МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

УТВЕРЖДАЮ

Т.Т. Идиатуллов
Заведующий кафедрой «СМАРТ-технологии»

_____ "____" июнь 2020 г.
подпись

ОТЧЕТ

по курсовому проекту: Системы технического зрения и обработки изображений

Индивидуальный вариант № 1.3: Система анализа корректности считанной графической схемы на основании известного набора схем-шаблонов

Вид отчета: заключительный

Исполнитель

Студент группы 171-311 / Недов Андрей Владимирович

Научный руководитель

Кандидат физико-математических наук _____ Т.Т. Идиатуллов

Москва, 2020

Содержание

Введение

Разработка системы визуализации совмещённых данных сенсорики мобильного робота с возможностью управления направлением обзора. Выполняется в рамках курсового проектирования по дисциплине « Технологии визуализации данных систем управления».

Основная цель курсового проектирования – освоение методов проектирования систем визуализации данных, подготовки отчетной документации по научно исследовательской и опытно-конструкторской работе, а также изучение основ проектирование программного обеспечения с использованием библиотеки OpenGL. Выбор данной библиотеки определяется широкой распространенностью ее компонентов, доступностью для освоения, а также поставленной задачей – разработать программу которая на основе полученного набора данных производит визуализацию параметров в объёмном пространстве. В данной системе предусмотрена возможность настройки расположения всех сеносров робота (координаты, пово-рот, склонение).

Система проектирования приложений Visual Studio. Данный продукт позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом**.**

Библиотека OpenGL — спецификация, определяющая платформонезависимый (независимый от языка программирования) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

В рамках проведения курсового проектирования и подготовки отчета последовательно выполняются все этапы, необходимы для проведения разработки:

- анализ первичного варианта индивидуального задания;

– проектирование функциональной модели проекта;
– проектирование алгоритмов обработки данных;
– непосредственное кодирование алгоритмов на языке C#;
– отладку приложения;
– подготовки отчета.

Данный документ является отчетом по курсовому проектированию и содержит описание этапов разработки, а также всю документацию по проведенной работе и необходимые иллюстрации. Текст отчета подготовлен в соответствии с ГОСТ 7.32 – 2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления.

1. Задачи этапа исследования, описание выбранной общей методики проведения

Общей целью работы является разработка системы визуализации данных сенсорики робота. Общая функциональность системы определяется техническим заданием, разработанным в соответствии с индивидуальным заданием на курсовое проектирование.

В рамках проведения работ должны быть решены следующие задачи:

- провести анализ систем технического зрения и провести разработку функциональной модели;
-
- подготовить документацию по выполнению работы и работы с заданной программой.

На основе результатов анализа и моделирования сформировано описание структуры данных и алгоритмов их обработки. Затем, на базе подготовленного описания выполнена разработка программного кода проекта. Проведены работы по отладке и анализ целесообразности ее развития (по результатам испытания созданного прототипа).

2. Техническое задание

Разработать систему, отображающую параметры работы робота с использованием средств дополненной реальности. Сделать возможность ручной установки расположения и склонения сенсоров относительно системы координат робота. Также реализовать возможность создания скриншотов 3D сцены, а также её запись в видеофайл.

Порядок обработки:

- определить пространственную схему размещения сенсорных систем на роботе;
- выполнить покадровое считывание изображений с камер;
- выполнить получение данных с сенсоров; · выполнить получение облака точек с 3D-лидара;
- разработать систему пространственной визуализации данных с управлением позиции просмотра с клавиатуры и мыши
- реализовать нанесение мониторинговых данных поверх изображения с учетом схемы размещения виджетов;
- реализовать показ изображений и сохранения их в виде видеопоследовательности (видеофайла).

3. Проектирование

На этапе проектирования было определено количество сенсоров робота. Лидар, камера, и два сонара. Таким образом на 3D сцене отрисовываются схематичные изображения всех вышеперичисленных сенсоров, а так же сам робот.

В ходе разработки на 3D сцену также был добавлен куб отсечения, отсекающий лишние точки получаемые лидара от интересующего на облака точек.

Для робота и каждого его сенсора предусмотрена возможность устанавливать позицию в пространстве независимо от остальных объектов.

Проекция с камеры представлена серым полигоном в пространстве, лидар - тёмно-серым параллелепипедом, сонары - двумя фиолетовыми фигурами и, соответственно, робот представлен зелёным кубом.

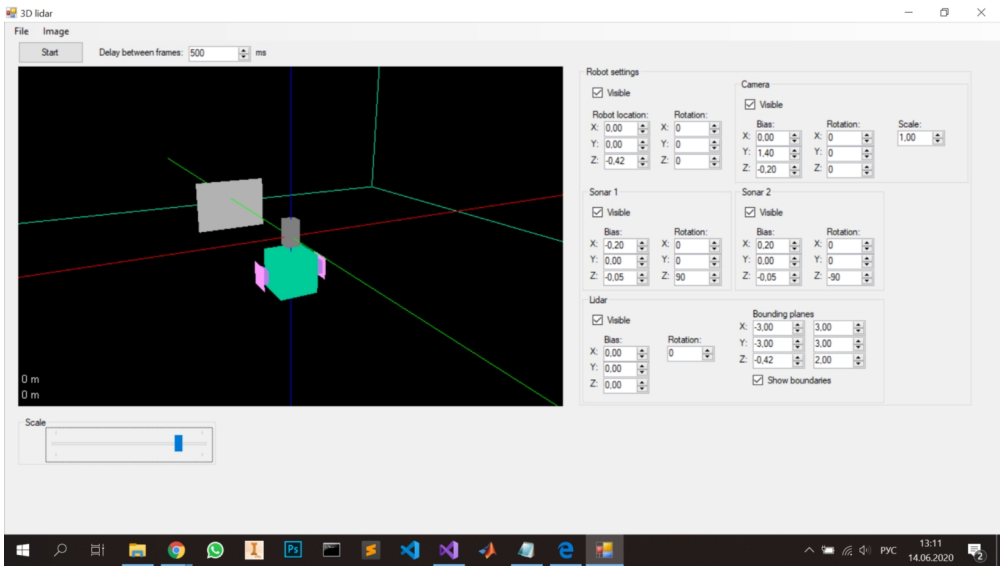

Рисунок 1 - Схематичное изображение робота и датчиков

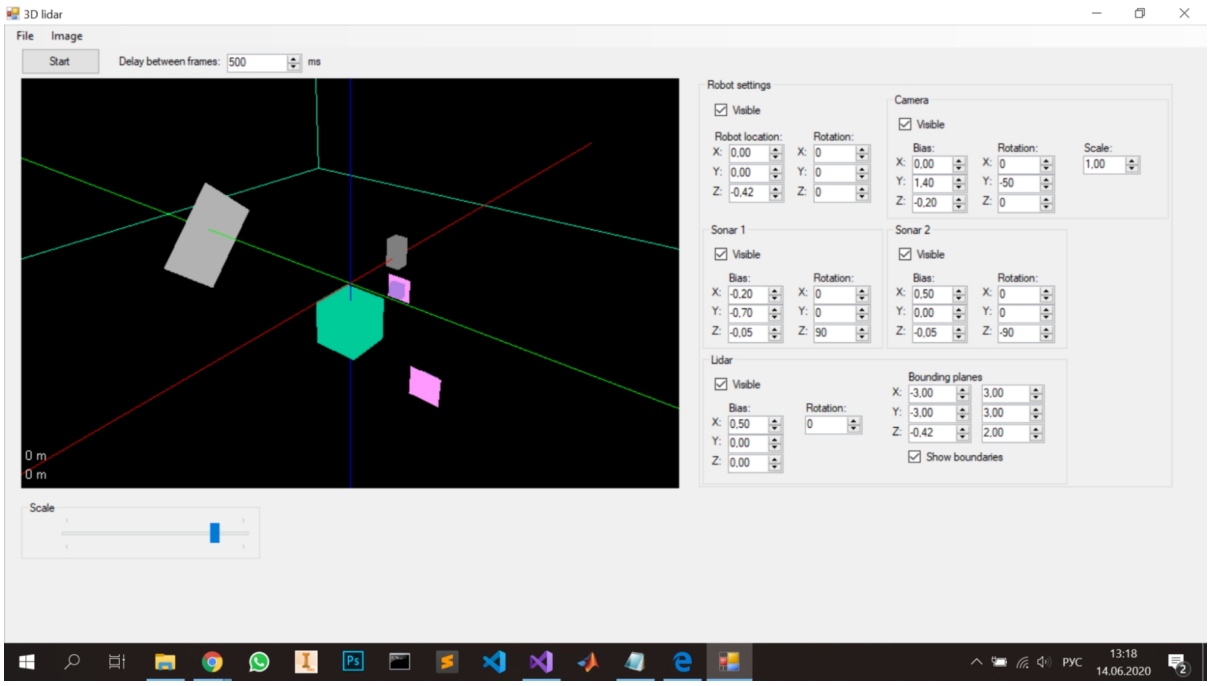

Рисунок 2 – Изменение расположения объектов

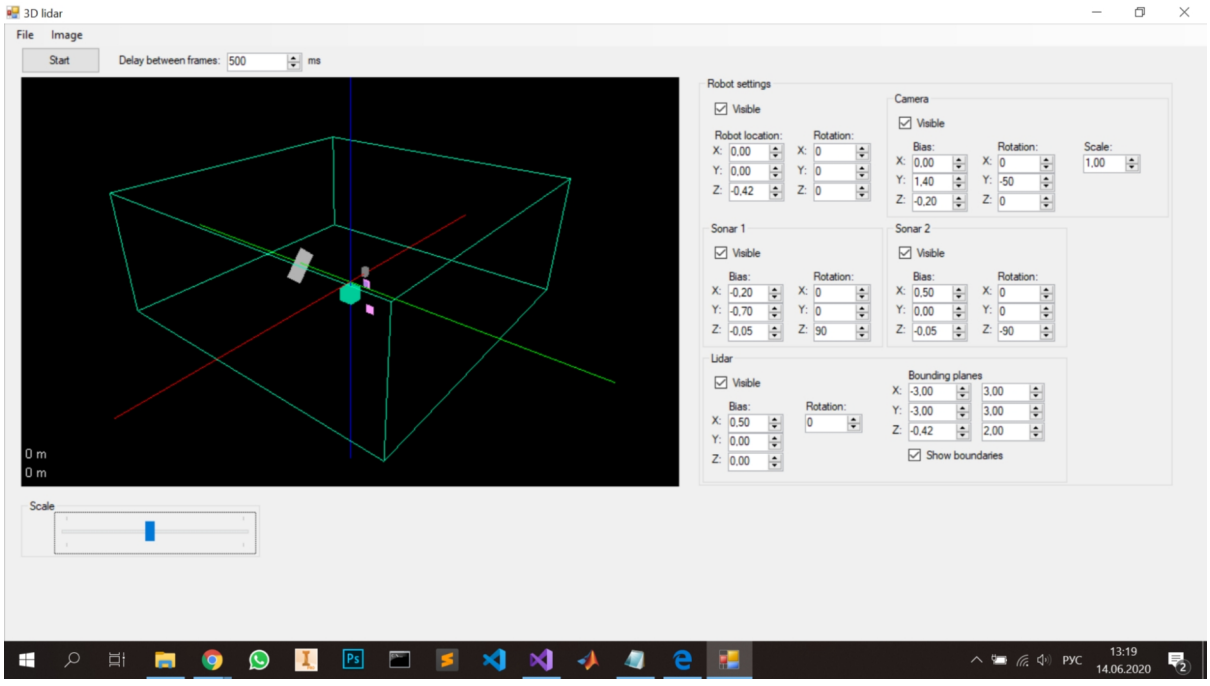

Рисунок 3 – Куб отсечения

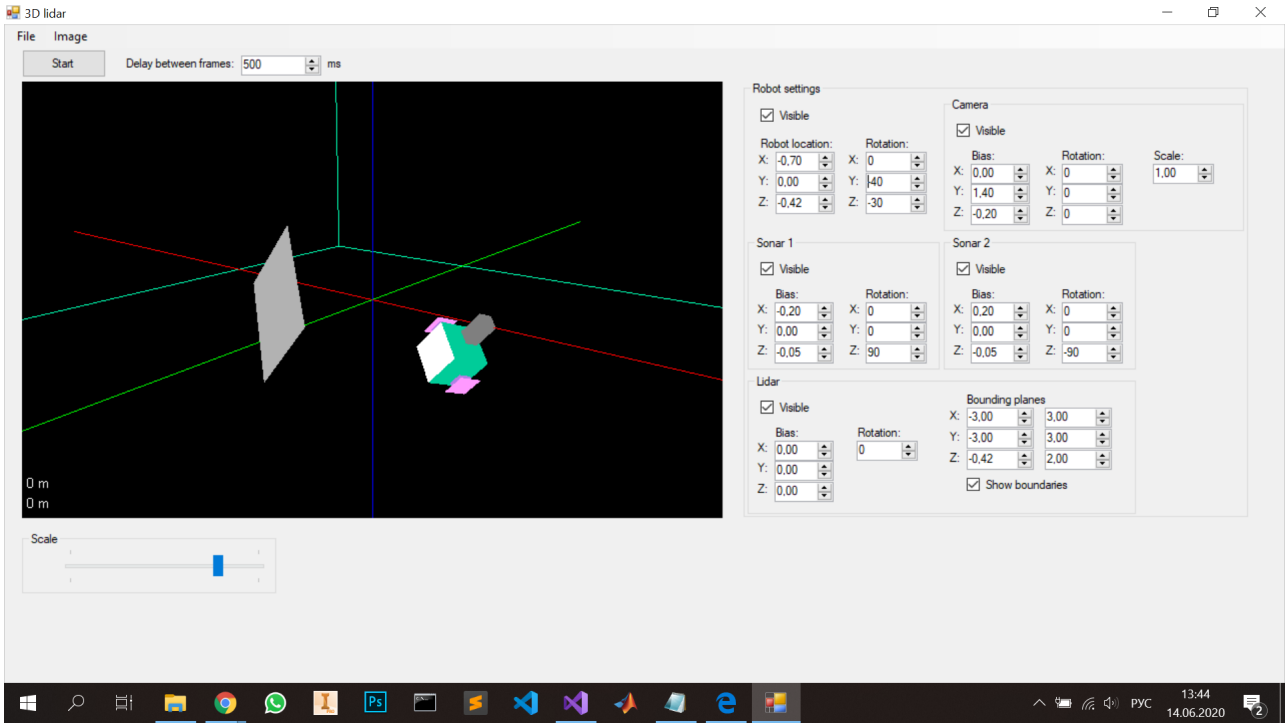

Рисунок 4 – Повёрнутый робот

Все датчики были помещены в систему координат робота. Таким образом при повороте или перемещении робота все датчики последуют за ним.

4. Обработка данных

Для объединения разных типов данных в одной системе были разработаны несколько модулей предназначенных для чтения разных типов данных. Так, один модуль отвечает за чтение данных с лидара, другой по разработанному заранее протоколу читает данные, приходящие от сонаров.

Ввиду эпидемии коронавируса доступ к лидару закрыт, поэтому чтение облака точек происходило из заранее записанного лидаром файла. Код системы приведён в приложении А.

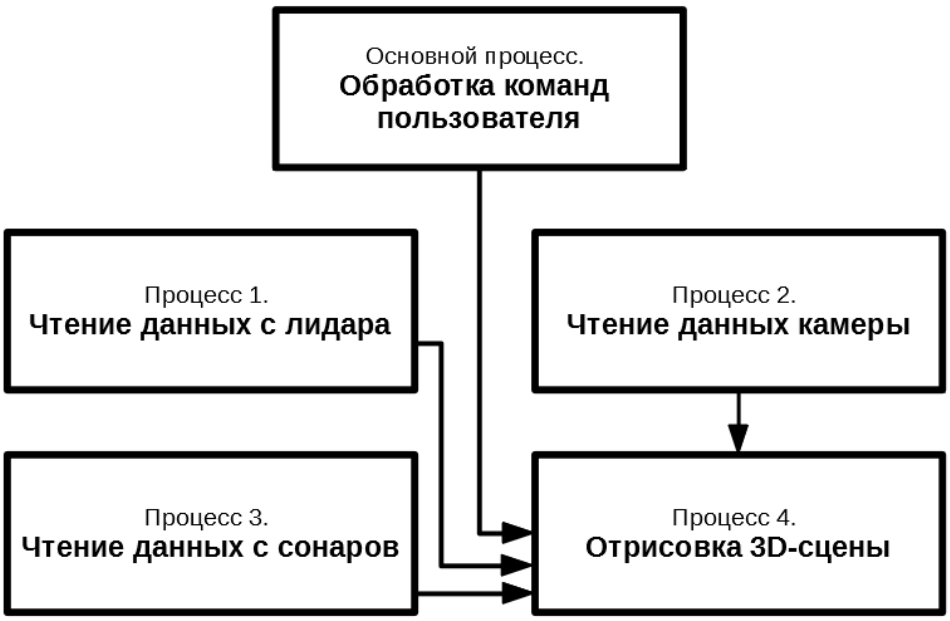

Рисунок 5 – Архитектура системы

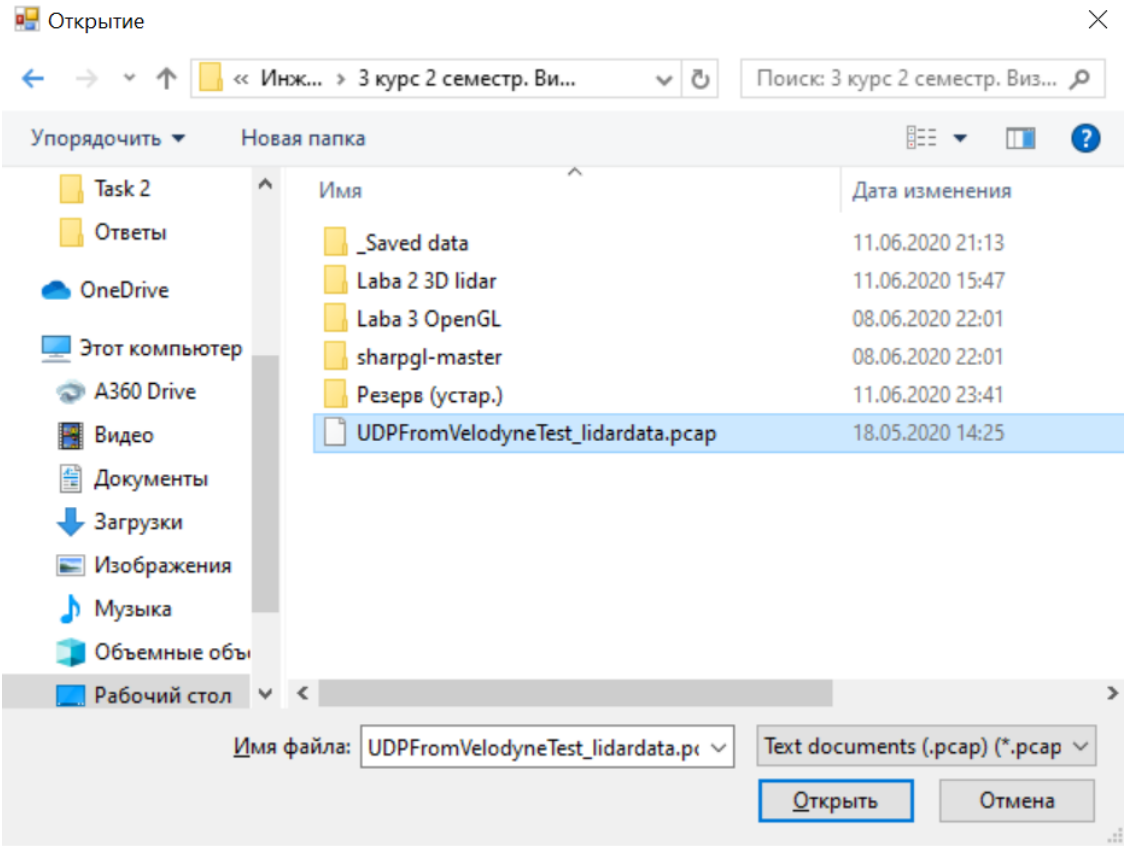

Рисунок 6 – Выбор записанного лидаром файла

Далее все данные тем или иным образом отодражаются в 3D пространстве.

Симуляция интерактивна, позволяет как управлять положением сцены, так и отулючать видимость ненужных сенсоров.

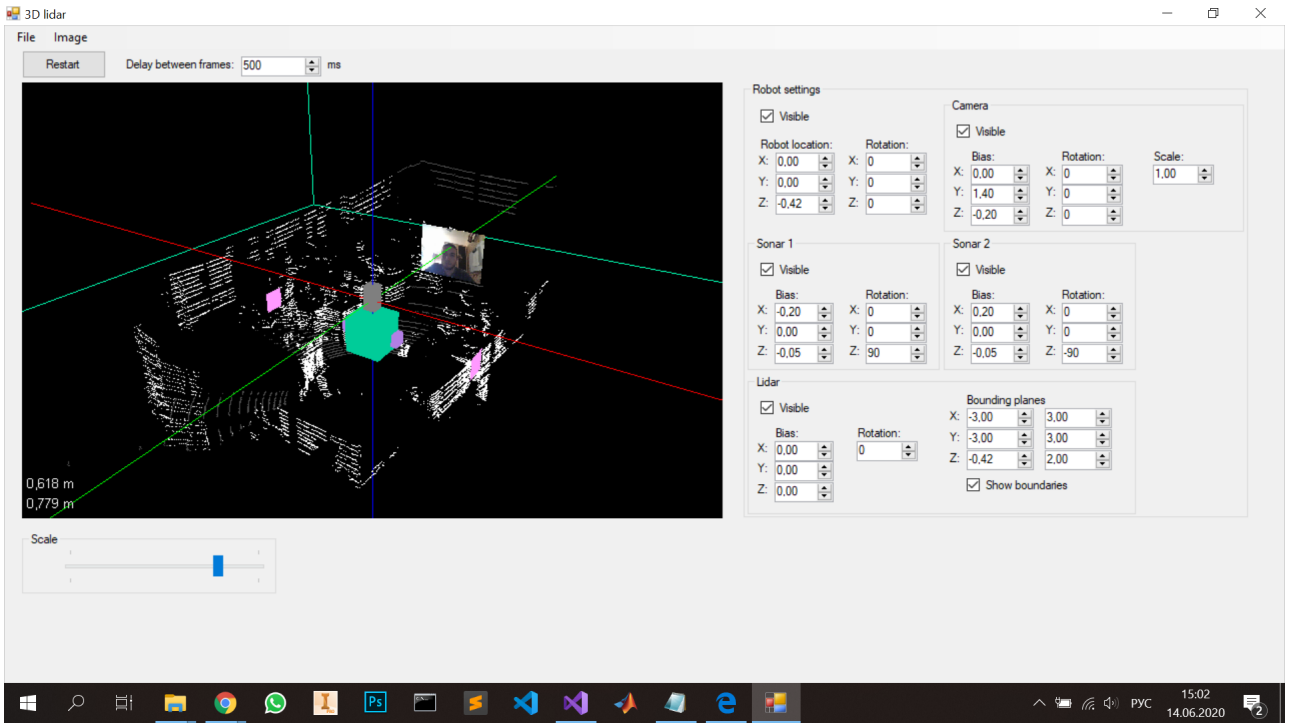

Рисунок 7 – Визуализация данных

Также была реализована возможность сохранять рендеры 3D сцены в виде графических изображений и сохранять видеозапись происходящего.

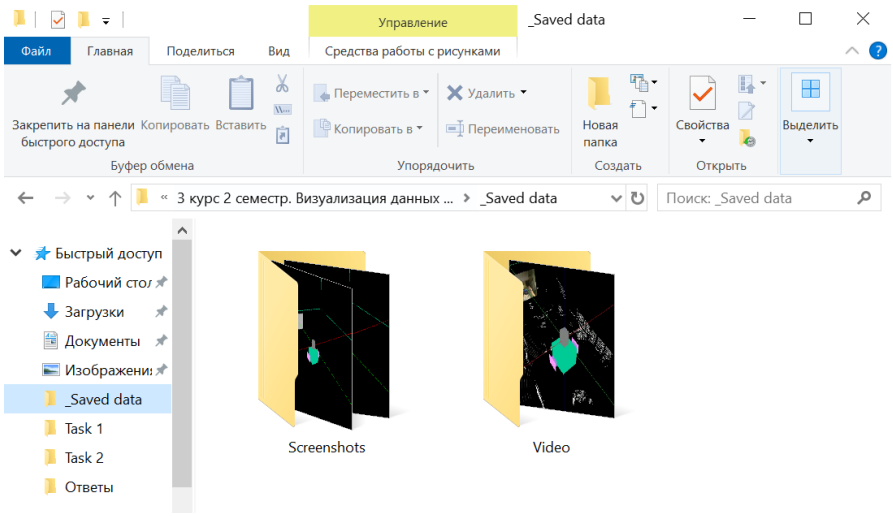

Рисунок 8 – Директория с сохраненными записями экрана

5. Рекомендации по использованию проекта

Данная система может быть установленна на любого робота с достаточно мощьным компьютером. Также возможна передача данных с сенсоров на отдельный ПК и дальнейшая визуализация этих данных уже на стороннем пк. Такой подход позвоялет значительно снизить на компьютер робота.

Заключение

В ходе выполнения работ по курсовому проектированию был разработан прототип системы визуализации данных с сенсорики робота. Также была разработана функциональная модель и написана документация с рекомендациями.

Основная область применения результатов работы – реализация на производстве и в его автоматизации. Использование результатов разработки позволит более точно и быстро проводить отладку роботизированной системы, а так же предоставлять наглядную картину состояния сенсоров робота.

Так как разработка велась с целью прототипирования полноценной системы, то уникальность разработки обеспечивается только индивидуализацией информационной базой и отчетных форм для построение масштабной системы потребуется более подробный анализ аналогичных решений от сторонних производителей.

Список использованных источников

1. ГОСТ 7.32 – 2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления.

2. ГОСТ 2.106-96 Единая система конструкторской документации (ЕСКД). Текстовые документы.

Приложение А

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Threading;
using SharpGL;
using System.Timers;
using OpenCvSharp;
using OpenCvSharp.Extensions;
using System.IO.Ports;
using Accord.Video.FFMPEG;
//using Tao.FreeGLut;
//using SharpGL.SceneGraph.Assets;

namespace Laba_3_OpenGL
{
    public partial class scale : Form
    {
        string pathVar = "C:\\Users\\andre\\Desktop\\Учёба\\Политех\\Инженерные проекты\\3 курс 2 семестр. Визуализация данных с датчиков робота\\UDPFromVelodyneTest_lidardata.pcap";
        string pathVideo = "C:\\Users\\andre\\Desktop\\Учёба\\Политех\\Инженерные проекты\\3 курс 2 семестр. Визуализация данных с датчиков робота\\_Saved data\\Video\\_cashe";
        string pathVideoResult = "C:\\Users\\andre\\Desktop\\Учёба\\Политех\\Инженерные проекты\\3 курс 2 семестр. Визуализация данных с датчиков робота\\_Saved data\\Video\\";
        string pathScreen = "C:\\Users\\andre\\Desktop\\Учёба\\Политех\\Инженерные проекты\\3 курс 2 семестр. Визуализация данных с датчиков робота\\_Saved data\\Screenshots";

        double[] tiltAngle = new double[]
        {
            -30.67,
            -9.33,
            -29.33,
            -8.00,
            -28.00,
            -6.66,
            -26.66,
            -5.33,
            -25.33,
            -4.00,
            -24.00,
            -2.67,
            -22.67,
            -1.33,
            -21.33,
            0.00,
            -20.00,
            1.33,
            -18.67,
            2.67,
            -17.33,
            4.00,
            -16.00,
            5.33,
            -14.67,
            6.67,
            -13.33,
            8.00,
            -12.00,
            9.33,
            -10.67,
            10.67,
        };

        int imageCounter = 1;
        int screenshotNumber = 0;
        int videoNumber = 0;
        bool goRecord = false;
        bool needAScreenshot = false;

        private Thread camera;
        Mat frame_in;
        Mat frame_out;
        Mat frame_draw;
        VideoCapture capture;
        bool cameraOpened = false;

        int xSave = 682;
        int ySave = 434;

        Bitmap cameraImg;

        bool readed_once = false;

        bool drawn = true;
        bool busy = false;
        bool eyeBusy = false;

        double global_scale = 1.3;
```

```
double sonar1Dist = 0;
double sonar2Dist = 0;

double l_path = 4;

double[,] bp = new double[8, 3];

double xL;
double xR;
double yL;
double yR;
double zL;
double zR;

List<byte> bufText = new List<byte>();
List<byte> bufShow = new List<byte>();

double[] tiltAngleSin = new double[32];
double[] tiltAngleCos = new double[32];

RenderEventArgs args;
OpenGL gl;

List<double[]> cloudEx = new List<double[]>();
double[,] laserHistory = new double[32, 2];
int[] laserHistoryIndex = new int[32];
List<double[]> cloudIN = new List<double[]>();
List<double[]> cloudOUT = new List<double[]>();
List<double[]> cloudINShow = new List<double[]>();
List<double[]> cloudOUTShow = new List<double[]>();


int middle_index = -1;

List<double[]> bound_points = new List<double[]>();
List<double[]> bps = new List<double[]>();
double[] robot_space = new double[3];

List<string> text = new List<string>();

string str = "";
byte prev = 0;

bool mouse_down = false;

int oldValueX;
int oldValueY;

int angleX = 0;
int angleY = 0;
int angleZ = 0;

bool firstRead = true;

Thread rosa;
Thread memosa;
Thread save;

SerialPort port;
string[] ports;

byte[] imageData;
byte[] imageData2;

public scale()
{
    InitializeComponent();
}

private void Form1_Load_1(object sender, EventArgs e)
{
    ports = SerialPort.GetPortNames();

    frame_in = new Mat();
    frame_out = new Mat();
    frame_draw = new Mat();

    camera = new Thread(new ThreadStart(CaptureCameraCallback));

    System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = false;

    BoundariesCountT();
    for (int i = 0; i < 32; i++)
    {
        double radAlfa = tiltAngle[i] * Math.PI / 180.0;

        tiltAngleSin[i] = Math.Sin(radAlfa);
        tiltAngleCos[i] = Math.Cos(radAlfa);

        laserHistory[i, 0] = -10000;
        laserHistory[i, 1] = -10000;

        laserHistoryIndex[i] = -1;
    }

    string[] aragog = pathVar.Split('\\');
    //labelPath.Text = aragog[aragog.Length - 1];
```

```
            r_z.Value = zlN.Value;

            openGLControl1_OpenGLDraw(sender, args);
        }

        private void CreateMovie()
        {
            int width = xSave;
            int height = ySave;
            var framRate = 2;

            // create instance of video writer
            using (var vFWriter = new VideoFileWriter())
            {
                // create new video file
                vFWriter.Open(pathVideoResult + "\\Video" + videoNumber + ".avi", width, height, framRate,
VideoCodec.MPEG4);

                string[] picList = Directory.GetFiles(pathVideo, "*");

                //loop throught all images in the collection
                foreach (var pic in picList)
                {
                    Bitmap bitmap = new Bitmap(pic);
                    vFWriter.WriteVideoFrame(bitmap);
                }

                vFWriter.Close();
            }
        }

        private void CaptureCameraCallback()
        {
            capture = new VideoCapture(0); // (0) -встроенная, (1) - первая подключенная
            capture.Open(0);

            if (capture.IsOpened())
            {
                try
                {
                    while (cameraOpened == true) //если камера запущена
                    {
                        if (drawn)
                        {
                            capture.Read(frame_in);

                            //frame_out = frame_in;

                            frame_out = frame_in.CvtColor(ColorConversionCodes.BGR2RGB);

                            double scale = (double)screenScale.Value;

                            double fr_h = frame_out.Size().Height * scale;
                            double fr_w = frame_out.Size().Width * scale;

                            Cv2.Resize(frame_out, frame_out, new OpenCvSharp.Size(fr_w, fr_h));

                            eyeBusy = true;

                            frame_draw = frame_out.Clone();
                            drawn = false;

                            readed_once = true;

                            eyeBusy = false;
                        }
                    }
                }
                catch { }
            }

        }

        private void BoundariesCountT()
        {
            xL = Convert.ToDouble(xlN.Value);
            xR = Convert.ToDouble(xrN.Value);
            yL = Convert.ToDouble(ylN.Value);
            yR = Convert.ToDouble(yrN.Value);
            zL = Convert.ToDouble(zlN.Value);
            zR = Convert.ToDouble(zrN.Value);

            bp[0, 0] = xR; bp[0, 1] = yL; bp[0, 2] = zR;
            bp[1, 0] = xL; bp[1, 1] = yL; bp[1, 2] = zR;
            bp[2, 0] = xL; bp[2, 1] = yL; bp[2, 2] = zL;
            bp[3, 0] = xR; bp[3, 1] = yL; bp[3, 2] = zL;
            bp[4, 0] = xR; bp[4, 1] = yR; bp[4, 2] = zR;
            bp[5, 0] = xL; bp[5, 1] = yR; bp[5, 2] = zR;
            bp[6, 0] = xL; bp[6, 1] = yR; bp[6, 2] = zL;
            bp[7, 0] = xR; bp[7, 1] = yR; bp[7, 2] = zL;
        }

        private void Load_file(object sender, EventArgs e)
        {
            System.Windows.Forms.OpenFileDialog dlg = new System.Windows.Forms.OpenFileDialog();
            dlg.FileName = "Document";
            dlg.DefaultExt = ".pcap";
            dlg.Filter = "Text documents (.pcap)|*.pcap";
```

```
    DialogResult result = dlg.ShowDialog();

    pathVar = dlg.FileName;

    //string[] aragog = pathVar.Split('\\');
    //labelPath.Text = aragog[aragog.Length - 1];
}

private double[] MR(double x, double y, double z, double alfa_v)
{
    alfa_v = (alfa_v * Math.PI) / 180.0;

    x = x * Math.Cos(alfa_v) + y * Math.Sin(alfa_v);
    y = x * -Math.Sin(alfa_v) + y * Math.Cos(alfa_v);

    return new double[] { x, y, z };
}

private void openGLControl1_OpenGLDraw(object sender, RenderEventArgs args)
{
    // Создаем экземпляр
    gl = openGLControl1.OpenGL;

    // Очистка экрана и буфера глубин
    gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT);

    //gl.ClearColor(1f, 1f, 1f, 1);

    // Сбрасываем модельно-видовую матрицу
    gl.LoadIdentity();

    double t_x = (double)r_x.Value;
    double t_y = (double)r_y.Value;
    double t_z = (double)r_z.Value;

    double s1_x = (double)sonar1X.Value;
    double s1_y = (double)sonar1Y.Value;
    double s1_z = (double)sonar1Z.Value;

    float s1_ax = (float)sonar1alfaX.Value;
    float s1_ay = (float)sonar1alfaY.Value;
    float s1_az = (float)sonar1alfaZ.Value;

    double s2_x = (double)sonar2X.Value;
    double s2_y = (double)sonar2Y.Value;
    double s2_z = (double)sonar2Z.Value;

    float s2_ax = (float)sonar2alfaX.Value;
    float s2_ay = (float)sonar2alfaY.Value;
    float s2_az = (float)sonar2alfaZ.Value;

    double c_x = (double)cameraX.Value;
    double c_y = (double)cameraY.Value;
    double c_z = (double)cameraZ.Value;

    float c_ax = (float)cameraAlfaX.Value;
    float c_ay = (float)cameraAlfaY.Value;
    float c_az = (float)cameraAlfaZ.Value;

    //screenXM = (float)screenXnum.Value;
    //screenYM = (float)screenYnum.Value;

    double l_x = (double)lidarX.Value;
    double l_y = (double)lidarY.Value;
    double l_z = (double)lidarZ.Value;

    float l_a = (float)lidarAngle.Value;


    // Двигаем перо вглубь экрана
    gl.Translate(0.0f, 0.0f, -10.0f * (2 - global_scale));

    gl.Rotate(angleX - 75f, angleY, angleZ + 20);

    gl.Begin(OpenGL.GL_LINES);

    gl.Color(1f, 0f, 0f);
    gl.Vertex(5f, 0f, 0f);
    gl.Vertex(-5f, 0f, 0f);

    gl.Color(0f, 1f, 0f);
    gl.Vertex(0f, 5f, 0f);
    gl.Vertex(0f, -5f, 0f);

    gl.Color(0f, 0f, 1f);
    gl.Vertex(0f, 0f, 5f);
    gl.Vertex(0f, 0f, -5f);

    gl.End();

    //double scale_norm = Convert.ToDouble(scale_2.Value) / 10;

    if (bound_show.Checked)
    {
        gl.Begin(OpenGL.GL_LINES);     //Cube

        gl.Color(0.0f, 1.0f, 0.7f);
```

```
            gl.Vertex(bp[0, 0], bp[0, 1], bp[0, 2]);
            gl.Vertex(bp[1, 0], bp[1, 1], bp[1, 2]);

            gl.Vertex(bp[1, 0], bp[1, 1], bp[1, 2]);
            gl.Vertex(bp[2, 0], bp[2, 1], bp[2, 2]);

            gl.Vertex(bp[2, 0], bp[2, 1], bp[2, 2]);
            gl.Vertex(bp[3, 0], bp[3, 1], bp[3, 2]);

            gl.Vertex(bp[3, 0], bp[3, 1], bp[3, 2]);
            gl.Vertex(bp[0, 0], bp[0, 1], bp[0, 2]);


            gl.Vertex(bp[2, 0], bp[2, 1], bp[2, 2]);
            gl.Vertex(bp[6, 0], bp[6, 1], bp[6, 2]);

            gl.Vertex(bp[1, 0], bp[1, 1], bp[1, 2]);
            gl.Vertex(bp[5, 0], bp[5, 1], bp[5, 2]);

            gl.Vertex(bp[3, 0], bp[3, 1], bp[3, 2]);
            gl.Vertex(bp[7, 0], bp[7, 1], bp[7, 2]);

            gl.Vertex(bp[0, 0], bp[0, 1], bp[0, 2]);
            gl.Vertex(bp[4, 0], bp[4, 1], bp[4, 2]);


            gl.Vertex(bp[4, 0], bp[4, 1], bp[4, 2]);
            gl.Vertex(bp[5, 0], bp[5, 1], bp[5, 2]);

            gl.Vertex(bp[5, 0], bp[5, 1], bp[5, 2]);
            gl.Vertex(bp[6, 0], bp[6, 1], bp[6, 2]);

            gl.Vertex(bp[6, 0], bp[6, 1], bp[6, 2]);
            gl.Vertex(bp[7, 0], bp[7, 1], bp[7, 2]);

            gl.Vertex(bp[7, 0], bp[7, 1], bp[7, 2]);
            gl.Vertex(bp[4, 0], bp[4, 1], bp[4, 2]);

            gl.End();
        }

        if (lidarVis.Checked)
        {
            gl.LoadIdentity();

            gl.Translate(0.0f, 0.0f, -10.0f * (2 - global_scale));
            gl.Rotate(angleX - 75f, angleY, angleZ + 20);

            gl.Translate(t_x, t_y, t_z + 0.15);   // robot
            gl.Rotate((float)alfaX.Value, (float)alfaY.Value, (float)alfaZ.Value);

            gl.Translate(l_x, l_y, l_z - 0.2);
            gl.Rotate(0, 0, l_a);

            gl.Begin(OpenGL.GL_POINTS);     // Cloud

            while (busy && playPause.Text == "Pause") ;

            gl.Color(1.0f, 1.0f, 1.0f);

            for (int i = 0; i < cloudINShow.Count; i++)
            {
                try
                {
                    gl.Vertex(cloudINShow[i][0], cloudINShow[i][1], cloudINShow[i][2]);
                }
                catch (Exception er)
                {
                    //MessageBox.Show(er.ToString() + "\n\r" + "i: " + i + "\n\r\n\rCloud: " + cloud.Count + "\n\rCLC:
" + clc);
                }
            }

            gl.Color(0.3f, 0.3f, 0.3f);

            for (int i = 0; i < cloudOUTShow.Count; i++)
            {
                try
                {
                    gl.Vertex(cloudOUTShow[i][0], cloudOUTShow[i][1], cloudOUTShow[i][2]);
                }
                catch (Exception er)
                {
                    //MessageBox.Show(er.ToString() + "\n\r" + "i: " + i + "\n\r\n\rCloud: " + cloud.Count + "\n\rCLC:
" + clc);
                }
            }

            gl.End();
        }

        gl.LoadIdentity();

        gl.Translate(0.0f, 0.0f, -10.0f * (2 - global_scale));
        gl.Rotate(angleX - 75f, angleY, angleZ + 20);

        gl.Translate(t_x, t_y, t_z + 0.15);   // robot
```

```
gl.Rotate((float)alfaX.Value, (float)alfaY.Value, (float)alfaZ.Value);

if (robotVis.Checked)
{
   gl.Begin(OpenGL.GL_POLYGON);
   gl.Color(0.0f, 0.8f, 0.6f);

   gl.Vertex(-0.15, -0.15, -0.15);
   gl.Vertex(-0.15, 0.15, -0.15);
   gl.Vertex(0.15, 0.15, -0.15);
   gl.Vertex(0.15, -0.15, -0.15);

   gl.End();

   gl.Begin(OpenGL.GL_POLYGON);

   gl.Vertex(0.15, -0.15, 0.15);
   gl.Vertex(0.15, 0.15, 0.15);
   gl.Vertex(-0.15, 0.15, 0.15);
   gl.Vertex(-0.15, -0.15, 0.15);

   gl.Vertex(-0.15, -0.15, -0.15);
   gl.Vertex(0.15, -0.15, -0.15);
   gl.Vertex(0.15, 0.15, -0.15);
   gl.Vertex(0.15, 0.15, 0.15);

   gl.End();

   gl.Begin(OpenGL.GL_POLYGON);
   gl.Color(1f, 1f, 1f);

   gl.Vertex(0.15, 0.15, 0.15);
   gl.Vertex(-0.15, 0.15, 0.15);
   gl.Vertex(-0.15, 0.15, -0.15);
   gl.Vertex(0.15, 0.15, -0.15);

   gl.End();

   gl.Begin(OpenGL.GL_POLYGON);
   gl.Color(0.0f, 0.8f, 0.6f);

   gl.Vertex(-0.15, 0.15, 0.15);
   gl.Vertex(-0.15, 0.15, -0.15);
   gl.Vertex(-0.15, -0.15, -0.15);
   gl.Vertex(-0.15, -0.15, 0.15);

   gl.End();
}

if (sonar1Vis.Checked)
{
   gl.LoadIdentity();

   gl.Translate(0.0f, 0.0f, -10.0f * (2 - global_scale));
   gl.Rotate(angleX - 75f, angleY, angleZ + 20);

   gl.Translate(t_x, t_y, t_z + 0.15);    // robot
   gl.Rotate((float)alfaX.Value, (float)alfaY.Value, (float)alfaZ.Value);

   gl.Translate(s1_x, s1_y, s1_z + 0.05);    // sonar 1
   gl.Rotate(s1_ax, s1_ay, s1_az);

   gl.Begin(OpenGL.GL_POLYGON);
   gl.Color(0.7f, 0.5f, 0.9f);

   gl.Vertex(-0.05, 0, -0.05);
   gl.Vertex(0.05, 0, -0.05);
   gl.Vertex(0.05, 0, 0.05);
   gl.Vertex(-0.05, 0, 0.05);

   gl.Vertex(-0.05, 0.05, 0.05);
   gl.Vertex(-0.05, 0.05, -0.05);
   gl.Vertex(0.05, 0.05, -0.05);
   gl.Vertex(0.05, 0, -0.05);

   gl.End();

   gl.Begin(OpenGL.GL_POLYGON);
   gl.Color(0.7f, 0.5f, 0.9f);

   gl.Vertex(0.05, 0.05, -0.05);
   gl.Vertex(0.05, 0, -0.05);
   gl.Vertex(0.05, 0, 0.05);
   gl.Vertex(0.05, 0.05, 0.05);

   gl.Vertex(-0.05, 0.05, 0.05);
   gl.Vertex(-0.05, 0, 0.05);
   gl.Vertex(0.05, 0, 0.05);

   gl.End();

   gl.Begin(OpenGL.GL_POLYGON); // sonars 1 plane
   gl.Color(1f, 0.6f, 1f);

   gl.Vertex(-0.09, 0.051 + sonar1Dist, -0.09);
   gl.Vertex(0.09, 0.051 + sonar1Dist, -0.09);
   gl.Vertex(0.09, 0.051 + sonar1Dist, 0.09);
   gl.Vertex(-0.09, 0.051 + sonar1Dist, 0.09);
```

```
    gl.End();

    gl.Color(0.8f, 0.8f, 0.9f);
    //gl.DrawText3D("Arial", 0.1f, 0.2f, 0.0f, "Text");
    gl.DrawText(5, 10, 190.0f, 128.0f, 255.0f, "Arial", 12.0f, sonar1Dist.ToString() + " m");
    gl.DrawText(5, 30, 190.0f, 128.0f, 255.0f, "Arial", 12.0f, sonar2Dist.ToString() + " m");
}

if (sonar2Vis.Checked)
{
    gl.LoadIdentity();

    gl.Translate(0.0f, 0.0f, -10.0f * (2 - global_scale));
    gl.Rotate(angleX - 75f, angleY, angleZ + 20);

    gl.Translate(t_x, t_y, t_z + 0.15);    // robot
    gl.Rotate((float)alfaX.Value, (float)alfaY.Value, (float)alfaZ.Value);


    gl.Translate(s2_x, s2_y, s2_z + 0.05);    // sonar 2
    gl.Rotate(s2_ax, s2_ay, s2_az);

    gl.Begin(OpenGL.GL_POLYGON);
    gl.Color(0.7f, 0.5f, 0.9f);

    gl.Vertex(-0.05, 0, -0.05);
    gl.Vertex(0.05, 0, -0.05);
    gl.Vertex(0.05, 0, 0.05);
    gl.Vertex(-0.05, 0, 0.05);

    gl.Vertex(-0.05, 0.05, 0.05);
    gl.Vertex(-0.05, 0.05, -0.05);
    gl.Vertex(0.05, 0.05, -0.05);
    gl.Vertex(0.05, 0, -0.05);

    gl.End();

    gl.Begin(OpenGL.GL_POLYGON);
    gl.Color(0.7f, 0.5f, 0.9f);

    gl.Vertex(0.05, 0.05, -0.05);
    gl.Vertex(0.05, 0, -0.05);
    gl.Vertex(0.05, 0, 0.05);
    gl.Vertex(0.05, 0.05, 0.05);

    gl.Vertex(-0.05, 0.05, 0.05);
    gl.Vertex(-0.05, 0, 0.05);
    gl.Vertex(0.05, 0, 0.05);

    gl.End();

    gl.Begin(OpenGL.GL_POLYGON); // sonars 2 plane
    gl.Color(1f, 0.6f, 1f);

    gl.Vertex(-0.09, 0.051 + sonar2Dist, -0.09);
    gl.Vertex(0.09, 0.051 + sonar2Dist, -0.09);
    gl.Vertex(0.09, 0.051 + sonar2Dist, 0.09);
    gl.Vertex(-0.09, 0.051 + sonar2Dist, 0.09);

    gl.End();
}

if (cameraVis.Checked)
{
    gl.LoadIdentity();

    gl.Translate(0.0f, 0.0f, -10.0f * (2 - global_scale));
    gl.Rotate(angleX - 75f, angleY, angleZ + 20);

    gl.Translate(t_x, t_y, t_z + 0.15);    // robot
    gl.Rotate((float)alfaX.Value, (float)alfaY.Value, (float)alfaZ.Value);

    gl.Translate(c_x, c_y, c_z + 0.4);    // camera
    gl.Rotate(c_ax, c_ay + 180, c_az);

    if (playPause.Text != "Start")
    {
        try
        {
            int h = frame_draw.Size().Height;
            int w = frame_draw.Size().Width;

            //if (h != 0)
            //    MessageBox.Show(h + "\n\r" + w);

            if (h % 2 != 0)
                h = h - 1;

            if (w % 2 != 0)
                w = w - 1;

            gl.Begin(OpenGL.GL_POINTS);

            for (int y = -h / 2; y < h / 2; y++)
            {
                for (int x = -w / 2; x < w / 2; x++)
                {
```

```
                if ((readed_once))
                {
                    Vec3b color = frame_draw.Get<Vec3b>(y + h / 2, x + w / 2);

                    gl.Color(color.Item0, color.Item1, color.Item2);
                    gl.Vertex(x * 0.001, 0, y * 0.001);
                }
            }
        }

        drawn = true;

        gl.End();
    }
    catch (Exception er)
    {
        MessageBox.Show(er.ToString());
    }
  }
  else
  {
    gl.Begin(OpenGL.GL_POLYGON);
    gl.Color(0.7f, 0.7f, 0.7f);

    double screenScl = (double)screenScale.Value;

    gl.Vertex(-0.35 * screenScl, 0, -0.25 * screenScl);
    gl.Vertex(0.35 * screenScl, 0, -0.25 * screenScl);
    gl.Vertex(0.35 * screenScl, 0, 0.25 * screenScl);
    gl.Vertex(-0.35 * screenScl, 0, 0.25 * screenScl);

    gl.End();
  }
}

if (lidarVis.Checked)
{
  gl.LoadIdentity();

  gl.Translate(0.0f, 0.0f, -10.0f * (2 - global_scale));
  gl.Rotate(angleX - 75f, angleY, angleZ + 20);

  gl.Translate(t_x, t_y, t_z + 0.15);    // robot
  gl.Rotate((float)alfaX.Value, (float)alfaY.Value, (float)alfaZ.Value);


  gl.Translate(l_x, l_y, l_z + 0.2);    // lidar
  gl.Rotate(0, 0, l_a);

  gl.Begin(OpenGL.GL_POLYGON);
  gl.Color(0.5f, 0.5f, 0.5f);
  gl.Vertex(0.05, 0.05, 0.2);
  gl.Vertex(-0.05, 0.05, 0.2);
  gl.Vertex(-0.05, -0.05, 0.2);
  gl.Vertex(0.05, -0.05, 0.2);
  gl.End();

  gl.Begin(OpenGL.GL_POLYGON);
  gl.Vertex(0.05, 0.05, 0.2);
  gl.Vertex(-0.05, 0.05, 0.2);
  gl.Vertex(-0.05, 0.05, 0);
  gl.Vertex(0.05, 0.05, 0);
  gl.End();

  gl.Begin(OpenGL.GL_POLYGON);
  gl.Vertex(0.05, -0.05, 0.2);
  gl.Vertex(-0.05, -0.05, 0.2);
  gl.Vertex(-0.05, -0.05, 0);
  gl.Vertex(0.05, -0.05, 0);
  gl.End();

  gl.Begin(OpenGL.GL_POLYGON);
  gl.Vertex(0.05, 0.05, 0.2);
  gl.Vertex(0.05, -0.05, 0.2);
  gl.Vertex(0.05, -0.05, 0);
  gl.Vertex(0.05, 0.05, 0);
  gl.End();

  gl.Begin(OpenGL.GL_POLYGON);
  gl.Vertex(-0.05, 0.05, 0.2);
  gl.Vertex(-0.05, -0.05, 0.2);
  gl.Vertex(-0.05, -0.05, 0);
  gl.Vertex(-0.05, 0.05, 0);
  gl.End();
}

gl.Flush();

imageData = new byte[4 * xSave * ySave];
imageData2 = new byte[4 * xSave * ySave];
gl.ReadPixels(0, 0, xSave, ySave, OpenGL.GL_RGBA, OpenGL.GL_UNSIGNED_BYTE, imageData);
gl.ReadPixels(0, 0, xSave, ySave, OpenGL.GL_RGBA, OpenGL.GL_UNSIGNED_BYTE, imageData2);


if (goRecord)
{
  if (imageCounter % 5 == 0)
  {
```

```
            SaveImage();
        }
        else
            imageCounter++;
    }
    else
    {
        imageCounter = 1;
    }

    if (needAScreenshot)
    {
        MadeAScreenShot();
        screenshotNumber++;
        needAScreenshot = false;
    }
}

private void SonarReciever()
{
    port = new SerialPort();

    try
    {
        // настройки порта
        port.PortName = ports[0];
        port.BaudRate = 9600;
        port.DataBits = 8;
        port.Parity = System.IO.Ports.Parity.None;
        port.StopBits = System.IO.Ports.StopBits.One;
        port.ReadTimeout = 1000;
        port.WriteTimeout = 1000;
        port.Open();
    }
    catch (Exception er)
    {
        Console.WriteLine("ERROR: невозможно открыть порт:" + er.ToString());
        return;
    }

    Encoding ascii = Encoding.ASCII;
    System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = false;

    while (true)
    {
        int intBuffer;
        intBuffer = port.BytesToRead;

        byte[] byteBuffer = new byte[intBuffer];
        port.Read(byteBuffer, 0, intBuffer);

        if (byteBuffer.Length != 0)
        {
            string msg = ascii.GetString(byteBuffer);
            if (msg.IndexOf("@") == -1)
            {
                string[] GPIOports = msg.Split('_');

                //отрисовываем

                //System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = false;

                try
                {
                    sonar1Dist = Convert.ToDouble(GPIOports[0]) / 1000.0;
                    sonar2Dist = Convert.ToDouble(GPIOports[1]) / 1000.0;
                }
                catch (Exception er)
                {
                    //MessageBox.Show(er.ToString());
                }

                //System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = true;
            }
        }
    }
}

private double[] NewBase(double x, double y, double z, double t_x, double t_y, double t_z, double alfa)
{
    double alfa_r = (alfa * Math.PI) / 180.0;

    double x_new = x * Math.Cos(alfa_r) + y * Math.Sin(alfa_r) - t_x;
    double y_new = x * -Math.Sin(alfa_r) + y * Math.Cos(alfa_r) - t_y;
    double z_new = z - t_z;

    return new double[] { x_new, y_new, z_new };
}

private void RosaFunc(object sender, EventArgs e)
{
    try
    {
        FileStream fs = File.OpenRead(pathVar);

        int c;
        int byteCount = 0;
        byte memoryByte = 0;
```

```
byte[] buf = new byte[1];

int clearCount = 0;

int rotatoinAngleByte1 = 0;
int rotatoinAngleByte2 = 0;
double rotatoinAngle = 0;

int laserNum = 0;
int byteLaserNum = 0;
double firstLaserByte = 0;
double secondLaserByte = 0;
bool volk = true;

while ((c = fs.Read(buf, 0, buf.Length)) > 0)
{
   bufText.Add(buf[0]);

   if (byteCount == 1)
      rotatoinAngleByte1 = buf[0];

   if (byteCount == 2)
   {
      rotatoinAngleByte2 = buf[0];
      rotatoinAngle = (rotatoinAngleByte2 * 256.0 + rotatoinAngleByte1) / 100.0;
   }

   if ((byteCount > 2) && (byteCount < 101))
   {
      if (byteLaserNum == 0)
         firstLaserByte = buf[0];

      if (byteLaserNum == 1)
         secondLaserByte = buf[0];

      if (byteLaserNum == 2)
      {
         double laser = (secondLaserByte * 256.0 + firstLaserByte) * 0.001;

         cloudEx.Add(new double[] { 0, tiltAngleCos[laserNum] * laser,
            tiltAngleSin[laserNum] * laser, laser}); // четвертой координатой записал дальность

         byteLaserNum = -1;
         laserNum++;
      }

      byteLaserNum++;
   }

   if ((buf[0] == 0xEE) && (memoryByte == 0xFF))
   {
      laserNum = 0;
      byteLaserNum = 0;

      double max_differential_h = 2;

      double radAngle = rotatoinAngle * Math.PI / 180.0;

      if (cloudEx.Count == 32)
      {
         for (int i = 1; i < 31; i++)
         {
            if ((Math.Abs(cloudEx[i - 1][3] - cloudEx[i][3]) > max_differential_h)
               && (Math.Abs(cloudEx[i + 1][3] - cloudEx[i][3]) > max_differential_h)
               )
               cloudEx[i][0] = -100;

            if (cloudEx[i][3] < 0.5)
               cloudEx[i][0] = -100;
         }

         for (int i = 0; i < 32; i++)
         {
            double newX = cloudEx[i][0] * Math.Cos(radAngle) + cloudEx[i][1] * Math.Sin(radAngle);
            double newY = cloudEx[i][0] * -Math.Sin(radAngle) + cloudEx[i][1] * Math.Cos(radAngle);
            double newZ = cloudEx[i][2];

            if ((newX >= xL) && (newX <= xR)
               && (newY >= yL) && (newY <= yR)
               && (newZ >= zL) && (newZ <= zR))
            {
               if (volk == true)
               {
                  cloudIN.Add(new double[] { newX, newY, newZ });

                  double t_x = (double)r_x.Value;
                  double t_y = (double)r_y.Value;
                  double t_z = (double)r_z.Value;
                  double angle = (double)alfaX.Value;

                  double[] barrier = NewBase(newX, newY, newZ, t_x, t_y, t_z, angle);

                  if ((barrier[0] >= -0.15) && (barrier[0] <= 0.15) &&
                     (barrier[2] >= 0) && (barrier[2] <= 0.3) &&
                     (barrier[1] >= 0))
                  {
                     l_path = barrier[1];
                  }
```

```
                    else
                        volk = false;
                }
                else
                    volk = true;
            }
            else
            {
                cloudOUT.Add(new double[] { newX, newY, newZ });
            }
        }

        byteCount = 0;

        cloudEx.Clear();

        clearCount++;

        try
        {
            if (clearCount == 2500)
            {
                busy = true;
                cloudINShow = cloudIN.Select(item => (double[])item.Clone()).ToList();
                cloudOUTShow = cloudOUT.Select(item => (double[])item.Clone()).ToList();
                busy = false;

                int num_of_points = cloudINShow.Count;

                double[,] cloudInArr = new double[num_of_points, 3];

                for (int i = 0; i < num_of_points; i++)
                {
                    cloudInArr[i, 0] = cloudINShow[i][0];
                    cloudInArr[i, 1] = cloudINShow[i][1];
                    cloudInArr[i, 2] = cloudINShow[i][2];
                }

                busy = true;
                bps = bound_points.Select(item => (double[])item.Clone()).ToList();
                cloudIN.Clear();
                cloudOUT.Clear();
                bound_points.Clear();
                busy = false;

                clearCount = 0;

                Thread.Sleep(Convert.ToInt32(readSpeed.Value));
            }
        }
        catch { }
      }
    }

    memoryByte = buf[0];

    byteCount++;
  }
 }
 catch (Exception er)
 {
  //MessageBox.Show(er.ToString());
 }
 playPause.Text = "Restart";
}

public double DistanceBetween3D(double x1, double y1, double z1, double x2, double y2, double z2)
{
 return Math.Sqrt(Math.Pow(x2 - x1, 2) + Math.Pow(y2 - y1, 2) + Math.Pow(z2 - z1, 2));
}

private void OpenGLDraw_func(object sender, RenderEventArgs args)
{
 openGLControl1_OpenGLDraw(sender, args);
}

private void Scale_change(object sender, EventArgs e)
{
 openGLControl1_OpenGLDraw(sender, args);

 global_scale = (double)scale_2.Value / 10.0;
}

private void KeyDown(object sender, KeyEventArgs e)
{

}

private void MouseDown(object sender, MouseEventArgs e)
{
 mouse_down = true;

 oldValueX = e.X;
 oldValueY = e.Y;
}

private void MouseUp(object sender, MouseEventArgs e)
```

```
        {
            mouse_down = false;
        }

        private void MouseMove(object sender, MouseEventArgs e)
        {
            if (mouse_down)
            {
                angleZ += Convert.ToInt32((e.X - oldValueX) / 2.0);
                angleX += Convert.ToInt32((e.Y - oldValueY) / 2.0);

                openGLControl1_OpenGLDraw(sender, args);

                oldValueX = e.X;
                oldValueY = e.Y;
            }
        }

        private void ReadDump(object sender, EventArgs e)
        {
            if (playPause.Text == "Start")
            {
                if (firstRead)
                {
                    rosa = new Thread(() => RosaFunc(sender, e));
                    memosa = new Thread(() => SonarReciever());
                    camera = new Thread(new ThreadStart(CaptureCameraCallback));
                    rosa.Start();
                    camera.Start();
                    memosa.Start();
                    cameraOpened = true;
                    firstRead = false;
                }
                else
                {
                    rosa.Resume();
                    camera.Resume();
                    memosa.Resume();
                }

                playPause.Text = "Pause";
            }
            else
            if (playPause.Text == "Pause")
            {
                rosa.Suspend();
                camera.Suspend();
                memosa.Suspend();
                playPause.Text = "Start";
            }
            else
            if (playPause.Text == "Restart")
            {
                rosa.Abort();
                //camera.Abort();
                //memosa.Abort();
                firstRead = true;
                playPause.Text = "Start";
            }
        }

        private void BrowseSave(object sender, EventArgs e)
        {
            System.Windows.Forms.FolderBrowserDialog dlg = new System.Windows.Forms.FolderBrowserDialog();

            DialogResult result = dlg.ShowDialog();

            //pathSave.Text = dlg.SelectedPath;
        }

        private void SaveImage()
        {
            Bitmap bit = new Bitmap(ySave, xSave);

            int arrCounter = 0;

            for (int i = 0; i < ySave; ++i)
            {
                for (int j = 0; j < xSave; ++j)
                {
                    bit.SetPixel(i, j, Color.FromArgb(imageData[arrCounter], imageData[arrCounter + 1],
imageData[arrCounter + 2]));
                    arrCounter += 4;
                }
            }

            Bitmap bit2 = new Bitmap(xSave, ySave);

            for (int i = 0; i < ySave; ++i)
            {
                for (int j = 0; j < xSave; ++j)
                {
                    Color c = bit.GetPixel(i, j);

                    int r = c.R;
                    int g = c.G;
                    int b = c.B;
```

```
                bit2.SetPixel(j, ySave - 1 - i, Color.FromArgb(r, g, b));
            }
        }

        bit2.Save(pathVideo + "//image" + imageCounter / 5 + ".png");

        imageCounter++;
    }

    private void Form1_ClosingEvent(object sender, FormClosingEventArgs e)
    {
        try
        {
            rosa.Resume();
            camera.Resume();
            memosa.Resume();
            port.Close();
        }
        catch { }

        try
        {
            rosa.Abort();
            camera.Abort();
            memosa.Abort();
            port.Close();
        }
        catch { }
    }

    private void BoundariesCount(object sender, EventArgs e)
    {
        BoundariesCountT();

        openGLControl1_OpenGLDraw(sender, args);
    }

    private void ChangeView(object sender, EventArgs e)
    {
        openGLControl1_OpenGLDraw(sender, args);
    }

    private void Rob_rot(object sender, EventArgs e)
    {
        //double robot_alfa = ((double)alfa.Value * Math.PI) / 180.0;

        //double t_x = ((double)r_x.Value + 0.15);
        //double t_y = ((double)r_y.Value + 0.15);
        //double t_z = ((double)r_z.Value);

        ////t_x = t_x * Math.Cos(robot_alfa) + t_y * Math.Sin(robot_alfa);
        ////t_y = -t_x * Math.Sin(robot_alfa) + t_y * Math.Cos(robot_alfa);

        //MessageBox.Show(t_x + "\n\r" + t_y + "\n\r" + t_z);
    }

    private void MadeAScreenShot()
    {
        Bitmap bit = new Bitmap(ySave, xSave);

        int arrCounter = 0;

        for (int i = 0; i < ySave; ++i)
        {
            for (int j = 0; j < xSave; ++j)
            {
                bit.SetPixel(i, j, Color.FromArgb(imageData[arrCounter], imageData[arrCounter + 1],
imageData[arrCounter + 2]));
                arrCounter += 4;
            }
        }

        Bitmap bit2 = new Bitmap(xSave, ySave);

        for (int i = 0; i < ySave; ++i)
        {
            for (int j = 0; j < xSave; ++j)
            {
                Color c = bit.GetPixel(i, j);

                int r = c.R;
                int g = c.G;
                int b = c.B;

                bit2.SetPixel(j, ySave - 1 - i, Color.FromArgb(r, g, b));
            }
        }

        bit2.Save(pathScreen + "//image" + screenshotNumber + ".png");
    }

    private void ScreenshotToolStripMenuItem_Click(object sender, EventArgs e)
    {
        needAScreenshot = true;
    }

    private void RecordToolStripMenuItem_Click(object sender, EventArgs e)
    {
```

```
            if (!goRecord)
            {
                string[] picList = Directory.GetFiles(pathVideo, "*");
                foreach (string f in picList)
                    File.Delete(f);

                goRecord = true;
                recordLab.Text = "Stop record";
            }
            else
            {
                videoNumber++;
                goRecord = false;
                recordLab.Text = "Saving...";

                CreateMovie();

                recordLab.Text = "Start record";
            }
        }
    }
}
```