WARNING}Program not complete. TS error detected.{bcolors.ENDC}")
                self.close_connection()
                self.robot_is_busy = False
                return 2

            if receive_string.find("Program halted.No = 1") > 0:
                _ = self.server.recv(4096)
                print(f"{bcolors.WARNING}Program not complete. Program halted.{bcolors.ENDC}")
                self.close_connection()
                self.robot_is_busy = False
                return 3

            if self.abort_operation:
                self.close_connection()
                self.robot_is_busy = False
                self.abort_operation = False
                return -1000
```

```python
def move_point(self, point: Position):
    #  Return:
    # 1 - everything is ok
    # 2 - XAC error detected
    # 3 - program HALT error detected
    # -1000 - communication with robot was aborted

    self.robot_is_busy = True
    self.abort_operation = False

    if self.connect() == -1:
        print("Can't establish connection with robot")
        return -1000

    error_counter = 0

    self.server.sendall(("do jmove trans" + str(tuple(point)) + "\r\n").encode())
    time.sleep(self.telnet_delay)

    if self.telnet_delay != 0:
        while True:
            time.sleep(self.telnet_delay)
            error_counter += 1
            receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n", " ")
            if receive_string.find("completed") > 0:
                _ = self.server.recv(4096)
                print(f"{bcolors.WARNING}line executed{bcolors.ENDC}")
                self.close_connection()
                self.robot_is_busy = False
                return 1

            if receive_string.find("(E6509) No work detected") > 0:
                _ = self.server.recv(4096)
                print(f"{bcolors.WARNING}Program not complete. TS error detected.{bcolors.ENDC}")
                self.close_connection()
                self.robot_is_busy = False
                return 2

            if receive_string.find("Program halted.No = 1") > 0:
                _ = self.server.recv(4096)
                print(f"{bcolors.WARNING}Program not complete. Program halted.{bcolors.ENDC}")
                self.close_connection()
                self.robot_is_busy = False
                return 3

            if self.abort_operation:
                self.close_connection()
                self.robot_is_busy = False
                self.abort_operation = False
                return -1000
```

aorehov

```python
def connect(self):
    error_counter = 0
    self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server.connect((self.ip_address, self.port_number))
    self.server.sendall(b"as\r\n")
    time.sleep(self.telnet_connection_timeout)
    while True:
        time.sleep(self.telnet_delay)
        error_counter += 1
        receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n", " ")
        if receive_string.find("This is AS monitor terminal") > 0:
            _ = self.server.recv(4096)
            print("Connection with robot established")
            return 0
        if error_counter > self.error_counter_limit:
            self.robot_is_busy = False
            print("Receive timeout is over")
            return -1
```

aorehov

```python
def ereset(self):
    #  Return:
    # 1 - everything is ok
    # -1000 - communication with robot was aborted

    self.robot_is_busy = True

    if self.connect() == -1:
        print("Can't establish connection with robot")
        return -1000

    error_counter = 0

    self.server.sendall("ereset\r\n".encode())
    self.server.sendall("\r\n".encode())
    time.sleep(self.telnet_delay)
```

```
412
413         if self.telnet_delay != 0:
414             while True:
415                 time.sleep(self.telnet_delay)
416                 error_counter += 1
417                 receive_string = self.server.recv(4096, socket.MSG_PEEK).decode("utf-8", 'ignore').replace("\r\n", " ")
418                 if receive_string.find("> >") > 0:
419                     _ = self.server.recv(4096)
420                     print(f"{bcolors.WARNING}ERESET robot complete{bcolors.ENDC}")
421                     self.close_connection()
422                     self.robot_is_busy = False
423                     return 1
424
425                 if error_counter > self.error_counter_limit:
426                     self.close_connection()
427                     self.robot_is_busy = False
428                     self.abort_operation = False
429                     print("Receive timeout is over")
430                     return -1000
431
432                 if self.abort_operation:
433                     self.close_connection()
434                     self.robot_is_busy = False
435                     self.abort_operation = False
436                     return -1000
437
        aorehov
438 def close_connection(self):
439     self.server.close()
```

Приложение 4 – Исходный код главной программы

```python
import pc_processing
import camera
import kawasaki_lib as kawa
import time
from math import pi, sin, cos, radians

VecXYZ = kawa.VecXYZ


 aorehov
def generate_circle(center: VecXYZ, radius, hover, resolution=12):
    points = []
    for i in range(resolution):
        angle = 2 * pi / resolution * i
        points.append(VecXYZ(round(center.x + radius * cos(angle), 2),
                             round(center.y + radius * sin(angle), 2),
                             center.z + hover))
    return points


IP = "192.168.1.220"
PORT = 23


if __name__ == "__main__":
    robot = kawa.RobotCommunication(IP, PORT)

    try:
        # raise TimeoutError
        robot.connect()
        robot.close_connection()
    except TimeoutError:
        print("unable to establish connection with robot")

        HEIGHT = 30
        lookat = [0, 0, 30]

        circle = generate_circle(VecXYZ(*lookat), 400, 250, resolution=12)
        program = [kawa.Position(point, lookat, rot=pi) for point in circle[:7]]
        circle = generate_circle(VecXYZ(*lookat), 350, 400, resolution=12)
        program.extend([kawa.Position(point, lookat, rot=pi) for point in circle[:7]])

        program = [[i + 1, program[i]] for i in range(len(program))]
        step = 1
        frame_size = 0
```

```python
        pc = pc_processing.Cloud(0, 640, 480, 3)

        for el in program:
            step, position = el

            print(f"step №{step} - {position}")
            pc.load_from_raw(f"files/{step}.raw")
            print("frame size: ", size := pc.process_frame(position.get_radians()))
            frame_size += size
            pc.save_frame()
            step += 1

        pc.save("pult_test.ply", overwrite=True)
        pc.clear()
    exit(0)

    HEIGHT = 30
    lookat = [0, 0, HEIGHT]
    points = [[-100, -100, HEIGHT + 300]]
    program = [kawa.Position(point, lookat, rot=pi) for point in points]

    circle = generate_circle(VecXYZ(*lookat), 400, 250, resolution=12)
    program = [kawa.Position(point, lookat, rot=radians(200)) for point in circle[:7]]
    circle = generate_circle(VecXYZ(*lookat), 350, 400, resolution=12)
    program.extend([kawa.Position(point, lookat, rot=pi) for point in circle[:7]])

    step = 1
    with camera.CameraHandler() as helios:
        helios.ready.wait()
        pc = pc_processing.Cloud(*helios.get_stream_data())

        for i in range(len(program)):
            try:
                position = program[i]
                print(f"step №{step} - {position}")
                robot.move_point(position)
                print("frame size: ", size := pc.process_frame(program[i].get_radians()))
                helios.new_frame.set()
                step += 1
            except KeyboardInterrupt:
                break
    pc.save("balansir.ply", overwrite=True)
    pc.clear()
```