Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана»
(национальный исследовательский университет)»(МГТУ им. Н.Э. Баумана)

Факультет «Робототехника и комплексная автоматизация»

Кафедра «Системы автоматизированного проектирования»

ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

ПО ТЕМЕ: «Разработка драйвера к системе стереозрения с использованием OpenCV, PCL и ROS»

Выполнил студент: Новокшанов Евгений Андреевич

фамилия, имя, отчество

Группа: РК6-46Б

Проверил: Козов А.В.

фамилия, имя, отчество

Оценка_____Дата_____

Москва, 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача формирования цифровой модели видимого объекта актуальна как в специализированных областях науки и техники, так и в повседневной жизни. Одним из направлений машинного зрения является стереозрение. Оно позволяет получить представление о глубине изображения и расстоянии до объектов, составить трехмерную картину окружающего мира.

На данный момент существует множество приборов для измерения и формирования 3D-изображения местности, однако стереозрение дает возможность обойтись без использования датчиков измерения расстояния: инфракрасных датчиков, звуковых локаторов или лазерных радаров. Это позволяет снизить стоимость технического решения.

Одним из примеров использования стереопары является проект NASA 2006 года под названием STEREO: два одинаковых космических аппарата наблюдают за Солнцем из двух разных точек, чтобы получать трехмерные изображения структур и явлений на Солнце

Обеспечение синхронности и точности работы стереокамер для обеспечения роботов зрительной информацией является основной проблемой, которым посвящена данная работа. Для этого необходимо подобрать параметры для камер, чтобы сохранить реальные объекты в цифровом виде: облако точек и карта смещений (карты глубины).

Цель: построение облака точек в ROS по данным карты смещения.

Задачи:

• разработка математического аппарата для создание трехмерной картины местности по существующей карте глубины;

• знакомство с библиотекой PCL и системой ROS;

• преобразование карты глубины в облако точек в системе ROS;

• подбор параметров для данной стереокамеры;

• уменьшение времени между обновлениями облака точек (усовершенствование работы в режиме «реального времени»).

1 Математическое описание метода формирования облака точек

На рисунке 1 показаны этапы формирования облака точек.

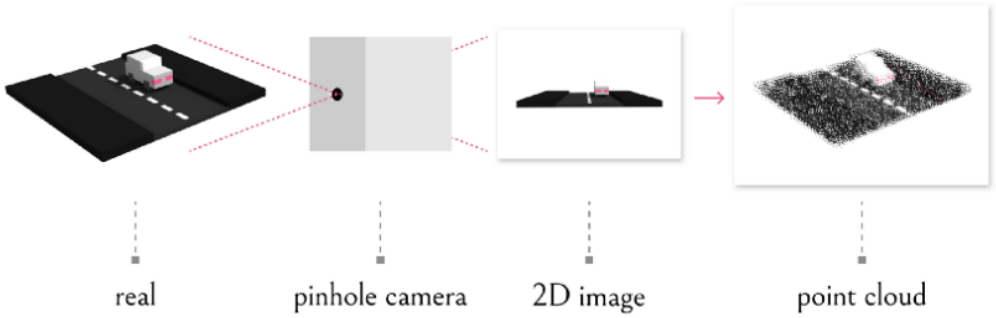

Рисунок 1 – Все этапы: от формирования карты смещений до формирования облака точек

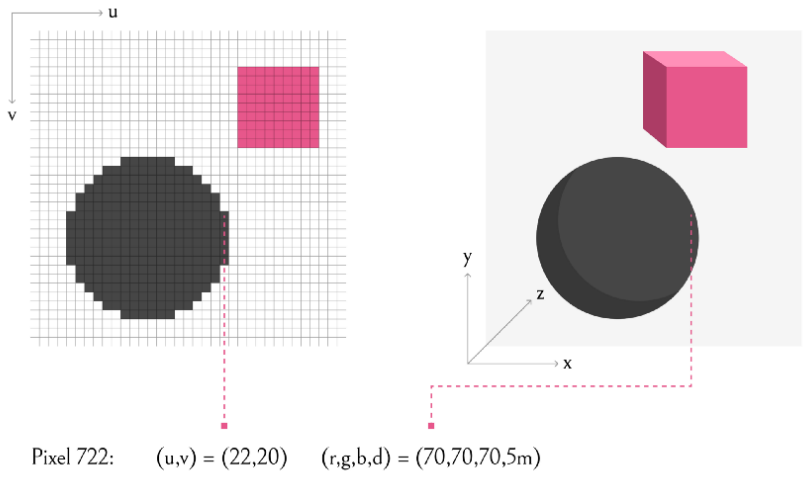

Рисунок 2 – Вводим оси u и v в карте смещений

Из подобных треугольников представленных на рисунке 2 выводим положение x из u и d каждого пикселя, что мы можем увидеть на 3-м рисунке. Для y и v аналогично. Для модели камеры-обскуры фокусное расстояние одинаково в направлениях x и y.

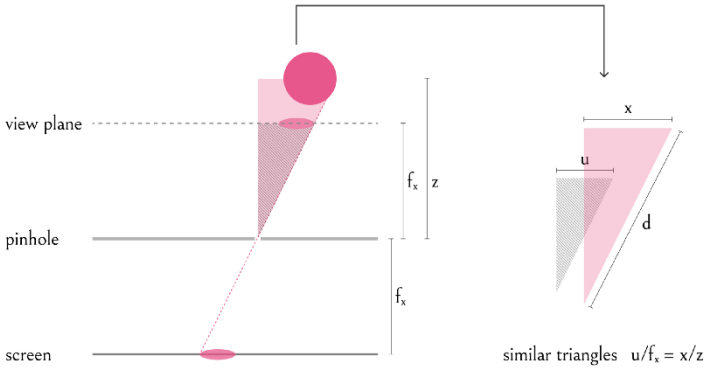

Рисунок 3 – Рассматриваем подобные треугольники

Из подобных треугольников получаем:

$$x = \frac{uz}{f_x}$$

Обычно f_x и f_γ идентичны.

Введем внутреннюю матрицу: единая матрица, которая включает в себя свойства камеры: фокусное расстояние и центр датчика камеры, и перекос.

$$\boldsymbol{K} = \begin{bmatrix} f_x & S & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Далее введем однородные координаты. Однородные координаты помогут нам записать преобразования (перемещения, повороты и наклоны) в виде матриц одинаковой размерности.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} 2u \\ 2v \\ 2 \end{bmatrix}$$

Теперь мы можем делать любые операции над однородными координатами. Все операции определены таким образом, что последний компонент остается неизменным.

Матрица вращения R, вектор переноса t и внутренняя матрица K составляют матрицу проекции камеры. Он определен для преобразования декартовых координат в координаты экрана:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} \underbrace{\boldsymbol{K}}_{3\times3} \underbrace{[\boldsymbol{R} \mid \boldsymbol{t}]}_{3\times4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

[R | t]: мы объединяем R и вектор-столбец t =transpose(t_0, t_1, t_2).

Мы не можем инвертировать матрицу 3x4. Матрицы 4x4 называются внутренними/внешними матрицами полного ранга. Поэтому дополняем матрицы до полного ранга, что мы можем увидеть далее.

$$\begin{bmatrix} u \\ v \\ 1 \\ 1/z \end{bmatrix} = \frac{1}{z} \underbrace{\begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}}_{4\times4} \underbrace{\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}}_{4\times4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Проверим сказанное выше на простейшем случае: начало отсчета камеры и начало мира выровнены, т.е. R и t можно пренебречь, перекос S равен 0, а датчик изображения отцентрован. Теперь обратная матрица камеры выглядит просто:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = z \underbrace{\begin{bmatrix} 1/f_x & 0 & 0 & 0 \\ 0 & 1/f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{inverse\ with\ c_x,c_y,S=0} \begin{bmatrix} u \\ v \\ 1 \\ 1/z \end{bmatrix}$$

Для стереопары с разными фокусами камер и другими погрешностями используем матрицу камеры такого вида:

$$\begin{bmatrix} 1/f_x & -S/\left(f_x f_y\right) & \left(Sc_y - c_x f_y\right)/\left(f_x f_y\right) \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{bmatrix}$$

Рассмотренные математические методы позволяют восстановить реальные объекты в цифровом виде, т.е. в виде облака точек.

2 Практическая реализация

2.1 Обзор библиотеки

Для программной реализации алгоритма была выбрана библиотека OpenCV, PCL и ROS.

OpenCV – библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков.

-Imgproc, features2d – обработка изображений (фильтрация, геометрические преобразования, преобразование цветовых пространств, сегментация, обнаружение особых точек и ребер, контурный анализ и др);

Рассматриваемая библиотека обладает широким набором инструментов для решения поставленной задачи и обработки изображений в целом. Например, модуль calib3d включает в себя методы (calibrateCamera, stereoRectify и др.) получения необходимых параметров для выравнивания пары изображений, а базовые функции библиотеки позволят подготовить стереопару к обработке. Помимо прочего, класс FileStorage обеспечивает удобную работу с XML / YAML / JSON файлами, что облегчает передачу данных между программами.

PCL (Библиотека облаков точек) – библиотека для работы с облаком точек, которая содержит множество современных алгоритмов, включая фильтрацию, оценку признаков, реконструкцию поверхности, регистрацию, подбор модели и сегментацию.

int savePCDFile () // Сохраните данные облака точек в файл PCD, содержащий точки nD.

pcl :: PCLPointCloud2 // Исходный формат данных облака точек

pcl_conversions::toPCL(*input, *cloud) //Конвертируем в формат данных облака точек в PCL

Процесс фильтрации

pcl :: VoxelGrid <pcl :: PCLPointCloud2> sor; // Создание экземпляра фильтрации

sor.setInputCloud (cloudPtr); // Устанавливаем входной фильтр

sor.setLeafSize (0.1, 0.1, 0.1); // Устанавливаем размер сетки вокселей

sor.filter (cloud_filtered); // сохраняем отфильтрованное облако точек

ROS (Robot Operating System) – это экосистема для программирования роботов, предоставляющая функциональность для распределённой работы.

Конвертация формата данных отфильтрованного облака точек в формат данных в ROS и публикация его

sensor_msgs :: PointCloud2 – Заявленный формат облака точек вывода

pcl_conversions :: fromPCL () – конвертация: Первый параметр – входной, второй – выходной

2.2 Алгоритм формирования облака точек

Задачу формирования облака точек целесообразно разбить на несколько этапов:

1) подготовка стереоустановки;
2) получение карты смещений;
3) перевод координат точек карты смещений в декартовые координаты облака точек;
4) настройка параметров камеры для получения облака точек близкого к реальной геометрии объекта.

Разберем каждый этап отдельно.

1. Этап подготовки стереоустановки представлен на блок-схеме (рисунок 8). Для успешного выполнения триангуляции нам необходима выровненная и измеренная стереоустановка (требуется провести калибровку и ректификацию), на этом этапе необходимо сформировать файл с ее параметрами для дальнейшего изменения изображений.

2. Получение карты смещений происходит в несколько этапов.

- Первым делом строим изображение с исправлением искажения камер.
- Далее программа строит изображение, где в каждом пикселе записывается расстояние, вычисленное исходя из разницы пикселей на изображениях, полученных с камер стереопары.
- На последнем этапе происходит настройка камеры с помощью программы calibration.cpp.

3. Перевод координат точек карты смещений в декартовые координаты облака точек происходит по алгоритму описанном в разделе «Математического описания»: мы для каждого пикселя(элемента матрицы) карты смещений рассчитываем декартовые координаты в системе отсчета облака точек. Сложность алгоритма O(n*m), где n и m – высота и ширина изображения карты смещений.

4. Исходя из полученных результатов прошлого пункта проводится подбор параметров для получения облака точек наиболее приближенного к реальному объекту сканирования.

Рисунок 4- блок-схема «Подготовки стереоустановки»

3 Оценка полученных результатов

Настройка стереокамеры и получение карты глубины

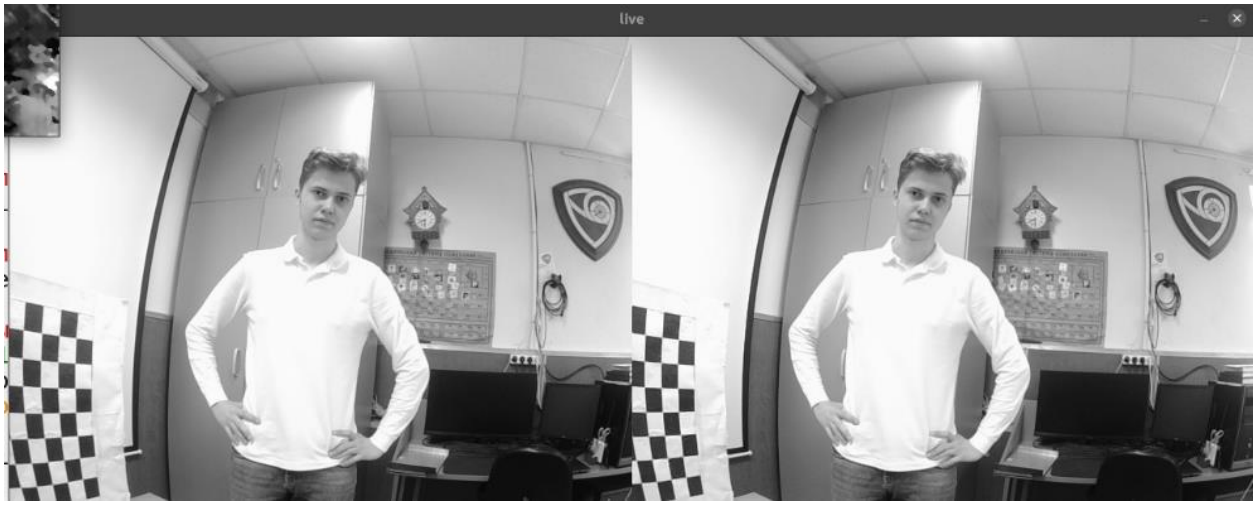

Рисунок 5 – Входное изображение

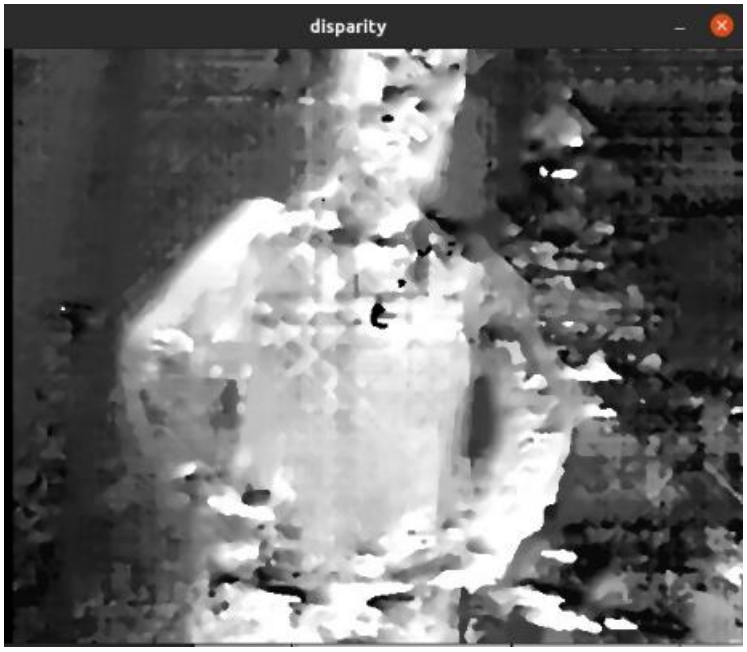

Рисунок 6 – Полученная карта смещений

Использование pcl_viewer для просмотра результата (облака точек)

Библиотека PCL предоставляет инструмент pcl_viewer для просмотра облаков точек, сохраненных в формате pcd.

Рисунок 7 – Входное изображение и карта глубины

Просмотр файла формата «pcd» представлен на рисунке 9 и 10

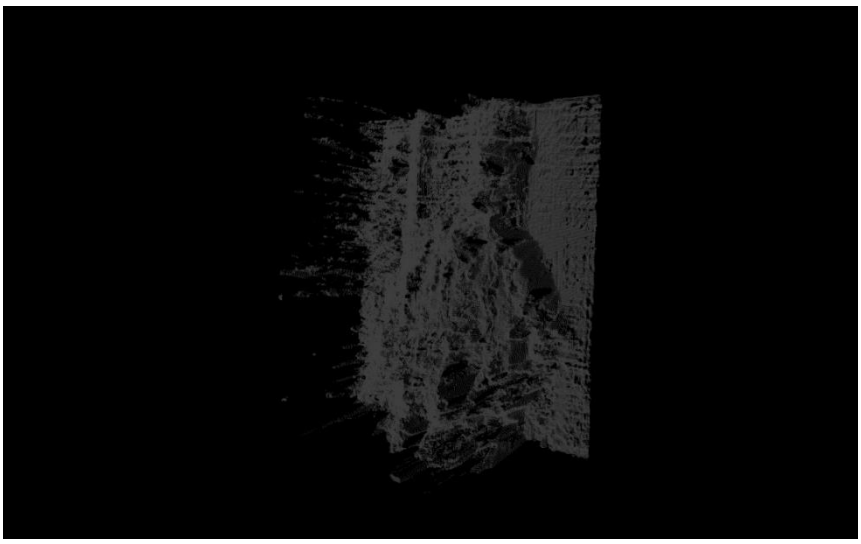

Рисунок 8 – Вид сбоку

Можем заметить, что отчетлива видна фигура человека, однако облако фона не соответствует реальным данным. Проводим настройку параметров в соответствии с полученными результатами.

Тест после настройки параметров можно увидеть на рисунках 11,12.13.

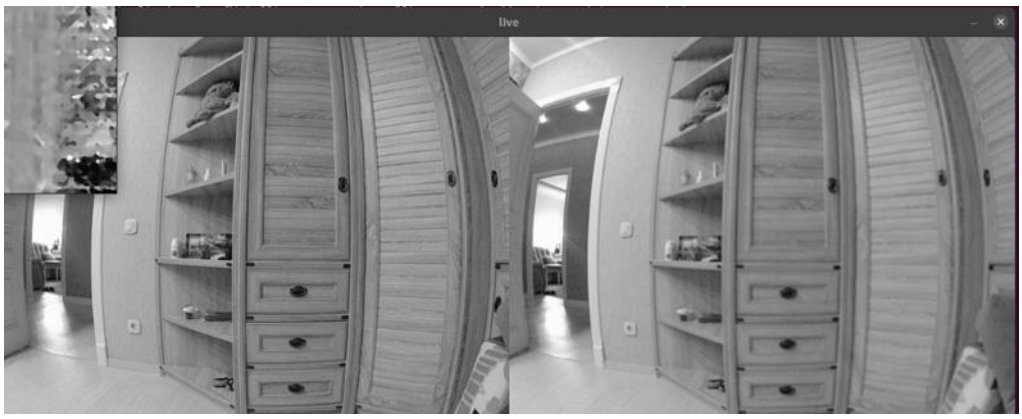

Рисунок 9 – входное изображение

Рисунок 10 – Карта смещений

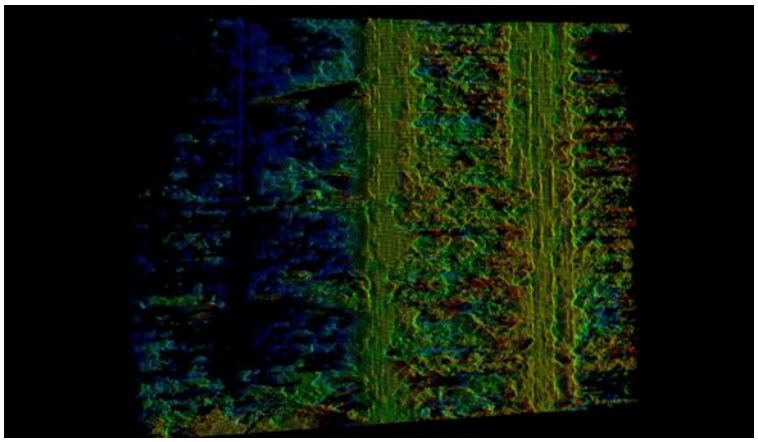

Рисунок 11 – Облако точек

Подобрав параметры, мы добились необходимой нам геометрической точности.

Просмотр полученного облака точек, адаптированного под ROS

Связав программу с узлом ROS передаем облако точек в ROS, которое мы можем увидеть с помощью утилиты rviz. На основе полученных результатов можно сказать, что алгоритм восстановления координат по карте смещений работает правильно, однако его точность во многом зависит от камер и карты смещений, что мы можем наблюдать на рисунке 12.

Рисунок 12 – Исходное фото

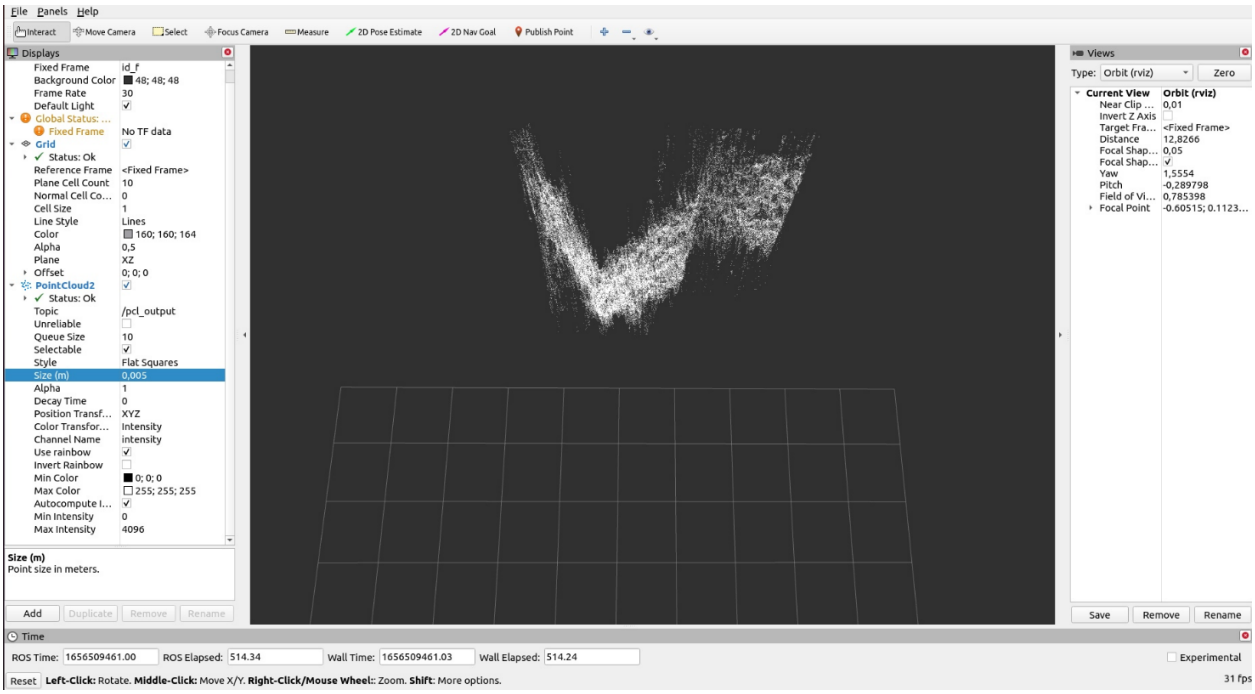

Рисунок 13 – Облако точек в программе rviz

ЗАКЛЮЧЕНИЕ

В ходе проведения НИРС были рассмотрены методы, позволяющие по существующей карте смещений восстановить объект в виде облака точек. На их основе был составлен и реализовал алгоритм на языке C++. Реализация была выполнены с использованием библиотек OpenCV, PCL и ROS. На основе полученных результатов сделана оценка работоспособности математической модели. Данную программу можно использовать как драйвер для подключения стереопары к ROS, что позволяет пользоваться камерой как полноценным зрительным источником 3-х мерной информации в дополнение к 2-х мерным изображениям для робота, дрона и в других областях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация OpenCV:сайт.URL:https://docs.opencv.org/master/d9/d0c/group__calib3d.html (дата обращения: 01.06.2021). – Текст: электронный.
2. From depth map to point: сайт.URL:cloudhttps://medium.com/yodayoda/from-depth-map-to-point-cloud-7473721d3f
3. сайт.URL:https://runebook.dev/ru/docs/point_cloud_library/structpcl_1_1_intensity#details
4. Wiki-ROS: сайт. URL: http://wiki.ros.org/pcl
5. Сайт. URL: https://russianblogs.com/article/82651189731/
6. Сайт. URL: https://russianblogs.com/article/97841343343/
7. От PCL к ROS: сайт. URL: http://wiki.ros.org/pcl_ros#ROS_C.2B-.2B-_interface
8. Стереозрение-Хабр: сайт. URL: https://habr.com/ru/post/130300/

ТЕКСТ ПРОГРАММЫ

```
 include <ros/ros.h>
#include <opencv2/opencv.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <stdio.h>
#include <iostream>
#include "opencv2/imgcodecs.hpp"
#include <opencv2/core/core_c.h>
#include <cmath>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>

#include <sensor_msgs/PointCloud2.h>
#include <pcl_conversions/pcl_conversions.h>
typedef pcl::PointXYZ PointT;
typedef pcl::PointCloud<PointT> PointCloud;
using namespace cv;
using namespace std;
Mat q;
int deviceID = 2;
int apiID = CAP_ANY;
const double camera_factor = 1000;
const double camera_cx = 256;
const double camera_cy = 212;
const double camera_fx = 1000;
const double camera_fy = 1000;


//PsP±CЪP°P±PsC,PєP° CЃPsP±C‹C,PëCЏ PSP°P¶P°C,PëCЏ PjC‹C€Pë
(PIC‹C‡PëCЃP»PµPSPëPµ CЪP°CЃCЃC,PsCЏPSPëCЏ)
void mouseEvent(int evt, int x, int y, int flags, void* param) {
    Mat* rgb = (Mat*)param;
    double rat = ((int)(*rgb).at<uchar>(y, x)), a = 0.879;
        rat = rat / 255;

    double f = q.at<double>(2, 3);
        double T = -1 / q.at<double>(1, 3);
    double k = (1 - pow(1 - rat, 4));

    if (evt == EVENT_LBUTTONDOWN) {
            cout << a * 100 * k * f * T / (int)(*rgb).at<uchar>(y, x) << " m"
<< endl;
    }
    }

//PїPsP»CѓC‡PµPSPëPµ PsC‡PµCЪPµPrPSPsPiPs PєP°PrCЪP°,
CѓCЃC,P°PSPsPIP»PµPSPSPsPiPs CЪP°P·CЪPµC€PµPSPëCЏ
Mat another(){
    Mat foto, gray;
    VideoCapture cap;
    cap.open(deviceID, apiID);

    cap.set(CAP_PROP_AUTOFOCUS, 0);
    cap.set(cv::CAP_PROP_FRAME_HEIGHT, 1000);
    cap.set(cv::CAP_PROP_FRAME_WIDTH, 1000);

    if (!cap.isOpened()) {
        cerr << "ERROR! Unable to open camera\n";
        exit(-1);
```

```
    }
    cap.read(foto);
    cvtColor(foto, gray, cv::COLOR_BGR2GRAY);
    return gray;
}


sensor_msgs::PointCloud2 object_msg;
int main(int argc, char *argv[])
{
        ros::init (argc, argv, "stereo_driver");
        ros::NodeHandle nh;
        ros::Publisher pcl_pub = nh.advertise<sensor_msgs::PointCloud2>
("pcl_output", 1);
        sensor_msgs::PointCloud2 output;
        PointCloud::Ptr cloud(new PointCloud);
    if (argc >= 2) { // РїСЂРѕРІРµСЂСЏРµРј РєРѕР»РёС‡РµСЃС‚РІРѕ
Р°СЂРіСѓРјРµРЅС‚РѕРІ,РѕРЅРѕ
                deviceID = atoi(argv[1]);
            }
    //РѕР±СЉСЏРІР»СЏРµРј Рё РёРЅРёС†РёР°Р»РёР·РёСЂСѓРµРј
РїР°СЂР°РјРµС‚СЂС‹,СЂРѕРІ СЃС,РµСЂРµРѕР±СЂР°Р·СѓСЋС‰
    Mat cameraMatrix[2], distCoeffs[2], R1, R2, P1, P2, Q;
    Size img_size;
    FileStorage fsp("parameters.yml", FileStorage::READ);
    fsp["cameraMatrixL"] >> cameraMatrix[0];
    fsp["distCoeffsL"] >> distCoeffs[0];
    fsp["RL"] >> R1;
    fsp["PL"] >> P1;
    fsp["cameraMatrixR"] >> cameraMatrix[1];
    fsp["distCoeffsR"] >> distCoeffs[1];
    fsp["RR"] >> R2;
    fsp["PR"] >> P2;
    fsp["Q"] >> Q;
    fsp["size"] >> img_size;

    q = Q;

    //СЃРѕР·РґР°РЅРёРµ РєР°СЂС, РїСЂРµРѕР±СЂР°Р·РѕРІР°РЅРёСЏ
    Mat map1[2], map2[2];
    initUndistortRectifyMap(cameraMatrix[0], distCoeffs[0], R1, P1, img_size,
CV_16SC2, map1[0], map2[0]);
    initUndistortRectifyMap(cameraMatrix[1], distCoeffs[1], R2, P2, img_size,
CV_16SC2, map1[1], map2[1]);


    Mat img, Left, Right, L, R;
    Mat disparity, d;

    //СЃРѕР·РґР°РЅРёРµ РѕР±СЉРµРєС‚Р° РєР»Р°СЃСЃР° StereoSGBM СЃ
Р·Р°СЂР°РЅРµРµ РІС‹С‡РёСЃР»РµРЅРЅС‹С‹ РїРё РїР°СЂР°РјРµС‚СЂС‹
    Ptr<cv::StereoSGBM> st = cv::StereoSGBM::create(1, 48, 3, 50, 1156, 55,
0, 1, 1, 59, cv::StereoSGBM::MODE_SGBM);

    //РЅРѕРѕРґРёРјС…РѕСЂРёРјРѕ РґР»СЏ РѕСЂРіР°РЅРёР·Р°С†РёРё СЃРѕС‹С‚С,РёРѕ СЃ
РјС‹С€Рё
    //namedWindow("disparity");
    //setMouseCallback("disparity", mouseEvent, &disparity);

    //РїРѕР»СѓС‡РµРЅРёРµ РєР°СЂС‚РёРЅРєРё РІ СЃРѕРѕС‰РёРЅРёРё РІ СЂРµР¶РёРјРµ
СЂРµР°Р»СЊРЅРѕРіРѕ РІСЂРµРјРµРЅРё СЃ РІРѕР·РјРѕР¶РЅРѕСЃС‚СЊСЋ
СЂР°СЃСЃС‡РµС,Р° СЂР°СЃСЃ,РѕСЏРЅРёСЏ РґРѕ РѕР±СЉРµРєС,Р° РїСЂРё
РѕР¶Р°С,РёРё РѕР° РѕРѕРµРіРѕ РјС‹С€СЊ
```

```
    while (ros::ok())
    {

        img = another();
        Mat l(img, cv::Rect(0, 0, img.cols / 2, img.rows));
        Mat r(img, cv::Rect(img.cols / 2, 0, img.cols / 2, img.rows));
        l.copyTo(Left);
        r.copyTo(Right);
        remap(Left, L, map1[0], map2[0], INTER_LINEAR, BORDER_CONSTANT,
cv::Scalar(0, 0, 0));
        remap(Right, R, map1[1], map2[1], INTER_LINEAR, BORDER_CONSTANT,
cv::Scalar(0, 0, 0));
        medianBlur(L, L, 5);
        medianBlur(R, R, 5);
        st->compute(L, R, d);
        d.convertTo(disparity, CV_8U, 255 / (20 * 20.));
        medianBlur(disparity, disparity, 5);
        imshow("disparity", disparity);
        imshow("live", img);

        // PïCЂPµPsP±CЂP°P·PsPIP°PSPëPµ PjP°C,CЂPëC†C‹ PI PsP±P»P°PєPs
        /*for (int m = 0; m < d.rows; m++)
                for (int n = 0; n < d.cols; n++)
                {
                        //cout <<
                         // PµPsP»CґC‡P°PµPj P·PSP°C‡PµPSPëPµ PI C,PsC‡PєPµ (m,
n) PSP° PєP°CЂC,Pµ PiP»CґP±PëPSC‹
                        ushort d_pcl = d.ptr<ushort>(m)[n];
                         // d PjPsP¶PµC, PSPµ PëPjPµC,CЊ P·PSP°C‡PµPSPëCЏ,
PµCЃP»Pë PґP°, PïCЂPsPïCґCЃC,PëC,Pµ CЌC,PsC, PïCґPSPєC,
                        if (d_pcl == 0)
                                continue;
                         // P•CЃP»Pë d PëPjPµPµC, P·PSP°C‡PµPSPëPµ,
PґPsP±P°PIP»CЏPµPj C,PsC‡PєCґ PI PsP±P»P°PєPs C,PsC‡PµPє


                         // P’C‹C‡PëCЃP»PëC,CЊ PïCЂPsCЃC,CЂP°PSCЃC,PIPµPSPSC‹Pµ
PєPsPsCЂPґPëPSP°C,C‹ CЌC,PsP№ C,PsC‡PєPë
                        //cout<< double(d_pcl) << endl;

                }*/
        for (int m = 0; m < d.rows; m++)
                for (int n = 0; n < d.cols; n++)
                {
                        //cout <<
                         // PµPsP»CґC‡P°PµPj P·PSP°C‡PµPSPëPµ PI C,PsC‡PєPµ (m,
n) PSP° PєP°CЂC,Pµ PiP»CґP±PëPSC‹
                        ushort d_pcl = d.ptr<ushort>(m)[n];
                         // d PjPsP¶PµC, PSPµ PëPjPµC,CЊ P·PSP°C‡PµPSPëCЏ,
PµCЃP»Pë PґP°, PïCЂPsPïCґCЃC,PëC,Pµ CЌC,PsC, PïCґPSPєC,
                        if (d_pcl == 0)
                                continue;
                         // P•CЃP»Pë d PëPjPµPµC, P·PSP°C‡PµPSPëPµ,
PґPsP±P°PIP»CЏPµPj C,PsC‡PєCґ PI PsP±P»P°PєPs C,PsC‡PµPє
                        PointT p;
                        //if(m!=0 && n !=0 && m!=d.rows && n != d.cols)
                                //d_pcl = (d.ptr<ushort>(m-1)[n] +
d.ptr<ushort>(m+1)[n] + d.ptr<ushort>(m)[n-1] + d.ptr<ushort>(m)[n+1])/4;
                         // P’C‹C‡PëCЃP»PëC,CЊ PïCЂPsCЃC,CЂP°PSCЃC,PIPµPSPSC‹Pµ
PєPsPsCЂPґPëPSP°C,C‹ CЌC,PsP№ C,PsC‡PєPë
                        //cout<< double(d_pcl) << endl;
                        if(d_pcl > 80 and d_pcl < 350){
                        p.z = (10-10*double(d_pcl) / (camera_factor));
```

```
                        p.x = -(n - camera_cx) * p.z / camera_fx;
                        p.y = -(m - camera_cy) * p.z / camera_fy;
                         // PµPsP»CґC‡P°PµPj PµPiPs C†PIPµC, PëP·
PëP·PsP±CЂP°P¶PµPSPëCЏ rgb
                         // rgb - CЌC,Ps C,CЂPµC…PєP°PSP°P»CЊPSPsPµ
PëP·PsP±CЂP°P¶PµPSPëPµ PI C„PsCЂPjP°C,Pµ BGR, PïPsCЌC,PsPjCґ
PïPsP»CґC‡P°P№C,Pµ C†PIPµC,P° PI CЃP»PµPґCґCЂC‰PµPj PïPsCЂCЏPґPєPµ
                        //p.b = 999999;
                        //p.g = 999999;
                        //p.r = 999999;

                         // PґPsP±P°PIP»CЏPµPj p PI PsP±P»P°PєPs C,PsC‡PµPє
                        cloud->points.push_back(p);}
                }/**/

         //P·P°PIPµCЂC€PµPSPëPµ PïCЂPë PSP°P¶P°C,PëPë Esc
        // if (cv::waitKey(1) == 27) break;
         //sensor_msgs::PointCloud2 output;
         //output = cloud;

          // PћC‡PëCЃC,PëC,CЊ PґP°PSPSC‹Pµ Pë PIC‹P№C,Pë
          cloud->height = 1;
        cloud->width = cloud->points.size();
        cout << "point cloud size = " << cloud->points.size() << endl;
        cloud->is_dense = false;
        pcl::io::savePCDFile("222.pcd", *cloud);


        cout << "Point cloud saved." << endl;
        //cloud.reset(new pcl::PointCloud);
        pcl::toROSMsg(*cloud.get(),output );
        output.header.frame_id = "id_f";
        ros::Rate loop_rate(1);
        pcl_pub.publish(output);
        ros::spinOnce();
        loop_rate.sleep();
        //ros::Subscriber sub = nh.subscribe ("input", 1,cloud_cb);
        //pub = nh.advertise<sensor_msgs::PointCloud2> ("output", 1);


        cloud->points.clear();

  // Spin
        //ros::spin ();
    }

    return 0;
}
```