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

Отчёт о прохождении производственной практики

По направлению: *15.03.06 - "Мехатроника и робототехника"*
Место прохождения практики: *"Центральный научно-исследовательский институт робототехники и технической кибернетики"*
Сроки практики: *10.06.2023-08.07.2023*

Руководитель практики: *Уланов В.Н.*
Научный руководитель: *Варлашин В.В.*
Студент гр. 3331506/00401: *Шевцов Иван*
Проект на Github
Оценка:

лето 2023

Содержание

1 Введение

Практика проходила в ЦНИИ РТК в отделе 111 ЦМРТК. ЦНИИ РТК - научно-исследовательский и опытно-конструкторский институт, разрабатывающий мехатронные, кибернетические и робототехнические системы и комплексы от стадии концептуального проектирования до изготовления изделий, готовых к серийному производству. Также является инновационной компанией, первой в России в роботостроении специального назначения.

Задача была поставлена в рамках НИОКТР, выполняемых ЦНИИ РТК при реализации комплексного проекта по созданию высокотехнологичного производства «Разработка роботизированного диагностического комплекса для внутритрубного контроля трубопроводов». Внутритрубная диагностика – необходимое мероприятие для инспекции текущего состояния эксплуатируемых трубопроводов. В зависимости от целей диагностики трубопроводов проводится контроль различных параметров и элементов трубопровода: параметры геометрии трубопровода, элементы обустройства, соединительные детали, швы, геометрические дефекты трубопровода, параметры поверхностных дефектов и дефектов металла трубопровода, картография трубопровода, степень загрязненности трубопровода [1].

Одним из методов внутритрубной диагностики является визуально-измерительный контроль – группа методов, основанных на системах технического зрения, таких как сканирование поверхности 2D- и 3D-лазерными сканерами, видеокамерами со структурированной подсветкой, Time-of-Flight (ToF) камерами. В зависимости от используемых методов, ВИК позволяет определять наличие ржавчины, вмятин, крупных трещин и других геометрических дефектов внутри трубы, а также производить картографирование трубы. [1].

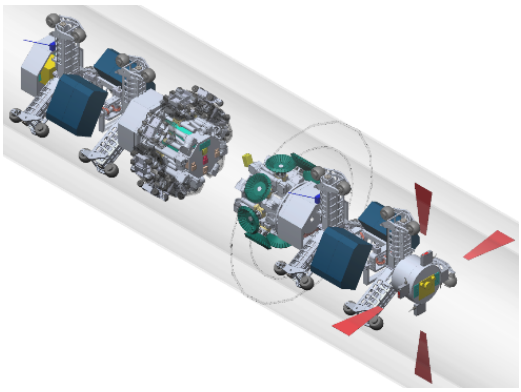

Рис. 1: Проектный облик разрабатываемого устройства [1].

Постановка задачи

В качестве практического задания необходимо было внедрить лазерный триангуляционный датчкик LS2D от компании RPIZMA в систему ROS (robot operation system) робота. В представленном компанией решении есть библиотека на языке C++ по работе с датчиком, необходимо её приспособить для потокового вывода данных в ROS topic.

Для автоматизации сборки проекта со всеми зависимостями и работе на любой операционной системе использовался docker: образ контейнера содержит операционную систему ubuntu 20.04, ros noetic, инструменты отладки и некоторые дополнительные пакеты. Описание запуска и работы с проектом содержится в readme файле репозитория проекта.

Краткие теоретические сведения

Определение 1.1. *docker* - это инструмент работы с контейнирами, с помощью них можно задавать зависимости сборки, необходимые программы. Благодаря этому гарантируется одинаковый запуск проекта на разных платформах.

Определение 1.2. *ROS* — свободно распространяемая мета-операционная система для роботов. ROS обеспечивает стандартные службы операционной системы, такие как:

- ▷ аппаратная абстракция
- ▷ низкоуровниевое управление устройствами
- ▷ гототвые реализации часто используемых фунций
- ▷ передачу информации между процессами
- ▷ управление пакетами

Определение 1.3. *Нода, узел* - единица программного кода, исполняемая в отдельном потоке и выполняющая определенную вычислительную функцию. Ноды коммуницируют друг с другом по средсвам топиков.

Определение 1.4. *Топик* - система передачи данных с механизмом подписки/публикации. Нода отправляет данные в топик, другая нода может читать из этого топика. Одна нода не имеет ограничений на колличество топиков для публикации и чтения.

Определение 1.5. *Тип сообщения* - описание формата сообщеиня, публикующегося в топике. Каждый топик публикует сообщение определённого типа.

2 Описание решения

2.1 Принцип работы датчка и типовые примеры использования

LS2D – лазерный триангуляционный 2-D датчик (сканер) со встроенной микропроцессорной системой управления. LS2D позволяет с высокой точностью мгновенно измерять профиль объекта без механического контакта с ним.

В сканерах LS2D используется принцип триангуляции. Развернутый в идеальную прямую линию лазерный луч проецируется на поверхность контролируемого объекта. Световая линия повторяет форму профиля объекта в сечении. Изображение световой линии проецируется на КМОП-фотоматрицу.

По координатам изображения на фотоприемнике микропроцессор производит вычисление реальных координат световой линии. Для получения трехмерной модели формы или поверхности объекта можно использовать перемещение сканера LS2D с учетом точной величины этого перемещения.

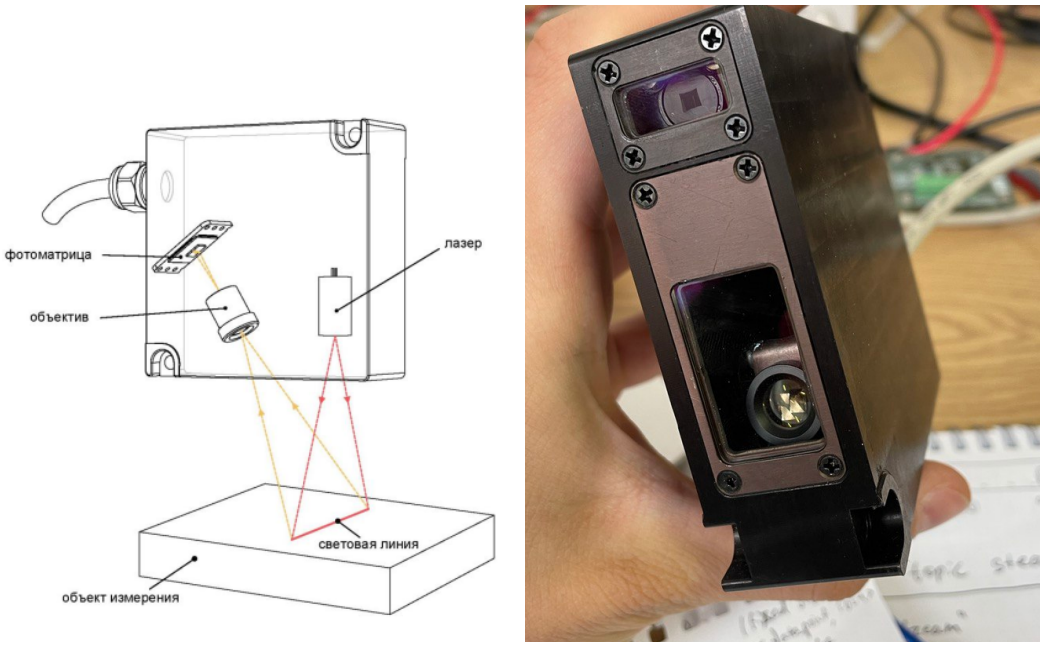

Рис. 2: Устройство датчика

Ширина сканирования зависит от удалённости объекта и спецификации отдельной модели. Подробнее о работе датчика и его ограничениях можно почитать на странице продукта в разделе "ссылки".

Измерение профиля поверхности одним сканером (движущаяся полоса продукции, протяжённые тела, тела вращения) позволяет определять: параметры разнотолщинности в продольном и поперечном направлении, кромку полосы, локальные дефекты, биения, отклонения профиля от нужной геометрии, внутренний и наружный диаметры, качество сварных швов, параметры резьбы и т. д. Для получения полной или частичной трехмерной модели объекта нужно использовать скорость движения объекта, заданную фиксировано или измеряемую динамически. Непрерывный контроль продукции позволяет непосредственно управлять техническим процессом. Например, можно сортировать изделия, идущие по конвейеру, или динамически корректировать толщину экструзионных изделий,

или учитывать объём сыпучих материалов, движущихся на транспортёрной ленте. Создание 3-D модели с использованием одного или двух сканеров одновременно позволяет полностью контролировать все допуски, осуществлять поиск и контроль характерных мест (отверстий, щелей, валов), измерять межосевые углы и расстояния и все другие параметры, которые очень сложно или практически невозможно точно контролировать другими механическими средствами измерения.

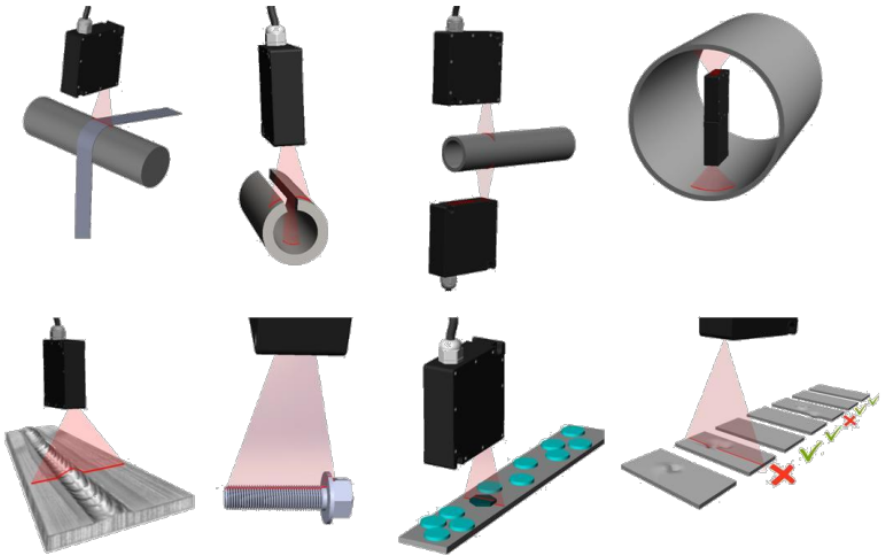

Рис. 3: Области применения

2.2 Описание работы кода

Алгоритм запуска ноды

В контейнер монтируется воркспейс, внутри него в папке src/ структура пакета выглядит следующим образом:

```
ls2d
|-- CMakeLists.txt
|-- config
|   |-- lazerscan_viz.rviz
|   `-- poitcloud2_viz.rviz
|-- include
|   `-- prism
|       |-- s6
|       |   |-- ls2d_device.h
|       |   |-- ls2d_settings.h
|       |   |-- sensor_info.h
|       |   `-- sensor_search.h
|       `-- utils
|           |-- byte_manip.h
|           `-- timer.h
|-- launch
|   |-- ls2d.launch
|   `-- point_cloud_to_lazerscan.launch
|-- msg
|   `-- Sensor_info.msg
|-- package.xml
|-- readme.md
`-- src
|-- ls2d_init_node.cpp
`-- prism
|-- s6
|   |-- ls2d_device.cpp
|   |-- ls2d_settings.cpp
|   |-- sensor_info.cpp
|   `-- sensor_search.cpp
`-- utils
`-- timer.cpp
```

Файлы из папок prism являются библиотекой производителя датчика, в них были внесены небольшие изменения: тип точки был изменён на float и данные с датчика записываются в другом формате.

Лаунч файл запускает ноду с параметрами датчика, при успешном соединении нода передаёт в терминал сообщение подтверждения соединения, иначе выдаётся сообщение об ошибке и нода завершает работу. Связь с датчиком осуществляется через ethernet кабель по udp протаколу. Таже лаунч файл запускает rviz с необходимом config-файлом и трансляцию static_tf между системой координат датчика и мира.

Данные датчика публикуются в соответсвующеем топике с кастомным сообщением типа Sensor_info.msg, составленного из типов std_msgs:

```
Header header
string ip
string descrioption
string serial
string firmware_ver
uint8 fractional_bits
```

После этого происходит публикация сообщений с координатами точек в топик /point_cloud, тип sensor_msgs/PointCloud2.msg:

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Датчик выдаёт 1024 точки каждые 10 мс, больше 10 тысяч точек в секунду, поэтому очень важно не допускать лишнего копирования точек из одного контейнера c++ в другой.

Для преобразования получаемого с датчика облака точек к сообщению типа ‘sensor_msgs/LaserSca нужно запусить лаунч файл point_cloud_to_lazerscan.launch из этого пакета. При необходимости в нём можно задать имя фрейма сенсора, названия топиков для входого облака точек (по умолчанию /point_cloud) и выходного сообщения laserscan(по умолчанию /scan_msgs).

Функция connect ноды

```
bool connect(ros::NodeHandle& n){
        ros::Publisher connect_pub =
                n.advertise<std_msgs::Bool>("connect_state_publisher", 1);
        std_msgs::Bool msg_connect;

        ROS_INFO("Connecting to IP [%s] ...", sensor_ip_addr.c_str() );
        if (!sens.connect(sensor_ip_addr)) {
                ROS_INFO("FAILED");
                msg_connect.data = false;
                connect_pub.publish(msg_connect);
                return false;
        }
        else {
                ROS_INFO("CONNECTED");
                msg_connect.data = true;
                connect_pub.publish(msg_connect);
        }
        ros::Publisher sensor_info_pub =
                        n.advertise<ls2d::Sensor_info>("sensor_info", 10);

        ls2d::Sensor_info msg;
        auto info = sens.info();
        msg.header.stamp    = ros::Time::now();
        msg.header.frame_id = "/sensor_base_link";
        msg.ip              = sens.info().ip_addr;
        msg.descrioption    = sens.info().description;
        msg.serial          = sens.info().serial;
        msg.firmware_ver    = sens.info().firmware_ver;
        msg.fractional_bits = sens.info().fractional_bits;

        sensor_info_pub.publish(msg);
        return true;
}
```

Основное тело ноды

```
int main(int argc, char **argv){
ros::init(argc, argv, "ls2d_sensor");
ros::NodeHandle n;

if (connect(n)){
        ros::Rate loop_rate(15);

        ros::Publisher stream_pub =
                n.advertise<sensor_msgs::PointCloud2>("point_cloud", 100000);

        result_info info;
        std::array<data_point, POINTS_IN_ARRAY> points;
        sensor_msgs::PointCloud2 cloud_msg;

        std::vector<unsigned char> vector_char;
        vector_char.resize(SIZEOF_FLOAT * POINTS_IN_ARRAY * 3);

        std::vector<sensor_msgs::PointField> field_vector_(3);

        field_vector_[0].name     = "x";
        field_vector_[0].offset   = 0;
        field_vector_[0].datatype = sensor_msgs::PointField::FLOAT32;
        field_vector_[0].count    = 1;

        field_vector_[1].name     = "y";
        field_vector_[1].offset   = SIZEOF_FLOAT;
        field_vector_[1].datatype = sensor_msgs::PointField::FLOAT32;
        field_vector_[1].count    = 1;

        field_vector_[2].name     = "z";
        field_vector_[2].offset   = SIZEOF_FLOAT * 2;
        field_vector_[2].datatype = sensor_msgs::PointField::FLOAT32;
        field_vector_[2].count    = 1;

        while (ros::ok()){
                sens.stream_enable(true);

                this_thread::sleep_for(11ms);
                sens.stream_read(&info, points.data());

                int offset = 0;
                for (unsigned point_idx = 0; point_idx < POINTS_IN_ARRAY;
                                ++point_idx) {
                        data_point point = points[point_idx];

                        memcpy(&vector_char[offset + SIZEOF_FLOAT * 0], &point.x,
                        SIZEOF_FLOAT);
                        memcpy(&vector_char[offset + SIZEOF_FLOAT * 1], &point.y,
                        SIZEOF_FLOAT);
                        memcpy(&vector_char[offset + SIZEOF_FLOAT * 2], &point.z,
                        SIZEOF_FLOAT);

                        offset += 3 * SIZEOF_FLOAT;
                }
```

```
                cloud_msg.data   = vector_char;
                cloud_msg.header.stamp    = ros::Time::now();
                cloud_msg.header.frame_id = "/sensor_base_link";
                cloud_msg.height          = 1;
                cloud_msg.width           = POINTS_IN_ARRAY;
                cloud_msg.fields          = field_vector_;
                cloud_msg.is_bigendian    = false;
                cloud_msg.point_step      = 3 * SIZEOF_FLOAT;
                cloud_msg.row_step        = 3 * SIZEOF_FLOAT   * POINTS_IN_ARRAY;
                cloud_msg.is_dense        = true;

                stream_pub.publish(cloud_msg);
                ROS_INFO("Published!");   // debug

                sens.stream_enable(false);

                ros::spinOnce();
                loop_rate.sleep();
        }

        }
        else   {ROS_INFO("Connection false, please, retry. Exit.");}
        return 0;
};
```

Граф rqt

Определение 2.1. *rqt* - это базирующийся на Qt фреймворк с графическим программным интерфейсом для ros. rqt_graph - один из классических инструментов, который позволяет визуализировать стртуру нод и топиков ros.

В случае данного проекта граф выглядит следующим образом:

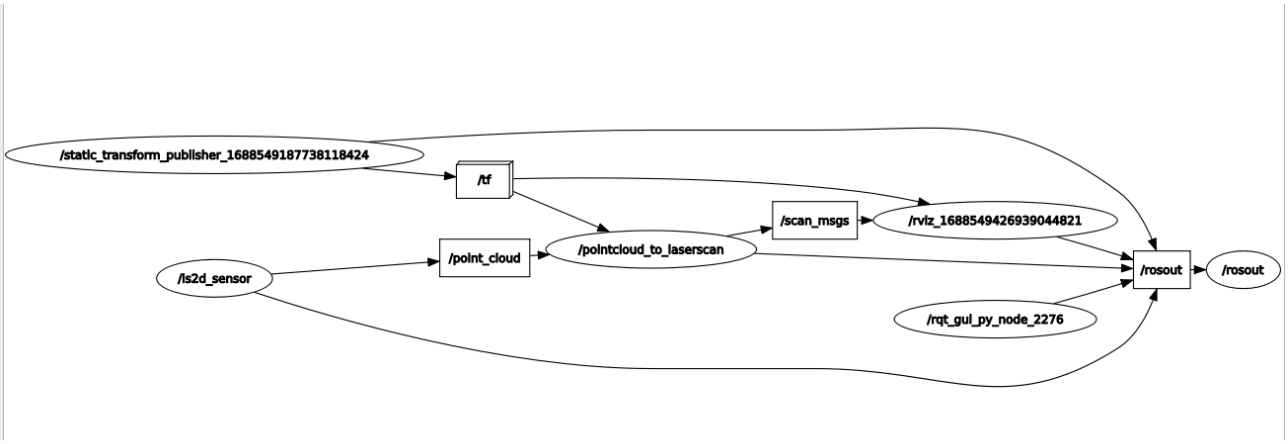

Рис. 4: rqt граф

Результат

В rviz можно визуализировать сообщения типа LaserScan из соответсвующего топика, результат работы пакета представлен на рисунке:

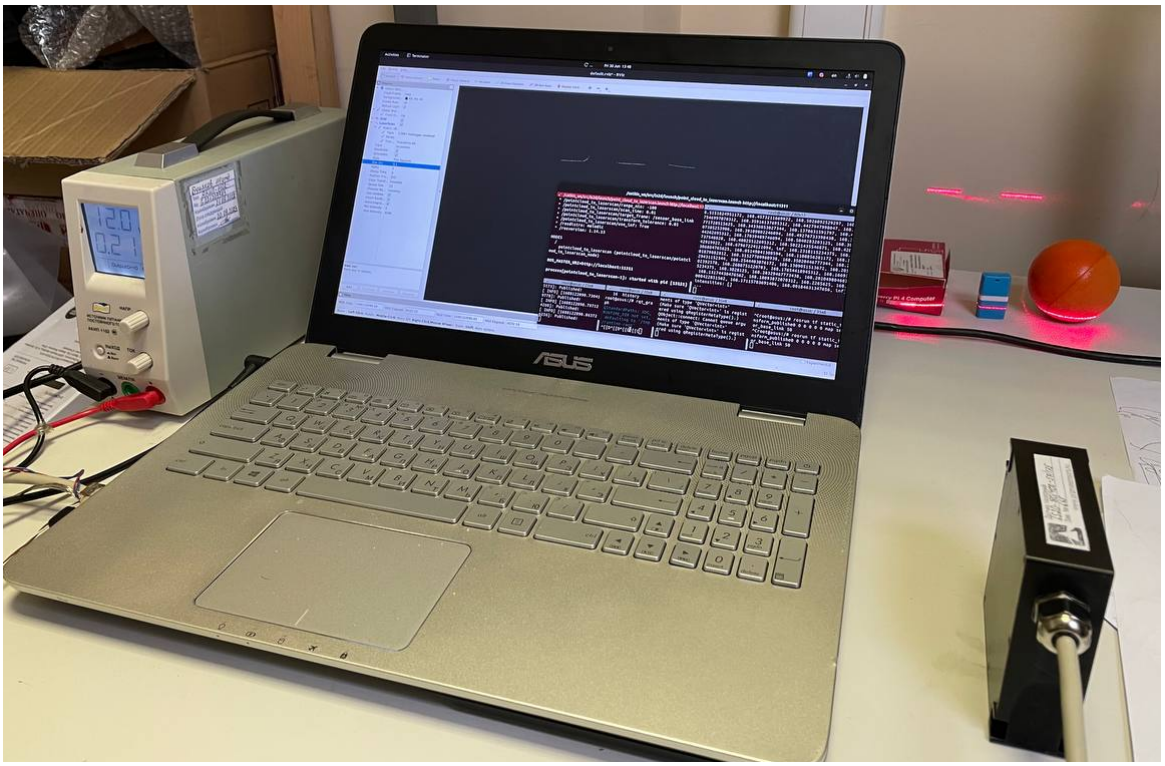

Рис. 5: Визуализация сканирования в rviz

3 Заключение

В результате работы был разработан проект получения данных с датчика в систему ros робота. В результате был получен опыт работы с реальным железом, ros1 и другими инструментами.

ROS представляет собой довольно развитый и популярный фреймворк в робототехническом стеке.

Используемые инструменты разработки

При решении задачи был получен опыт со следующими технологиями:

1. Docker
2. Robot operation system: работа с пакетами, сообщениями, rqt и средствами визуализации даннх
3. Язык программирования C++, библиотека стандартных шаблонов STL и многопоточность
4. Bash
5. Latex
6. Система сборки cmake
7. Система контроля версий git

4 Список литературы

1. Волков В.А. Конструктивные особенности робототехнических комплексов внутритрубной диагностики / В.А. Волков, В.В. Варлашин // Робототехника и техническая кибернетика. – Т. 10. - № 4. – Санкт-Петербург : ЦНИИ РТК. – 2022. – С. 309-320. – Текст : непосредственный.
2. Документация ROS. Электронный ресурс URL: `http:/ros.org`
3. Техническое описание датчика ls2d.

   Электронный ресурс URL: `http://prizmasensors.ru/files/teh-ls2d.pdf`