МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ЭнМИ
Кафедра РМДПМ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(бакалаврская работа)

Направление 15.03.06, Мехатроника и робототехника
(код и наименование)

Направленность (профиль) Компьютерные технологии управления в робототехнике и мехатронике

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Разработка бортовой системы визуальной одометрической навигации мобильного робота

Студент С-12-16 Кружков Е.П.
группа подпись фамилия и инициалы

Научный руководитель Канд. физ.-мат. наук, доцент Адамов Б.И.
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

«Работа допущена к защите»

Зав. кафедрой д.т.н. проф. Меркурьев И.В.
уч. степень звание подпись фамилия и инициалы

Дата 06.06.20

Москва, 2020

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ЭнМИ
Кафедра РМДПМ

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(бакалаврскую работу)

Направление 15.03.06, Мехатроника и робототехника
(код и наименование)

Направленность (профиль) Компьютерные технологии управления в робототехнике и мехатронике

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Разработка бортовой системы визуальной одометрической навигации мобильного робота

Студент С-12-16 Кружков Е.П.
группа подпись фамилия и инициалы

Научный руководитель Канд. физ.-мат. наук, доцент Адамов Б.И.
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Зав. кафедрой д.т.н. проф. Меркурьев И.В.
уч. степень звание подпись фамилия и инициалы

Место выполнения работы ФГБОУ ВО «НИУ «МЭИ»

СОДЕРЖАНИЕ РАЗДЕЛОВ ЗАДАНИЯ И ИСХОДНЫЕ ДАННЫЕ

П.1: Обзор современного состояния, имеющихся решений, исследований, разработок, публикаций

П.2: Исследование *SLAM*-метода *RTAB-Map*, калибровка стереокамеры в *Matlab* и *RTAB-Map*, сравнение их подходов к калибровке

П.3: Запуск *RTAB-Map* в *ROS*, калибровка стереокамеры в *ROS* пакете *camera_calibration*, демонстрация локализации робота в неизвестом пространстве *SLAM*-методом *RTAB-Map*

ПЕРЕЧЕНЬ ГРАФИЧЕСКОГО МАТЕРИАЛА

Количество листов -

Количество слайдов в презентации 10

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Рейнхард Клетте, Компьютерное зрение. Теория и алгоритмы, издательство «ДМК», 2019

2. Peter Corke, Robotics, Vision and Control Fundamental algorithms, Second Edition, Springer, 2017

3. дубров Д.В., Система построения проектов CMake, учебник, Издательство Южного федерального университета, Ростов-на-Дону, 2015

4. Джозеф Лентин, Изучение робототехники с помощью Python, «ДМК», 2019

5. M. Labbe, F. Michaud, “RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation,” in Journal of Field Robotics, vol. 36, no. 2, pp. 416–446, 2019. (Wiley)

Примечания:

1. Задание брошюруется вместе с выпускной работой после титульного листа (страницы задания имеют номера 2, 3).
2. Отзыв руководителя, рецензия(и), отчет о проверке на объем заимствований и согласие студента на размещение работы в открытом доступе вкладываются в конверт (файловую папку) под обложкой работы.

АННОТАЦИЯ

Рассматривается *SLAM*-метод навигации мобильного робота с помощью стереокамеры *HBV-1714-2 S2.0*. Сравниваются подходы к калибровке стереокамеры в программных средствах *Matlab*, *RTAB-Map* и пакете *ROS*: *camera_calibration*. В качестве *SLAM*-метода исследуется открытая кроссплатформенная библиотека *RTAB-Map*. Операционная система *ROS* используется в качестве операционной системы робота.

СОДЕРЖАНИЕ.

ВВЕДЕНИЕ

В 2020 году будет отправлено более 100 млрд. посылок и это число может удвоиться к 2030 [1]. При существующей инфраструктуре доставка посылок имеет массу логистических трудностей. В частности, это проблема «последней мили» – доставка от склада до потребителя компаниям обходится дороже всего [2]. В больших городах благодаря развитой инфраструктуре возможна доставка до потребителя в тот же день или на следующий. Но даже обычная доставка в область представляет сложность, не говоря уже о сельской местности или труднодоступных регионах. Возрастают потери компании на перевозе посылки и упускается возможная прибыль от доставки по более удобным маршрутам. Данную проблему могут решить автономные роботы курьеры и летающие дроны, что подтверждается растущем на них спросом [3].

Например, *Amazon Prime Air* [4] – дочерняя компания *Amazon*, специализирующаяся на доставке посылок автономными роботами. Она занимается как разработкой летающих дронов, так и наземными автономными роботами, например, её робот *Amazon Scout* (рисунок 1.1) использует пешеходную зону для передвижения [5]. На текущий момент компания тестирует сервис в различных странах. В своих роботах они используют стереокамеры [6], что можно объяснить их малым весом, возможностью работы на больших и малых дистанция, независимостью от внешних условий.

Рисунок 0.1: Amazon Scout

«Газпром нефть» используют беспилотные летательные аппараты (БПЛА) и автомобили (рисунок 1.2) для доставки грузов, в частности, в труднодоступные удалённые месторождения, поиска нефти, контроля состояния трубопроводов, контроля хода строительства.

Рисунок 0.2: Беспилотные автомобили «Камаз», применяемые «Газпром нефть»

Хотя проекты по автономной доставке обладают разной спецификой, для надёжного их функционирования необходима встроенная в робота система локализации и навигации, которую он сможет использовать при отсутствии связи с внешним миром. В данной работе ставится задача по созданию для робота системы, которая позволит ему определять своё положение в пространстве, которое он раньше “видел”.

Так как робот должен выполнять работу, которую традиционно выполняет человек, то можно подсмотреть как человек ориентируется по местности: человек, используя карту местности, способен найти соответствия того, что на карте, и

того, что он видит. После этого он определяет куда ему двигаться и цикл повторяется.

В данной работе ставится гипотеза о том, что поставленную задачу можно решить визуальным методом *SLAM* со стереокамерой в качестве источника визуальной информации.

Дана постановка задачи и её требования, после чего в первой главе сделан обзор современного состояния, имеющихся решений, исследований, разработок, публикаций по данной задаче с учётом сформулированной гипотезы. В результате анализа материала первой главы заключён вывод, что гипотеза была верна: визуальный метод *SLAM* со стереокамерой решает поставленную задачу. Выбрана библиотека *RTAB-Map* [7] в качестве реализации *SLAM*. Обосновано преимущество использования стереокамеры в качестве сенсора.

Далее проведено исследование библиотеки *RTAB-Map*, произведена калибровка стереокамеры в *RTAB-Map* и *Matlab* [8], сравниваются их подходы к калибровке и результаты. Построена карта местности.

После успешного теста *RTAB-Map* на имеющемся оборудовании, в следующей главе произведена настройка и запуск программного обеспечения в среде, аналогичной среде робота. Произведена калибровка стереокамеры пакетом *camera_calibration* [9]. Произведено испытание определения положения робота при перемещении в пространстве.

В заключении даны выводы по использованным технологиям, рекомендации по использованию полученных результатов.

ПОСТАНОВКА ЗАДАЧИ

Задача: Разработка встроенной в робота системы локализации.

Описание: Компания решает разработать робота курьера, который расширит доступную для доставки область, снизит время доставки, увеличит количество доставок в день, уменьшит расходы компании, уберёт человеческий фактор.

Ставится задача разработать встроенную в робота систему локализации, которая позволит вернуться ему на точку запуска в случае потери связи с внешними системами.

Требования:

- Система локализации должна быть встроена в робота и работать полностью автономно
- Работа системы локализации не должна зависеть от работы каких-либо внешних по отношению к роботу устройств
- Система локализации должна быть способна работать как на больших, так и на малых расстояниях
- Система локализации должна обеспечить возможность перемещения робота в выбранную позицию
- Система локализации должна учитывать статические препятствия на пути робота.

1. ОБЗОР СОВРЕМЕННОГО СОСТОЯНИЯ

На сегодняшний день существует много разных реализаций *SLAM*-алгоритмов. В работе [10] предложено описание общей структуры классических методов *SLAM*. Среди причин применения метода указано невозможность получения роботом внешних сигналов. Это действительно так, но, кроме того, *SLAM*-методы способны выдавать гораздо большую точность и скорость работы, чем позиционирование по таким системам как *GPS*, например, в работе [11] показано, что *SLAM*-метод *RTAB*-Map может работать в режиме реального времени без ухудшения точности, которая зависит от точности стереосистемы и, если имеется, точности дополнительного источника одометрической информации, в том числе *GPS*, так как *RTAB-Map* способен работать с любыми её источниками или их комбинациями. Можно использовать фильтрацию для объединения информации из нескольких источников одометрии, что может увеличить точность вычислений и робастность.

В [12] рассмотрены некоторые базовые алгоритмы *SLAM* и приведено их сравнение, что позволяет ориентироваться в отличительных чертах *SLAM*-методов. А в [13] произведено сравнение некоторых алгоритмов *SLAM* из *ROS*, из которого видно, что вычислительная нагрузка *RTAB-Map* или других рассмотренных алгоритмов не увеличивается с размером поля.

Более широкое сравнение многих популярных *SLAM*-методов *ROS* приведено в [14], а также рассказано как был создан *RTAB-Map*, его особенности и отличия от других *SLAM*-методов. Так, например, среди рассмотренных методов с использованием лидара только *RTAB-Map* и *SegMatch* [15] могут быть использованы для одновременного сканирования множества локаций или применения роем роботов. Среди рассмотренных визуальных *SLAM*-методов оказалось, что, не считая *RTAB-Map*, только *MCPTAM* [16] и *RGBDSLAMv2* [17] не

зависят от размера пространства и обладают достаточной робастностью. А также из рассмотренных методов только *RTAB-Map* и *MCPTAM* могут работать со множеством различных сенсоров одновременно, а для робота важен полный обзор пространства. В [18] рассмотрены всенаправленные датчики, но для работы с ними нужно преобразовать информацию будто она получена со множества камер и далеко не всегда возможно найти на роботе место для установки, с которого открывается полный обзор на окружающее пространство, поэтому возможность получать данные с нескольких сенсоров разного типа одновременно очень удобна. По количеству предоставляемой пользователю на выходе алгоритма информации ближе всех к *RTAB-Map* находится *RGBDSLAMv2*. Но и *MCPTAM*, и *RGBDSLAMv2* уже несколько лет не получают обновлений в своём репозитории на *GitHub*.

В [19] рассмотрена работа *RTAB-Map* на протяжении длительного времени при автономном патрулировании роботом пространства в помещении, за время которого он проехал 10.5 км, встречая на своём пути динамические препятствия (139 людей за всю поездку). Показано, что мощность необходимых вычислений не изменяется с течением времени, но увеличивается количество потребляемой алгоритмом памяти.

Для применения *RTAB-Map* очень удобно воспользоваться операционной системой *ROS* [20], но необходимо убедиться, что её использование в задаче возможно. *ROS* – операционная система, устанавливаемая поверх другой операционной системы, обычно поверх *Linux*, но ведутся разработки по внедрению её на *Windows*. *ROS* позволяет абстрагироваться от аппаратного обеспечения, поэтому программный код будет работать на любом железе без изменений. В работах [21] и [22] рассмотрена *ROS*, показан принцип её работы и преимущества. Но в них не сказано о недостатках *ROS*, так, важным её недостатком является её сложность. Хотя *ROS* сильно упрощает создание робототехнических систем, но нужно потратить много времени, чтобы

разобраться в том, как использовать *ROS* и может быть трудно обойтись без навыка компилирования программ с использованием таких средств как *CMake* [23]. Вторым её недостатком является всё ещё очень сильная привязанность к *Linux*. И ещё одним недостатком *ROS* является сложность его использования конечным пользователем, конечно, можно создать специальный интерфейс, но в таком случае может быть проще реализовать решение с чистого листа без использования *ROS*. Нельзя забывать и об открытости исходного кода *ROS*, что может быть приемлемо не во всех случаях.

Для текущей задачи единственным недостатком *ROS* является сложность её освоения, которая компенсируется преимуществами, поэтому для робота выбрана конфигурация *ROS + RTAB-Map*.

Также необходимо рассмотреть вопрос аппаратного обеспечения робота. Преимуществом *RTAB-Map* является в среднем постоянная скорость вычислений не зависимо от размера карты, что позволяет удовлетворить всем поставленным в задании требованиям. Библиотека поддерживает датчики любого типа, но робот должен быть оснащён хотя бы одним *RGB-D* датчиком (стереокамера, *Kinect, Asus X-tion* и др.). Необходимость использования *RGB-D* датчика не является проблемой в данном случае, так как работа алгоритма необходима и в закрытых и в открытых пространствах, то лучшим решением для такого случая является стереосистема. Дело в том, что стереокамерам не мешает солнечный свет, в отличие от камер с инфракрасной подсветкой типа *Kinect* и они меньше стоят и весят по сравнению с лидарами. А также, в отличие от последних, стереокамеры можно применять на летающих над препятствиями роботах. Хороший обзор сенсоров, используемых для определения расстояния дан в [24]. В [25] приведены оценки точности определения глубины недорогой стереокамерой, сделан вывод, что выбранную камеру можно использовать для навигации роботов.

Касаемо аппаратной части, авторы *RTAB-Map* не указывают жёстких системных требований для работы библиотеки, но говорят, что для визуализации рекомендуется дискретная видеокарта, *SSD* и хотя бы 6 – 8 гигабайт оперативной памяти. На низкой скорости алгоритм может работать и на *Raspberry Pi* 3.

Так, например, в качестве контроллера для робота подойдут встраиваемы системы *NVIDIA Jetson* [26], оптимизированные для работы с компьютерным зрением. Их возможности сильно превосходят требуемые и оставляют пространство для встраивания искусственного интеллекта.

Таким образом, выбран алгоритм *RTAB-Map* для решения поставленной задачи, определено необходимое аппаратное и программное обеспечение.

2. ИССЛЕДОВАНИЕ БИБЛИОТЕКИ RTAB-MAP

RTAB-Map (*Real-Time Appearance-Based Mapping*) – это кроссплатформенная библиотека с открытым исходным кодом, реализующая графовый метод *SLAM*.

RTAB-Map можно использовать как *ROS* пакет на *Linux*, как *stand-alone* приложение (на *Linux* и *Windows*) или как *C++* библиотеку (на *Linux* и *Windows*).

Наиболее простой и дающий больше всего возможностей способ запуска библиотеки – запуск в *ROS*. *ROS* и *ROS* 2 в последнее время помимо работы на *Linux*, получили также возможность устанавливаться и работать на *Windows* – это направление сейчас активно развивается и поддерживается разработчиками *ROS* и *Microsoft*. Тем не менее хотя *ROS* и предоставляет абстрагированный от аппаратной части интерфейс, для работы пакетов *ROS* на *Windows* необходима их компиляция, отличная от имеющихся скомпилированных пакетов для *Linux*. Таким образом на текущий момент больше всего пакетов доступно для *ROS* на *Linux*. Несколько меньшее пакетов доступно на *ROS* 2 на *Linux*, так как хотя для *ROS* 2 их тоже нужно перекомпилировать, но операционная система (*Linux*) остаётся той же. На *Windows* же наблюдается большой дефицит готовых пакетов, хотя и возможно своими силами перекомпилировать исходный код для работы на *Windows*, но для этого необходимы знания в данной области и, в частности, сложность в том, что у пакетов могут быть уникальные для операционной системы зависимости, в таком случае для системы, у которой этих зависимостей нет, нужно искать аналоги и вручную делать соответствующие изменения, в частности, в исходном коде.

Среди многих пакетов *ROS*, отсутствующих в версии для Windows, находится и *RTAB-Map*. Так как робот, для которого разрабатывается программное решение, предполагается, будет работать на *Linux*, то данная ситуация не вызывает никаких проблем. Но до начала сборки робота, на этапе

анализа ПО придётся искать решение данной проблемы. В моём случае при попытке установки *Linux* как второй ОС выяснилось, что производитель компьютера (*Maibenben*) не поддерживает ОС *Linux*, и на попытки её установить компьютер выдавал только различные ошибки. Таким образом для исследования *RTAB-Map* в этой главе я выбрал *stand-alone* приложение на *Windows*. А для решения поставленной задаче в среде, аналогичной среде робота, в главе 2 работа продолжена в эмуляторе *VirtualBox* [27] на ОС *Ubuntu* 16.04 [28].

Для работы с библиотекой удобно использовать предоставляемый её графический интерфейс. На рисунке 3.1 он представлен в процессе работы программы. В первом окне (окно в левом верхнем углу) отображается текущее изображение с одной из камер стереокамеры. Как можно заметить, верх и низ изображений имеют необычную кривую форму – это связано с тем, что они проходят процесс ректификации. Под ним находится окно, которое сообщает найдено ли место, которое робот видел прежде (зелёный цвет вокруг значит, что найдено). Ниже находится окно *Odometry* с жёлтыми, зелёными и красными точками на изображении. Эти точки используются для вычисления визуальной одометрии робота, а их цвет соответствует их надёжности. Справа находится окно *3D Map*, на котором представлено всё снятое пространство в *3D*, а под ним окно *Graph view*, на котором находится сечение *3D* пространства выбранной плоскостью. Чёрные точки на нём представляют непроходимые для робота препятствия, если он движется по плоскости, белые – свободное для перемещения пространство и серая область – неизвестная зона.

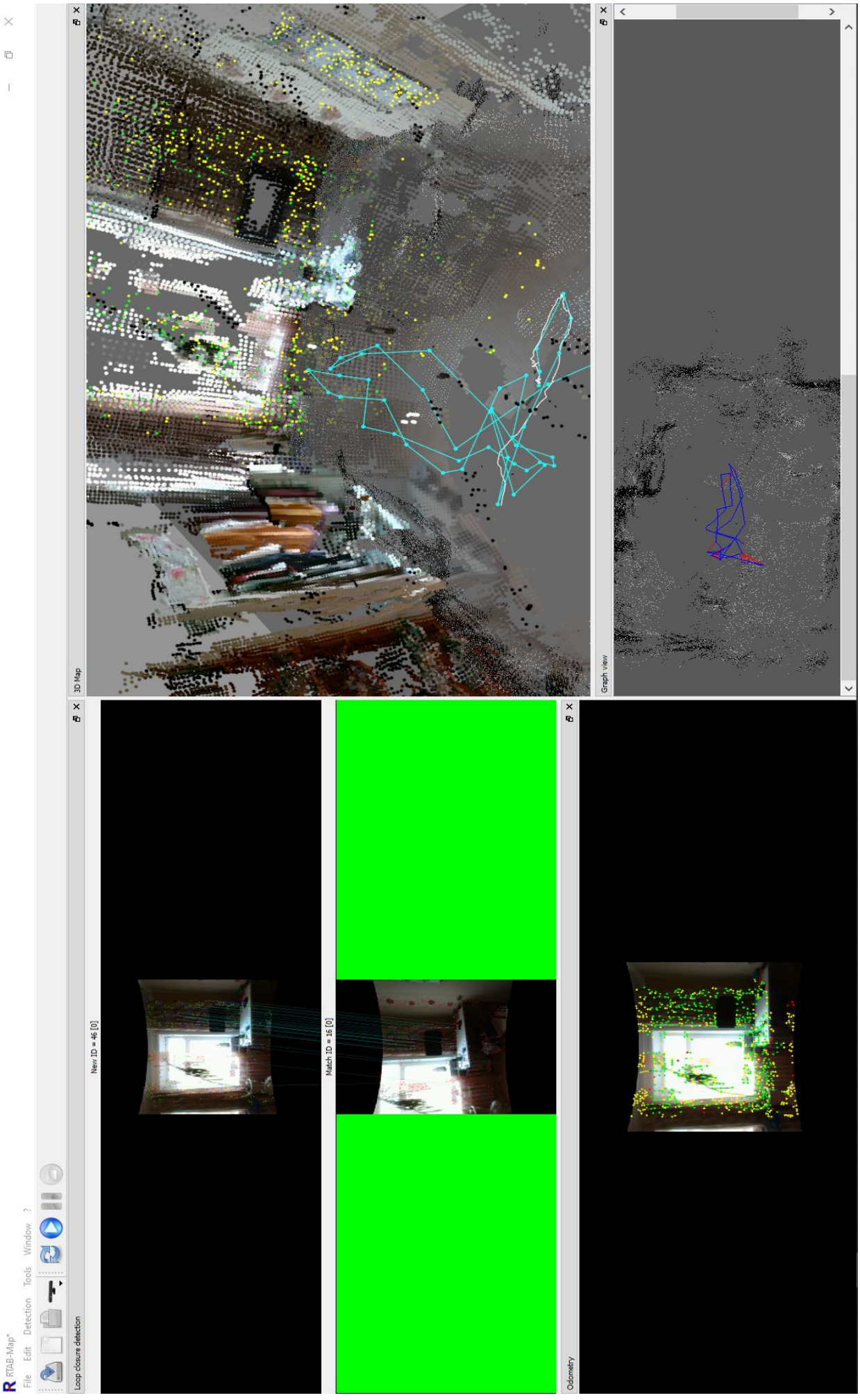

Рисунок 2.1: RTAB-Map в процессе работы

На рисунке 3.2 представлено ещё одно важное окно *RTAB-Map*. В данном окне видна различная статистическая информация, обновляемая в реальном времени. Из рисунка 3.2 видно, что *RTAB-Map* запомнил уже 14400 точек для идентификации сцен (*Dictionary size* 1.44e+4 *words*).

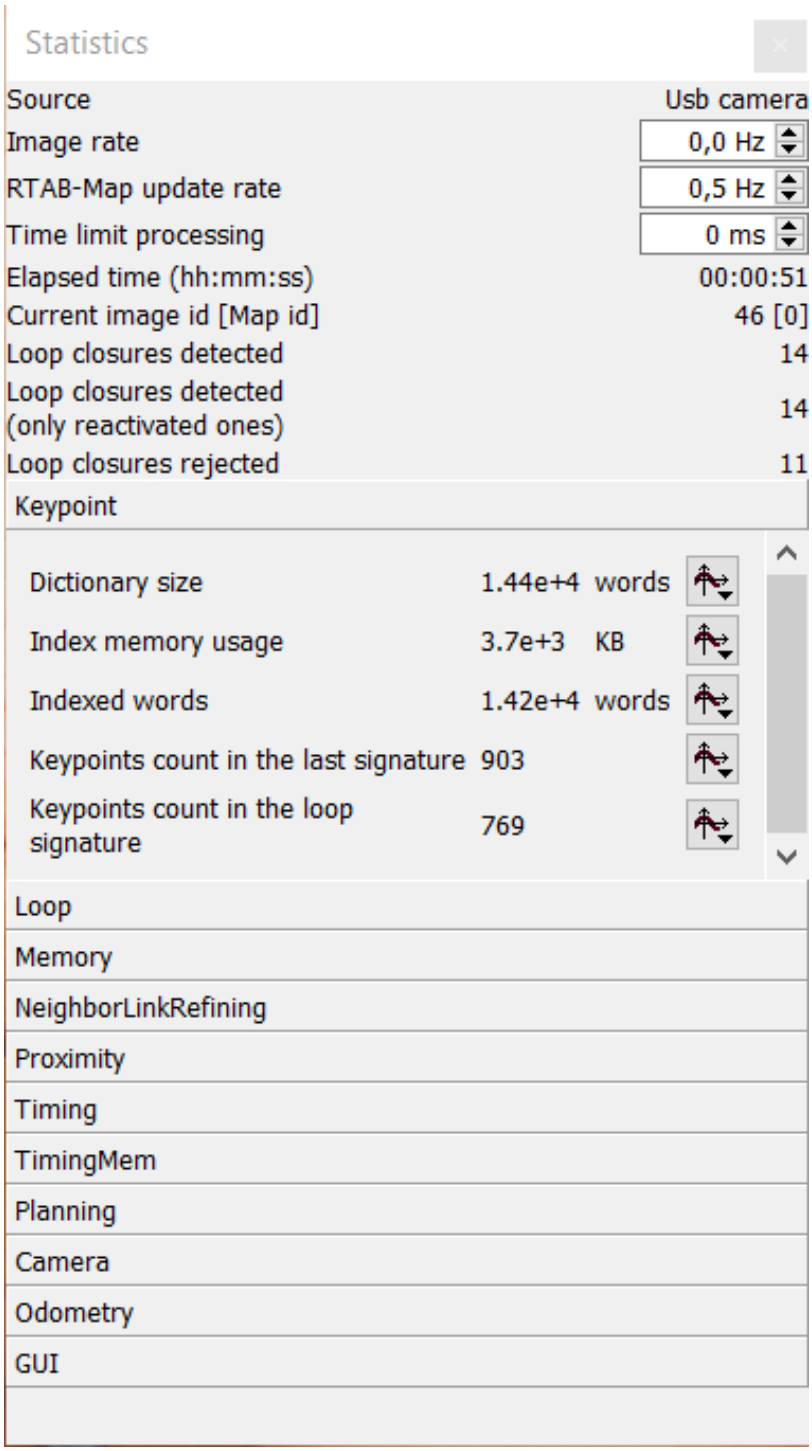

Рисунок 2.2: Окно с собранными RTAB-Map статистическими данными

Любую учитываемую статистическую информацию можно получить в виде графика, для наглядности того, как они меняются с течением времени. Например, на рисунке 3.3 представлено как меняется время, за которое *RTAB-Map* производит свои вычисления.

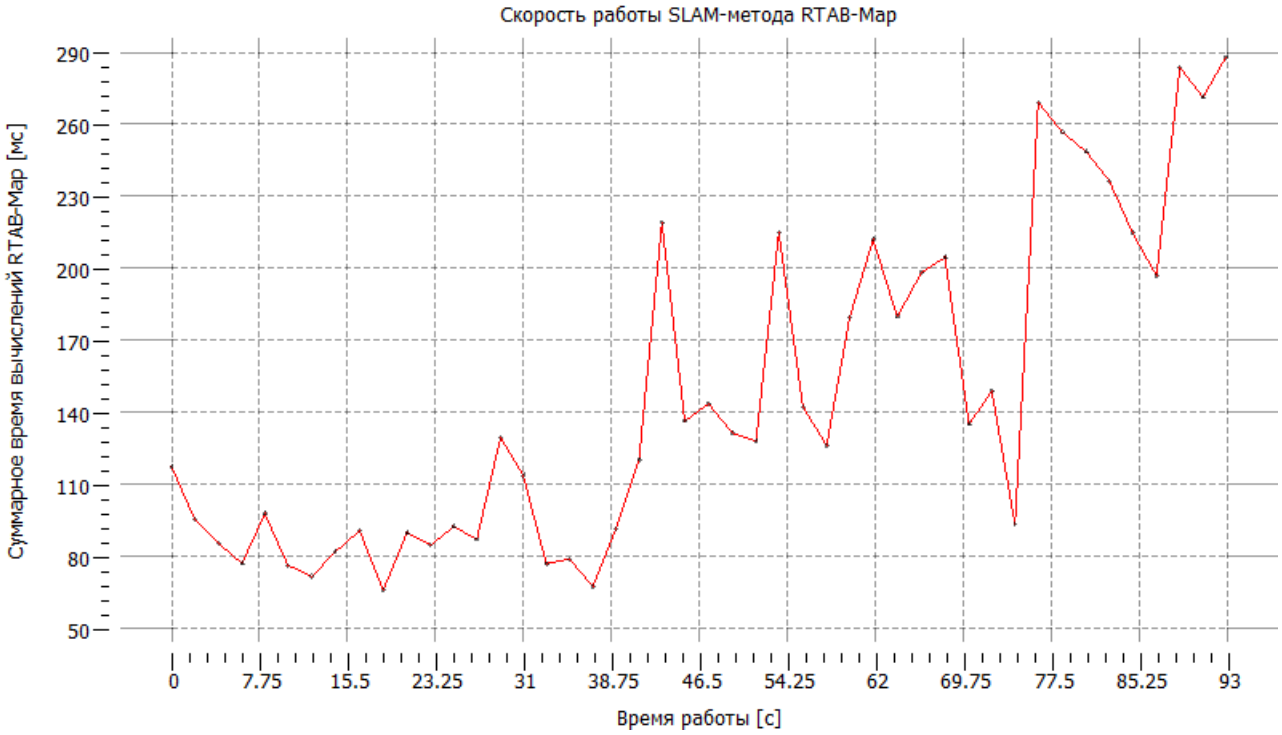

Рисунок 2.3: Производительность RTAB-Map с отображением 3D графики

Как видно, время, затрачиваемое на вычисления, до 40 секунды колеблется около некоторого среднего положения – такое поведение и ожидается, так как *RTAB-Map* рассчитан на работу онлайн и скорость его вычислений не должна снижаться с течением времени, независимо от размера рабочей области.

Но после становится заметно, как увеличивается время вычислений, также это становится ощутимо при работе программы, она начинает подвисать. Причина данной проблемы оказывается в огромном количестве точек, которое графическая система пытается вывести в пользовательский интерфейс (окна *3D Map* и *Graph*

view). На рисунке 3.4 показано, как увеличивается время, необходимое для вывода *3D* информации на экран.

Рисунок 2.4: Ресурсы, потребляемые отображением 3D графики

Становится понятным, что с определённого момента времени графическая подсистема становится перегруженной. Для решения данной проблемы следует уменьшить нагрузку на графическую подсистему. Так как во время автономной работы робота вывод *3D* карты не требуется, то был проведён эксперимент при закрытых окнах *3D Map* и *Graph view*, результаты представлены на рисунке 3.5. В процессе тестирования не замечено ухудшение скорости работы, что видно из графика: в большинстве случаев библиотеке требовалось менее 0.1 секунды на вычисления. В некоторых моментах можно увидеть увеличение времени вплоть до 0.4 секунды, что может быть вызвано поиском замыканий на графе, но *RTAB-Map* всё равно возвращается к вычислениям порядка 0.1 секунды. В работе [11] приведено сравнение скорости работы *RTAB-Map* с ограничением времени на операции и без, из которого следует, что все вычислительные операции при

увеличении длительности эксперимента продолжат выполняться за приемлемое время.

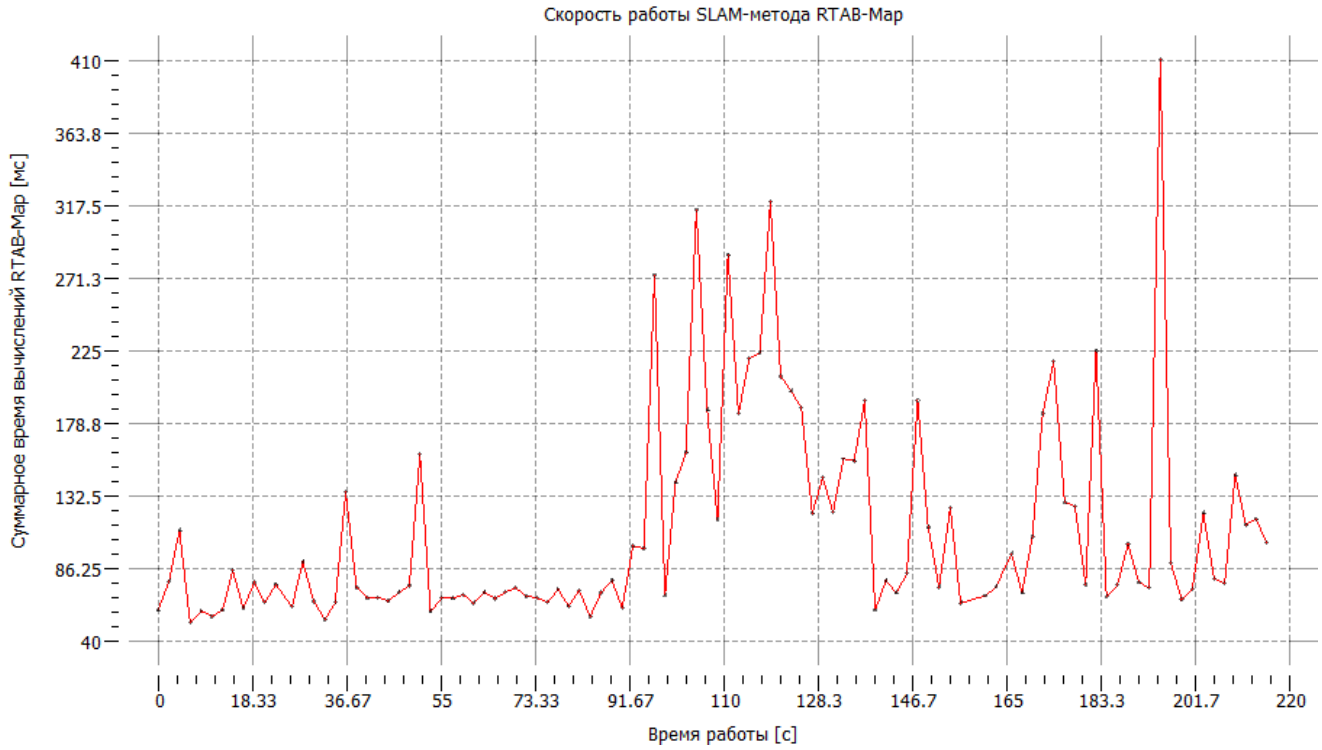

Рисунок 2.5: Производительность RTAB-Map без отображения 3D графики

Основная часть *RTAB-Map* находится в выделенной пунктирной прямоугольной части (*rtabmap_ros/rtabmap*) на рисунке 3.6.

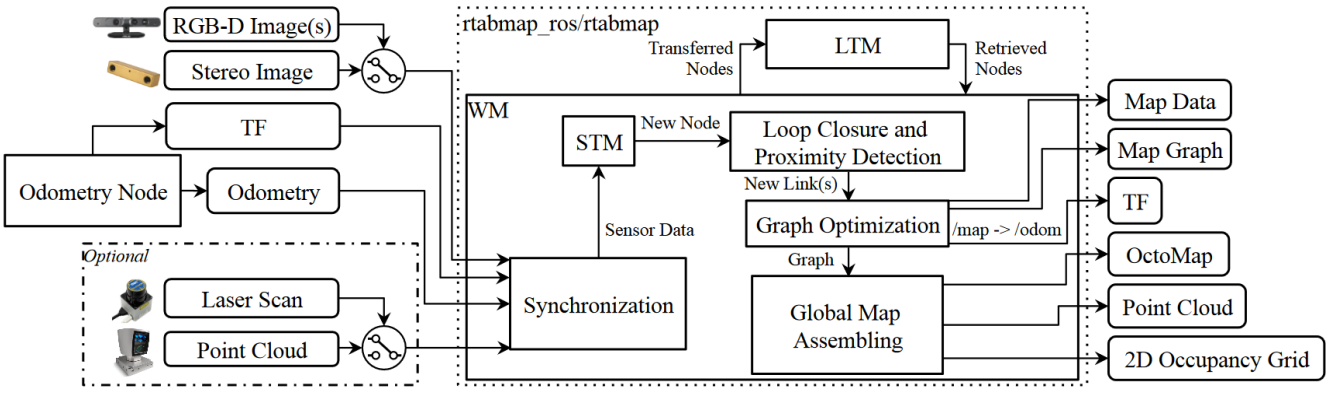

Рисунок 2.6: Модульная структура SLAM-метода RTAB-Map

Рисунок 3.6 хорошо показывает, что в *RTAB-Map* входят модули контроля памяти (*STM* и *LTM*), модуль приёма данных и их синхронизации (*Synchronization*), а также модули для построения графа и работы с ним (*Global*

Optimization, Loop Closure and Proximity Detection). Таким образом в своей основе *RTAB-Map* представляет способ хранения и обновления данных. Такие элементы как модули считывания данных, получения из данных одометрической информации, пространственной или любой другой, модули работы с сенсорами робота не являются частью библиотеки, а лишь распространяются вместе с ней для удобства работы, легко могут быть заменены на пользовательские и делают из *RTAB-Map* полноценную *SLAM*-библиотеку.

Стоит отметить, что для своей работы *RTAB-Map* перед запуском создаёт базу данных или открывает существующую (на выбор пользователя), а по завершению работы базу данных можно сохранить для продолжения работы при следующем запуске или оффлайн анализа, обработки.

Как следует из рисунка 3.6, *RTAB-Map* как *SLAM*-метод в своём составе имеет множество элементов, поэтому помимо задания параметров для основы *RTAB-Map*, необходимо также задать параметры остальных компонентов, таких как: настроить подключение к необходимому датчику, при необходимости, откалибровать его, задать параметры алгоритмов, обрабатывающих информацию с датчика (например, обрабатывающих стереоизображения и вычисляющих карту глубины), выбрать и настроить источник одометрической информации и обработку информации от него, выбрать и настроить алгоритм поиска особенностей изображений, а также определиться с дополнительными настройками, отвечающими за вывод информации пользователю. Настройки не обязательно вбивать вручную, можно указать их при запуске робота, что будет показано при запуске *RTAB-Map* в *ROS*.

На рисунке 3.7 представлено дерево настроек *RTAB-Map* в ветке *Source* (ветка настроек датчика). Для первого запуска достаточно настроить только подключение стереокамеры, а остальные параметры оставить по умолчанию.

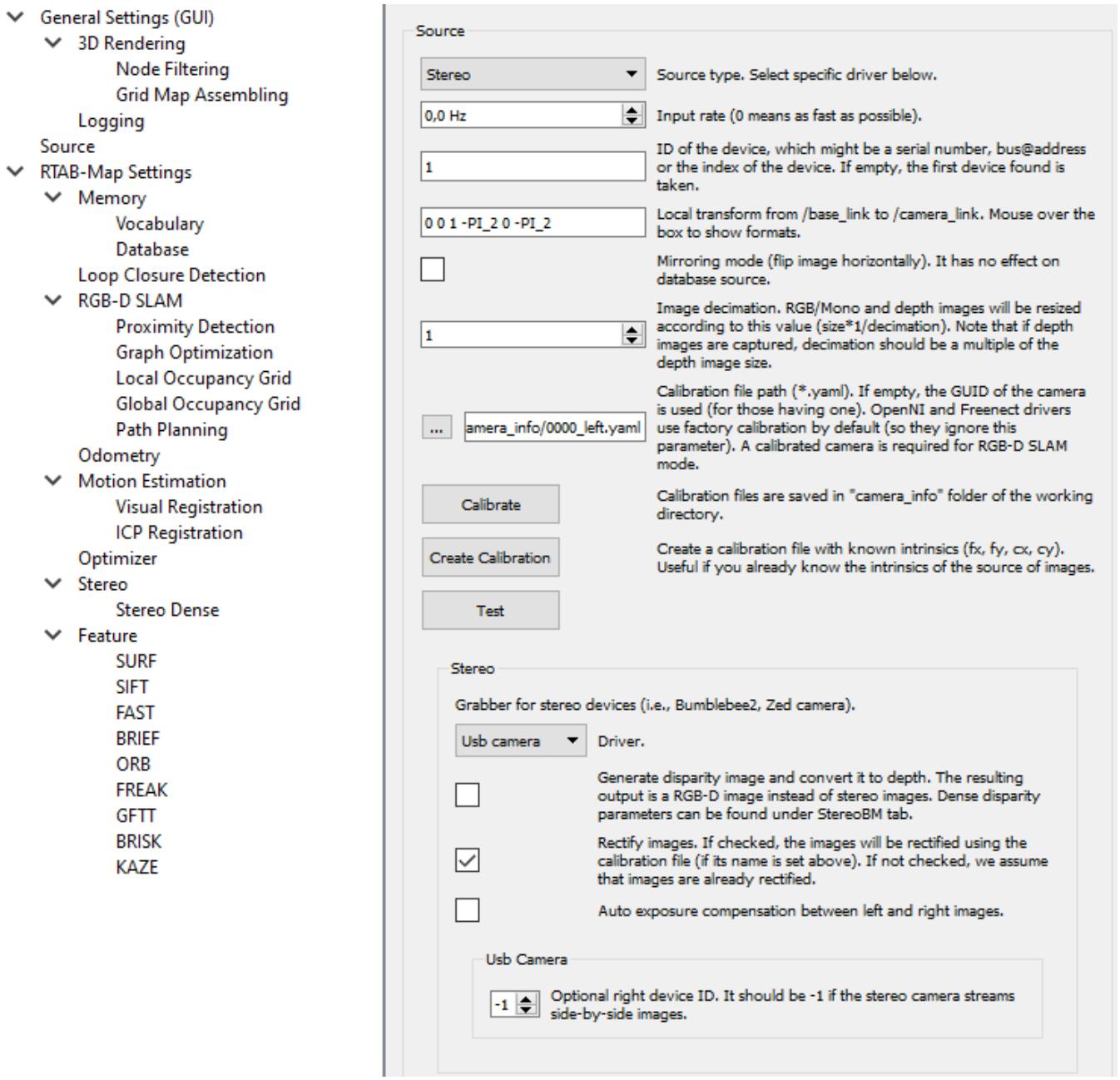

Рисунок 2.7: Настройки RTAB-Map. Подменю Source

На рисунке 3.7 установлен тип сенсора: стереокамера. Значение (-1) в параметре *right device ID* означает, что изображения с левой и правой камеры приходят склеенными в одно.

Используемая стереокамера *HBV-1714-2 S2.0* (рисунок 3.8) работает как обычная *USB*-камера.

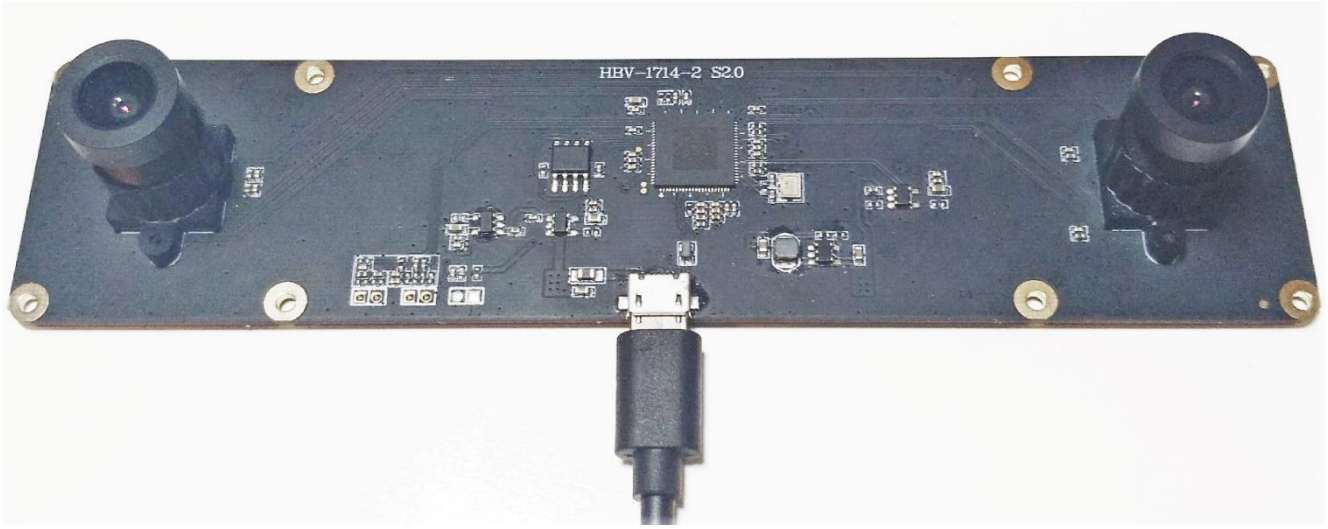

Рисунок 2.8: Используемая стереокамера HBV-1714-2 S2.0

При подключении стереокамеры к *RTAB-Map* на *Windows*, максимальное разрешение, которое удалось получить, составляет всего 320 на 480 пикселей для каждой камеры, а при работе с ней на *ROS*, запущенной в виртуальной среде *VirtualBox*, через пакет для *UVC* камер удалось получить разрешение 1280 на 960 пикселей для каждой камеры.

На рисунке 3.9 представлен пример изображения, снятого на стереокамеру. Как видно, оно сильно искажено: присутствует заметная бочкообразная дисторсия. На изображениях не должно быть никаких искажений и горизонтальные линии на изображениях с левой и правой камеры должны проходить через одни и те же признаки (*features*) на обоих изображениях для корректной работы системы стереозрения.

Видно, что горизонтальные линии не проходят через одинаковые признаки изображений: самая верхняя линия проходит через начало координат на левом изображении, но не проходит через него на правом.

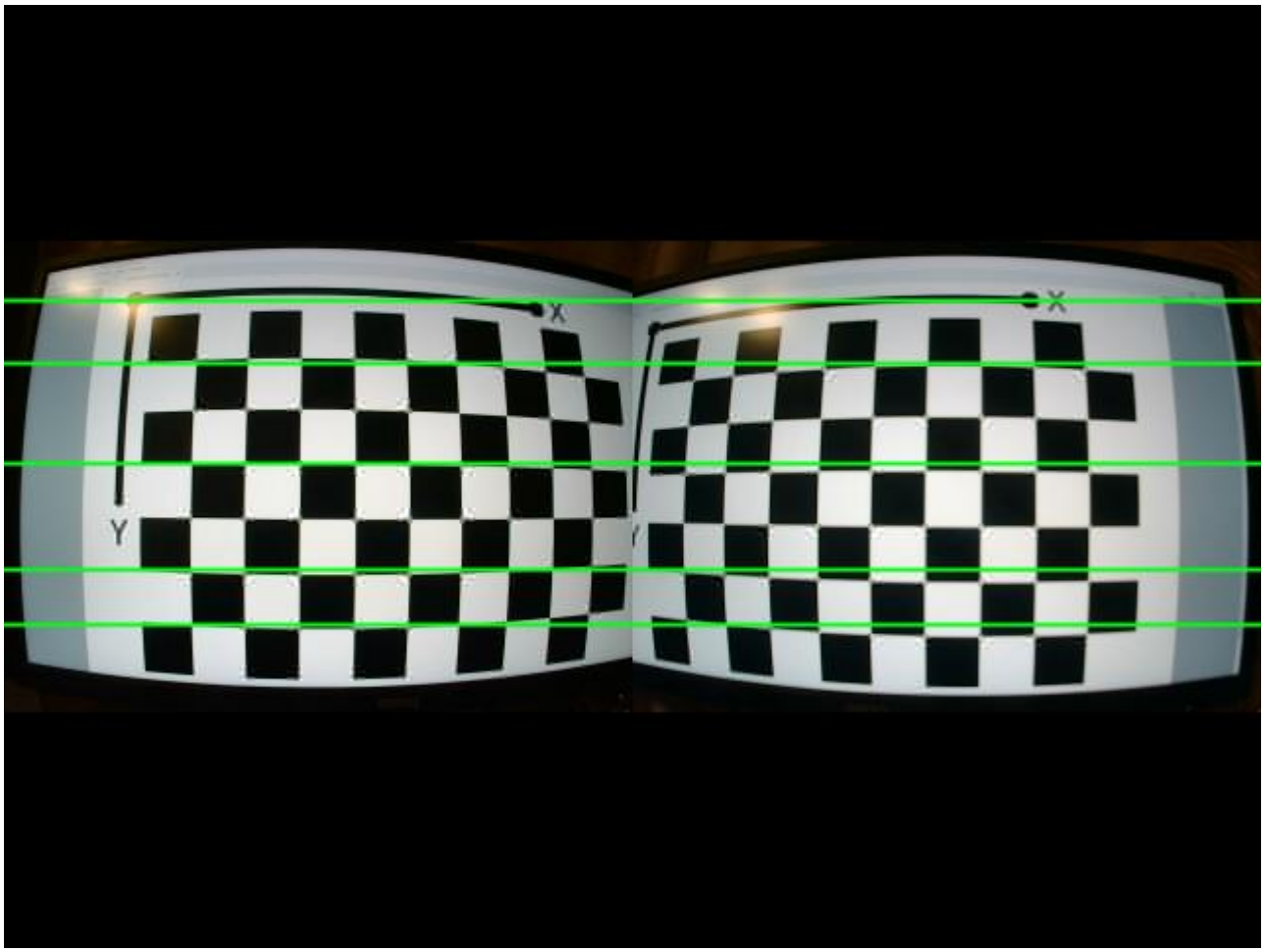

Рисунок 2.9: Изображение со стереокамеры до её калибровки

Чтобы убрать искажения с изображения, необходима калибровка стереокамеры. Для калибровки используется прямоугольный паттерн в виде шахматной доски. Нужно распечатать его и приклеить на ровную поверхность. Алгоритму калибровки указываются размеры доски по горизонтали и вертикали в клетках и размер клеток (клетки квадратные).

С точки зрения точности калибровки, чем больше калибровочная доска заполняет изображение, тем лучше. Важной особенностью калибровки (и работы) стереокамеры от калибровки одной камеры является обязательность синхронизации камер стереокамеры во времени.

В результате калибровки вычисляются внутренние и внешние параметры стереокамеры, которые используются для ректификации изображений и извлечения из них информации о глубине сцены.

В *RTAB-Map* калибровку можно произвести кнопкой *Calibrate* (рисунок 3.7). Для калибровки в *RTAB-Map* нужно передвигать камеру напротив паттерна, в то время как программа выводит в процентах на сколько камера откалибрована. Когда она решит, что снято достаточно данных, то будет доступна кнопка, по нажатию на которую начнётся процесс вычисления параметров стереокамеры.

У меня данный процесс ни разу не завершился успехом: калибровка занимала долгое время, а программа не начинала считать, что данных достаточно, хотя для каждой из камер стереокамеры было сделано порядка 100 снимков. Одной из особенностей калибровки в *RTAB-Map* является то, что она может снять разное количество калибровочных изображений для левой и правой камеры. Вероятно, неудача калибровки вызвана слишком маленьким размером калибровочной доски для выбранной стереокамеры, у которой расстояние между камерами составляет 120 мм, и её низким разрешением 320 на 480 пикселей для каждой камеры.

Тем не менее, *RTAB-Map* позволяет произвести вычисления параметров принудительно, не дожидаясь разрешения калибровочной программы. Поэтому был проведён повторный эксперимент с принудительным началом вычислений после снятия небольшого количества снимков. Результат представлен на рисунке 3.10. Видно, что был выпрямлен монитор, то есть прямые линии стали отображаться на изображения в прямые линии и, что горизонтальные линии проходят через одни и те же клетки доски на обоих изображениях.

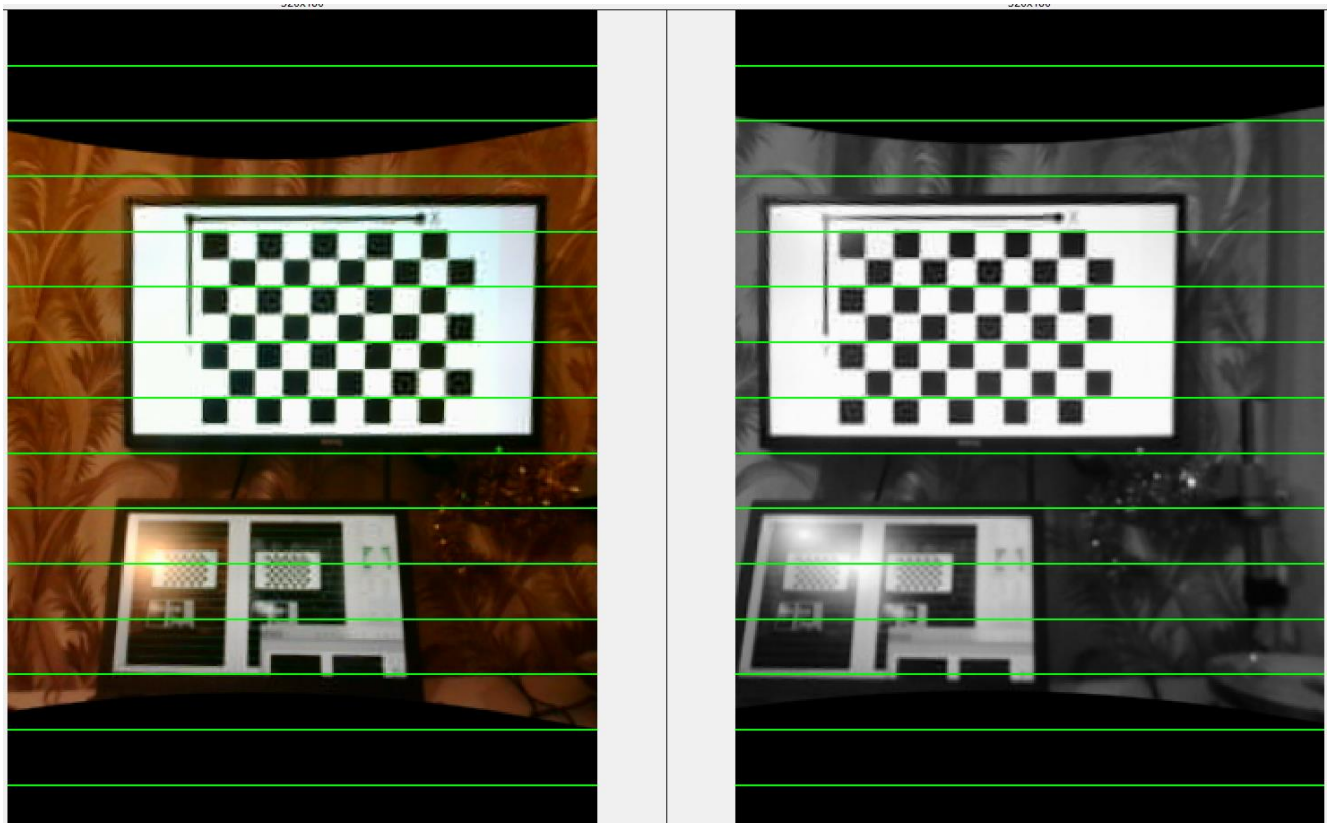

Рисунок 2.10: Изображение со стереокамеры после калибровки

Результаты калибровки сохраняются в файлах **_left.yaml*, **_right.yaml*, **_pose.yaml*, содержащих соответственно параметры левой и правой камеры и их относительное положение.

Вероятно, можно добиться лучшего результата с увеличением размера доски и разрешающей способности камеры.

От точности калибровки стереокамеры сильно зависит точность построения карты и определения положения. С целью получить лучшие результаты, произведено исследование параметров стереокамеры в *Matlab*. Для калибровки стереокамеры в *Matlab* нужно заранее снять 10-20 [29] изображений калибровочной доски и передать их программе для вычисления параметров стереокамеры.

Для снятия изображений со стереокамеры на языке *Python* написан код [30] (приложение 1), захватывающий изображения со стереокамеры и сохраняющий

их в файлы. Для захвата использована библиотека *OpenCV* [31]. Максимальное разрешение, которое удалось получить таким способом не изменилось: 320 на 480 пикселей для каждой камеры. Для эксперимента было создано 20 наборов изображений по 20 снимков в каждом наборе.

Изображения для всех наборов снимались при одинаковом освещении и прочих равных условиях, расстояние от стереокамеры до доски во всех наборах не превышает одного метра (на таком расстоянии были получены лучшие результаты в предыдущих экспериментах для данного разрешения стереокамеры и размера калибровочной доски, но по рекомендации [29] расстояние должно соответствовать расстоянию, на котором будет находиться стереокамера от объекта измерения).

Ожидается, что на всех 20 наборах изображений калибровка даст приблизительно одинаковые параметры камеры, взяв среднее [32] по которым, можно будет получить параметры, хорошо приближенные к действительным.

Калибруя стереокамеру на каждом наборе независимо, получены параметры камеры для каждого набора. Для калибровки на языке программирования *Matlab* написан код [33] (приложение 2, приложение 3). На рисунках 3.11 – 3.15 показано как изменяются вычисленные параметры левой и правой камеры для каждого набора изображений. Пунктирными линиями обозначены средние значения для соответствующих величин. Представленные на рисунках 3.11 – 3.15 средние значения вычислены с учётом всех наборов изображений.

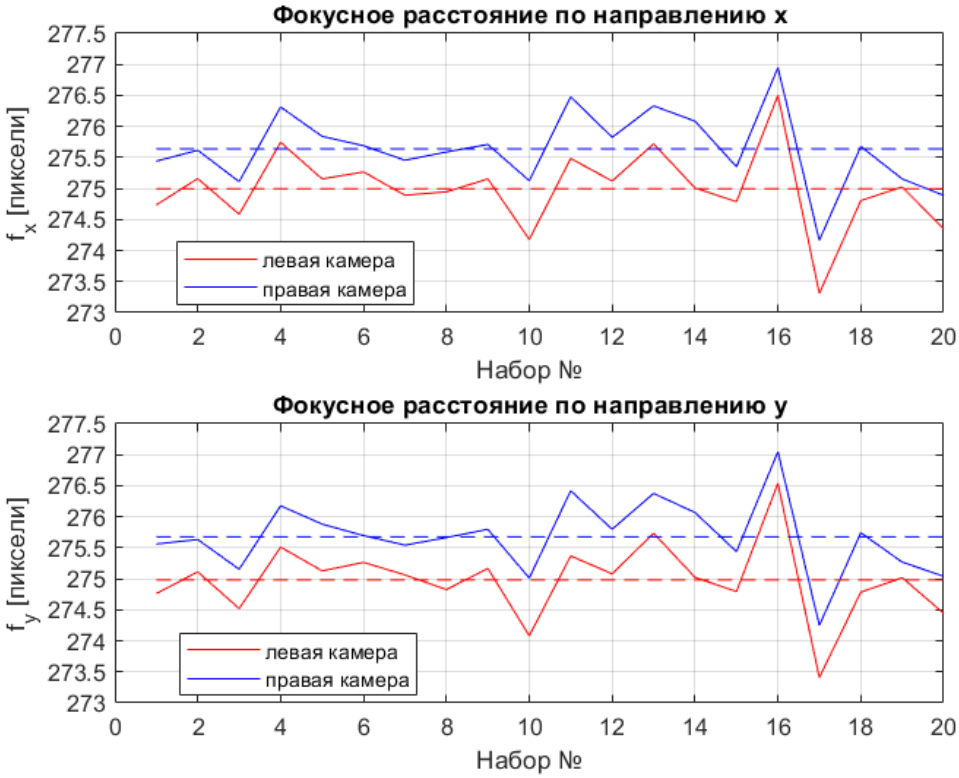

Рисунок 2.11: Вычисленные в Matlab фокусные расстояния камер

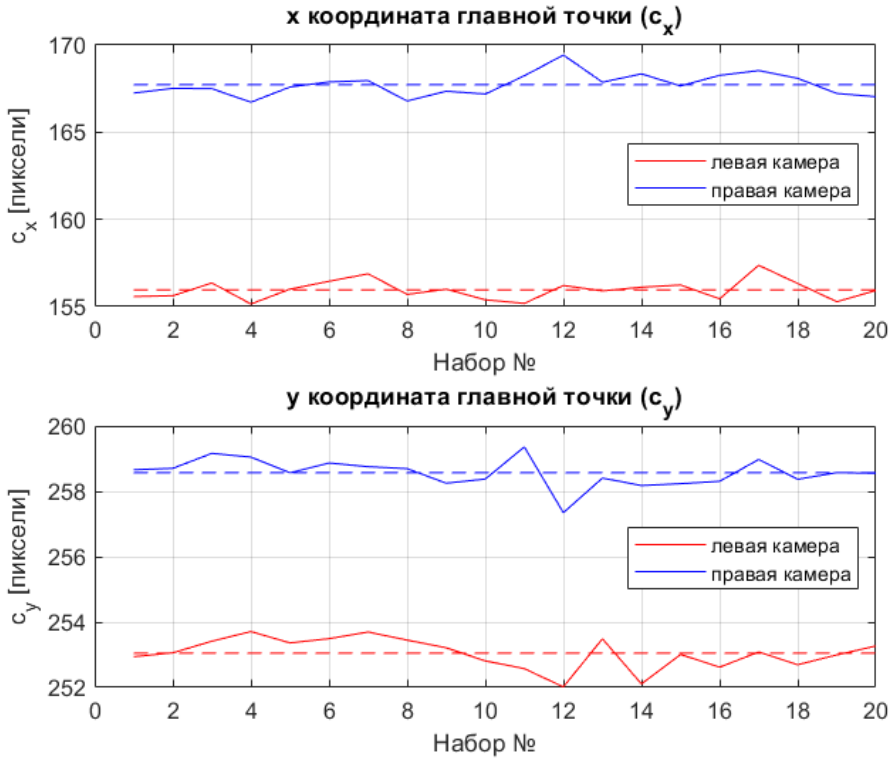

Рисунок 2.12: Вычисленные в Matlab координаты главной точки изображений

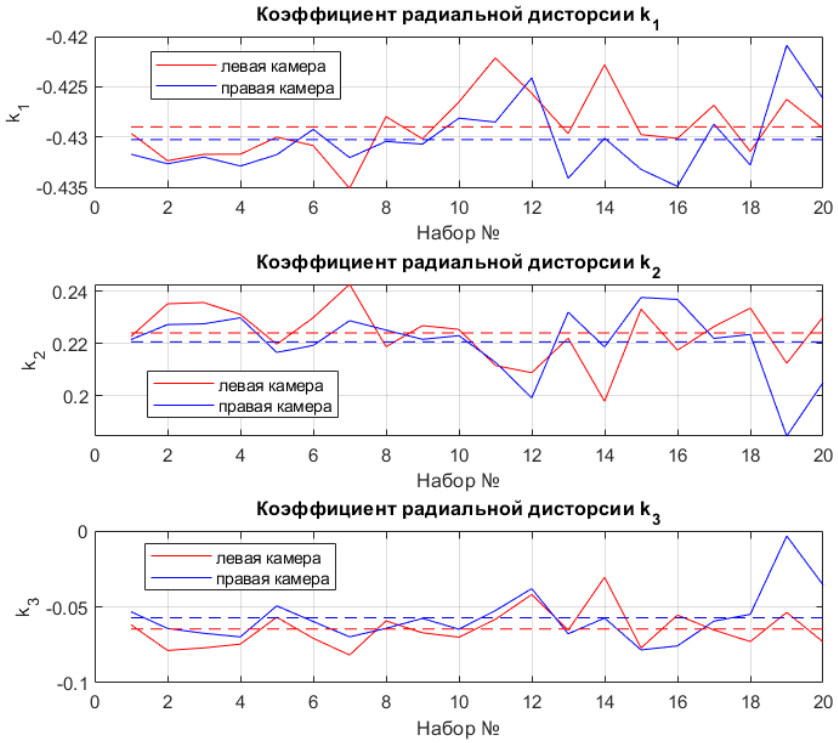

Рисунок 2.13: Вычисленные в Matlab коэффициенты радиальной дисторсии

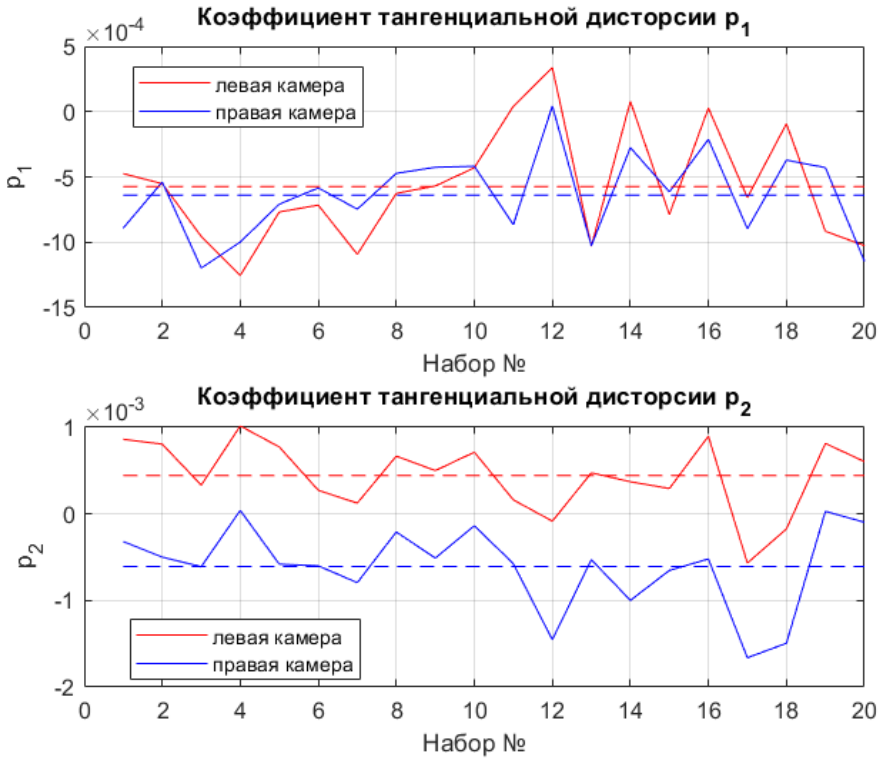

Рисунок 2.14: Вычисленные в Matlab коэффициенты тангенциальной дисторсии

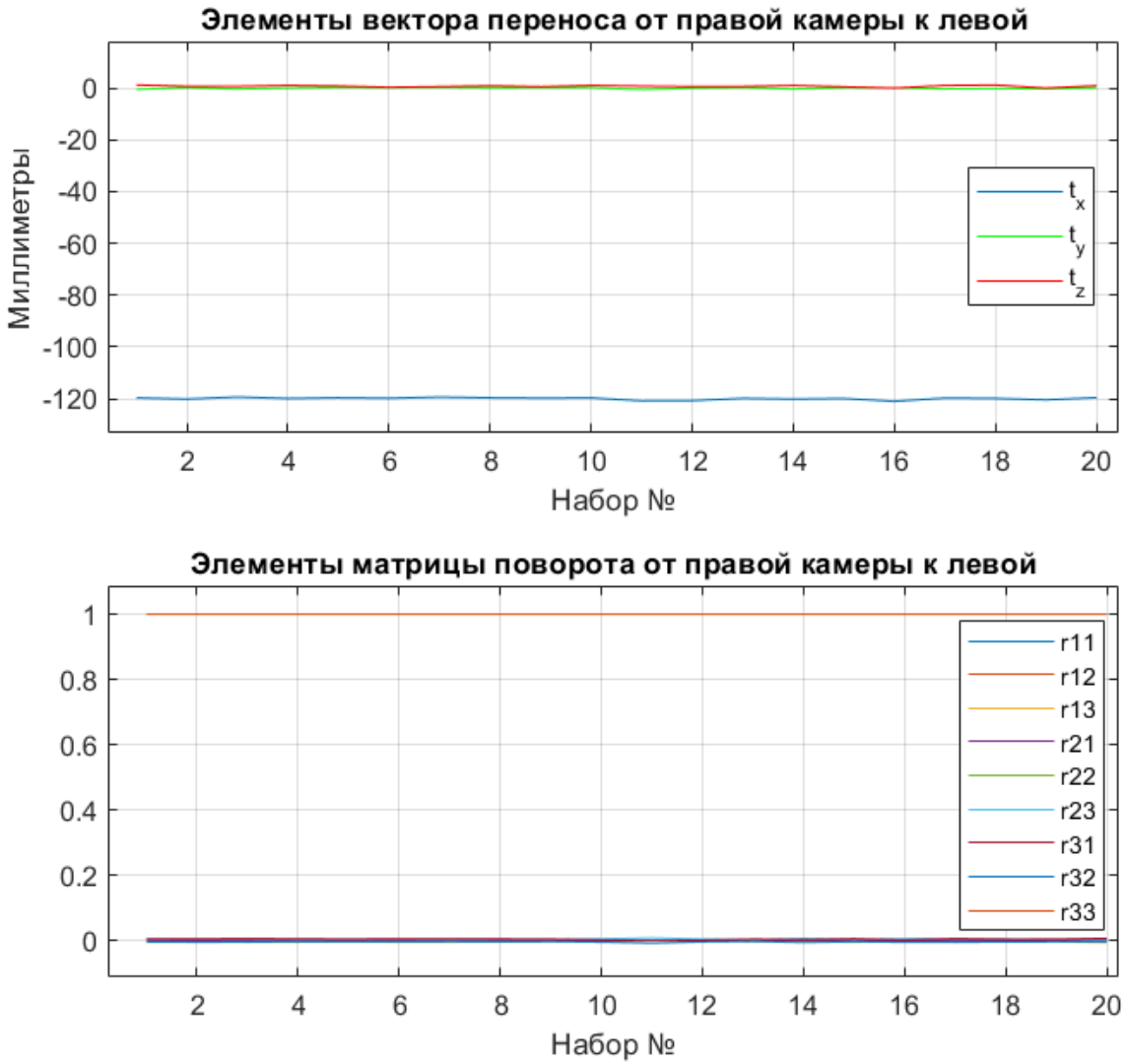

Рисунок 2.15: Вычисленные в Matlab внешние параметры стереокамеры

Подстановка полученных средних значений в *RTAB-Map* дала результаты гораздо хуже, чем полученные калибровкой в *RTAB-Map*. *RTAB-Map* не смог правильно ректифицировать изображения с такими параметрами. В связи с этим была произведена попытка исключения при подсчёте среднего значения точек на рисунках 3.11 – 3.15, которые похожи на выбросы. Подсчитанные таким образом параметры также показали те же результаты при ректификации в *RTAB-Map*, что и параметры, подсчитанные со всеми точками.

Также произведена попытка, вычисления параметров по одному набору изображений. На рисунке 3.16 представлен результат ректификации с этими параметрами в *Matlab*. Видно, что изображения очень хорошо ректифицируются с такими параметрами, средняя ошибка нахождения точек доски не превышает 0.08 пикселя, а расстояние между доской и стереокамерой определено как изменяющееся от 300 до 600 мм для снимков, что полностью верно: снимки для данного набора производились на таких расстояниях.

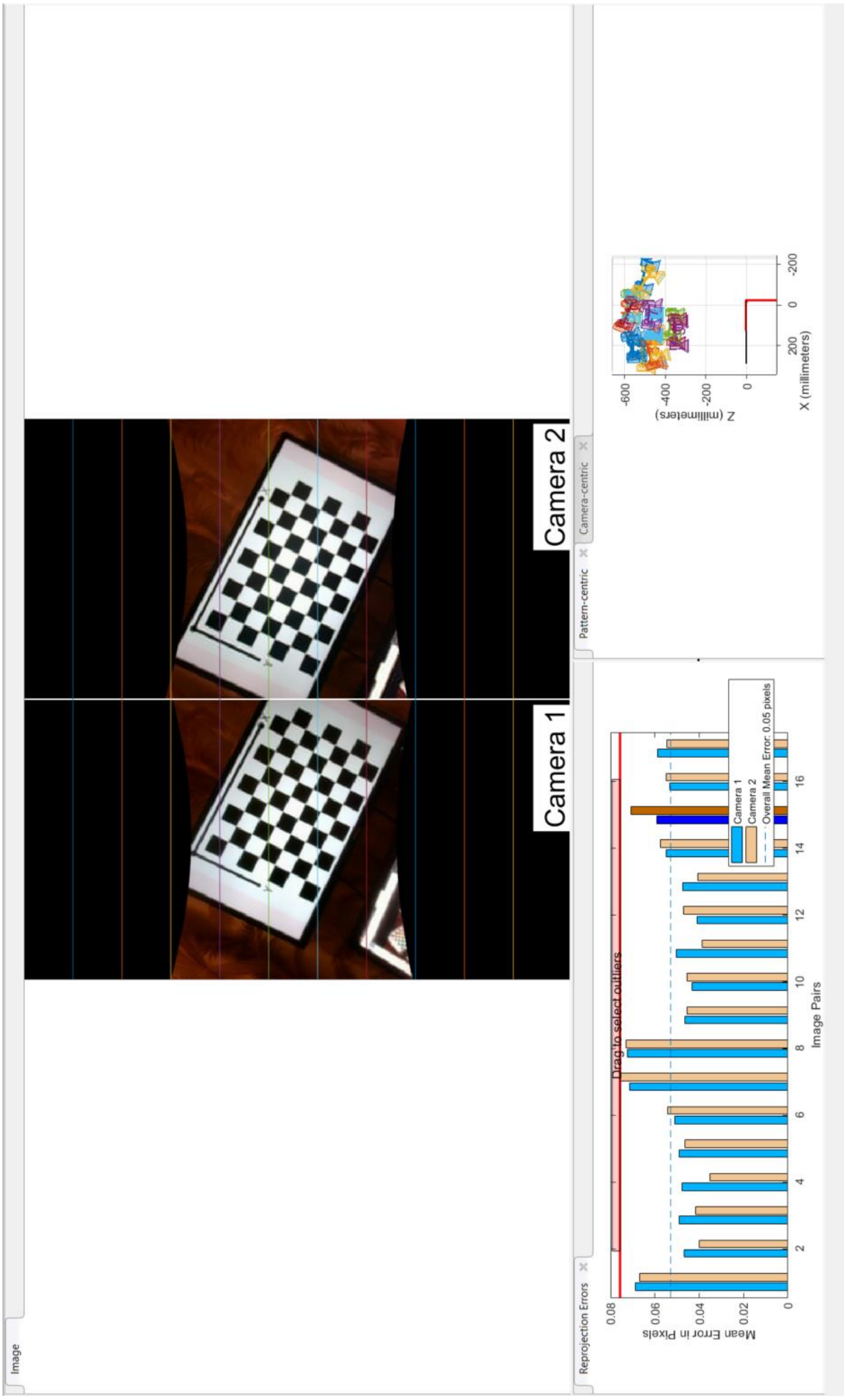

Рисунок 2.16: Результат ректификации изображений в Matlab

Подстановка же данных значений в *RTAB-Map* даёт те же неправильные результаты, что и подстановка средних значений параметров, и вместо ректифицированных изображений получаются искажённые изображения (рисунок 3.17). Скорее всего проблема вызвана разной размерностью некоторых из параметров. Возможно, *Matlab* и *RTAB-Map* по-разному работают с системами координат изображений (возможно, *Matlab* нормализует их [34], а *RTAB-Map* – нет, что приводит к отличной размерности параметров), что и вызывает данную проблему.

Рисунок 2.17: Результат ректификации в RTAB-Map с параметрами, вычисленными в Matlab

Для сравнения, подсчитанные средние значения и значения, полученные калибровкой в *RTAB-Map*, приведены в таблице.

Сопоставление результатов калибровки в Matlab и RTAB-Map

	Matlab		*RTAB-Map*	
параметр	Левая камера	Правая камера	Левая камера	Правая камера
f_x [пиксели]	274.9961	275.6397	2.7525298738490488e+02	2.7402275022942831e+02
f_y [пиксели]	274.9813	275.6784	2.7537908714896446e+02	2.7409274236031399e+02
c_x [пиксели]	155.9545	167.7224	1.5386561611599336e+02	1.6528810281534021e+02
c_y [пиксели]	253.0507	258.5686	2.5212370904158013e+02	2.5837949210776674e+02
k_1	-0.4290	-0.4302	-4.4402892163991875e-01	-4.2473155845928734e-01
k_2	0.2241	0.2207	2.8089036151036861e-01	2.1887432818801611e-01
k_3	-0.0646	-0.0572	-1.2892172011739422e-01	-7.0405515528998430e-02
p_1	-5.7340e-04	-6.3901e-04	-3.6202558206682364e-04	-2.7135482122054551e-03
p_2	4.4300e-04	-6.0828e-04	1.1238244945793842e-03	-9.1824897303893836e-04
t_x [мм]	-119.9586		-1.1731238556798615e+02	
t_y [мм]	-0.0451		2.6253246577993884e-01	
t_z [мм]	0.7014		-1.7129873290304989e-00	

Стоит отметить, что, производя вынужденную калибровку в *RTAB-Map*, не всегда удаётся получить хорошие параметры, часто после ректификации такими параметрами результат будет аналогичен результату на рисунке 3.17.

Как видно из таблицы параметры левой и правой камер различны. Также, полученные в *Matlab* значения отличаются от значений в *RTAB-Map*, но большинство параметров довольно близки друг к другу. Поэтому неудача с использованием значений *Matlab* в *RTAB-Map* может быть связана с особенностями размерностей используемых величин в этих программах.

По рисунке 3.15 видно, что матрица поворота от правой камеры ко второй практически не отлична от единичной на всех наборах изображений. Отстоит правая камера от левой на расстояние приблизительно равное 120 мм и на всех наборах данный параметр тоже был определён примерно одинаково: все значениях входят в погрешность 0.1 мм, с которой были измерены размеры квадратов доски. Таким образом относительная позиция камер друг относительно друга была подсчитана достаточно хорошо.

По рисунке 3.12 видно, что на большинстве наборов координаты главной точки отклоняются не более чем на 1 пиксель. Сравнивая отклонения параметров от средних значений на рисунках 3.11 – 3.15, можно заметить, что на одних и тех же наборах изображений некоторые параметры могут быть определены хорошо, в то время как другие иметь сильные отклонения. Так, например, на рисунках 3.12 – 3.14 на 12-ом наборе видны сильные отклонения от среднего значения, но на рисунке 3.11 для 12-ого набора фокусные расстояния вычислены довольно близкие к средним значениям. Можно сделать вывод, что для различных комбинаций снимков некоторые параметры вычисляются лучше, а другие хуже (например, параметры относительного расположения камер на всех наборах вычислены очень точно).

Тангенциальная дисторсия на рисунке 3.14 на всех наборах имеет маленькие значения, но довольно сильно колеблется относительно среднего. Согласно [34] для разрешения 320 на 480 при таком маленьком значении её влияние не значительно. Сильные колебания её значений могут быть связаны с тем, что в некоторых наборах оказалось недостаточно снимков с хорошим ракурсом для определения данного параметра.

На рисунке 3.13 можно видеть сильное отклонение на 19 наборе у коэффициентов радиальной дисторсии правой камеры. На 11 наборе для левой камеры можно заметить, что все 3-и параметра могут отклоняться от среднего в разной степени, хотя все являются коэффициентами радиальной дисторсии. Согласно [35] использование только k_1 и k_2 компенсирует 90% радиальной дисторсии.

В целом, то, что параметры различаются на разных наборах, соответствует рекомендации калибровать стереокамеру на расстоянии, на котором будет располагаться интересующий объект [29]. Кроме того, на всех наборах в *Matlab* удаётся получить хорошо ректифицированные изображения, поэтому использование средних значений параметров также даст удовлетворительный результат на всех наборах. Еще один вариант калибровки – объединить все снимки в один набор и откалибровать стереокамеру по всех снимкам одновременно.

Графики фокусного расстояния на рисунке 3.11 довольно сильно колеблются, в частности, это вызвано небольшим разрешением камеры, что приводит к определению точек доски недостаточно точно.

Хотя не удалось получить идеальные параметры для стереокамеры, но полученных результатов достаточно для того, чтобы произвести пробное сканирование области. А также, получено представление об поведении изменения

параметров при калибровке, что будет полезно для произведения в дальнейшем более точной калибровки стереокамеры.

После калибровки можно приступать к построению карты. Результат представлен на рисунке 3.18.

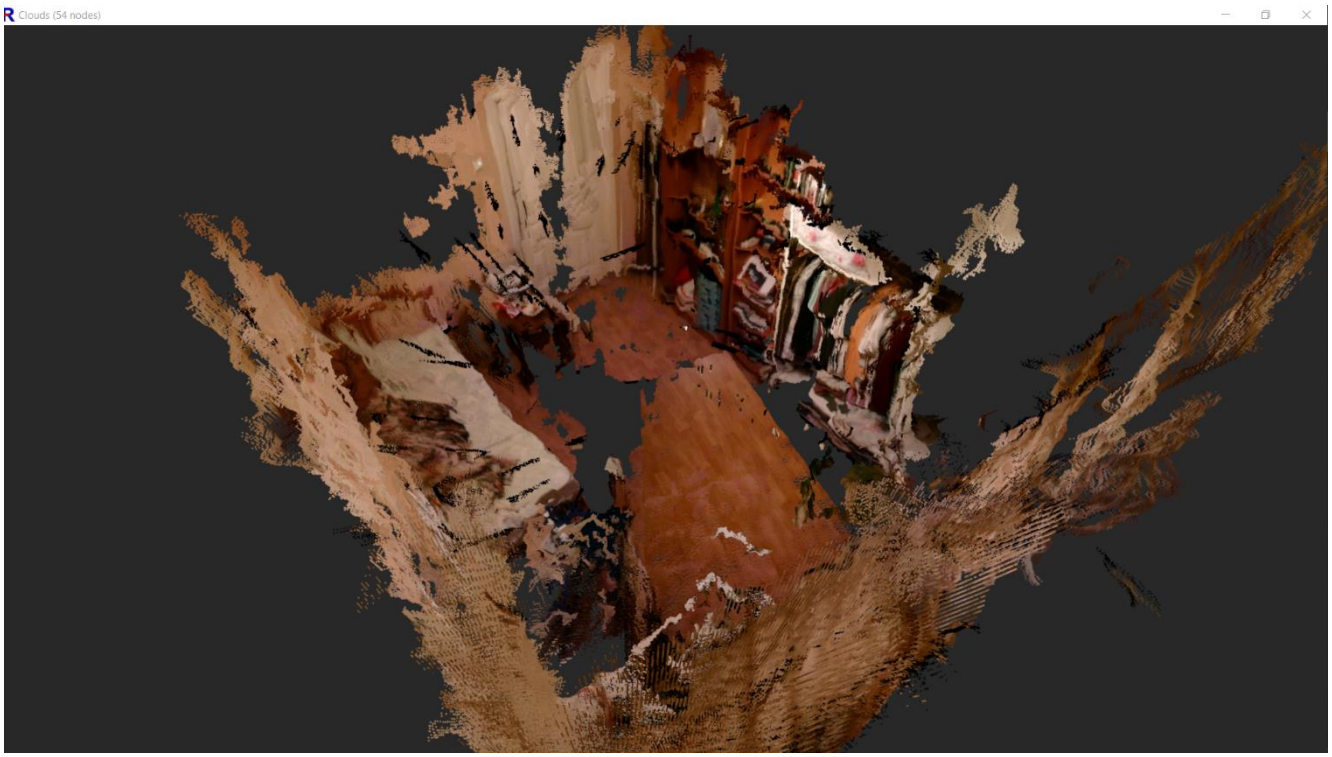

Рисунок 2.18: Снятая SLAM-методом RTAB-Map карта

Карта состоит из плотных облаков цветных точек расположенных в *3D* пространстве. На ней видны пустые места, некоторые из которых вызваны однородностью объектов, из-за которой стереосистема не может найти уникальные точки на объектах. Но таких участков немного. В большей степени проблема заключается в низком разрешении получаемых изображений, в частности, при низком разрешении плохо различимы некоторые текстуры. А также в качестве калибровки, что также влияет на корректное определение расстояниях до объектов.

На рисунке 3.19 представлена *Occupancy Grid* данной карты. *Occupancy grid* – представляет собой *2D* карту препятствий. Она строится сечением рисунка 3.18 выбранной плоскостью.

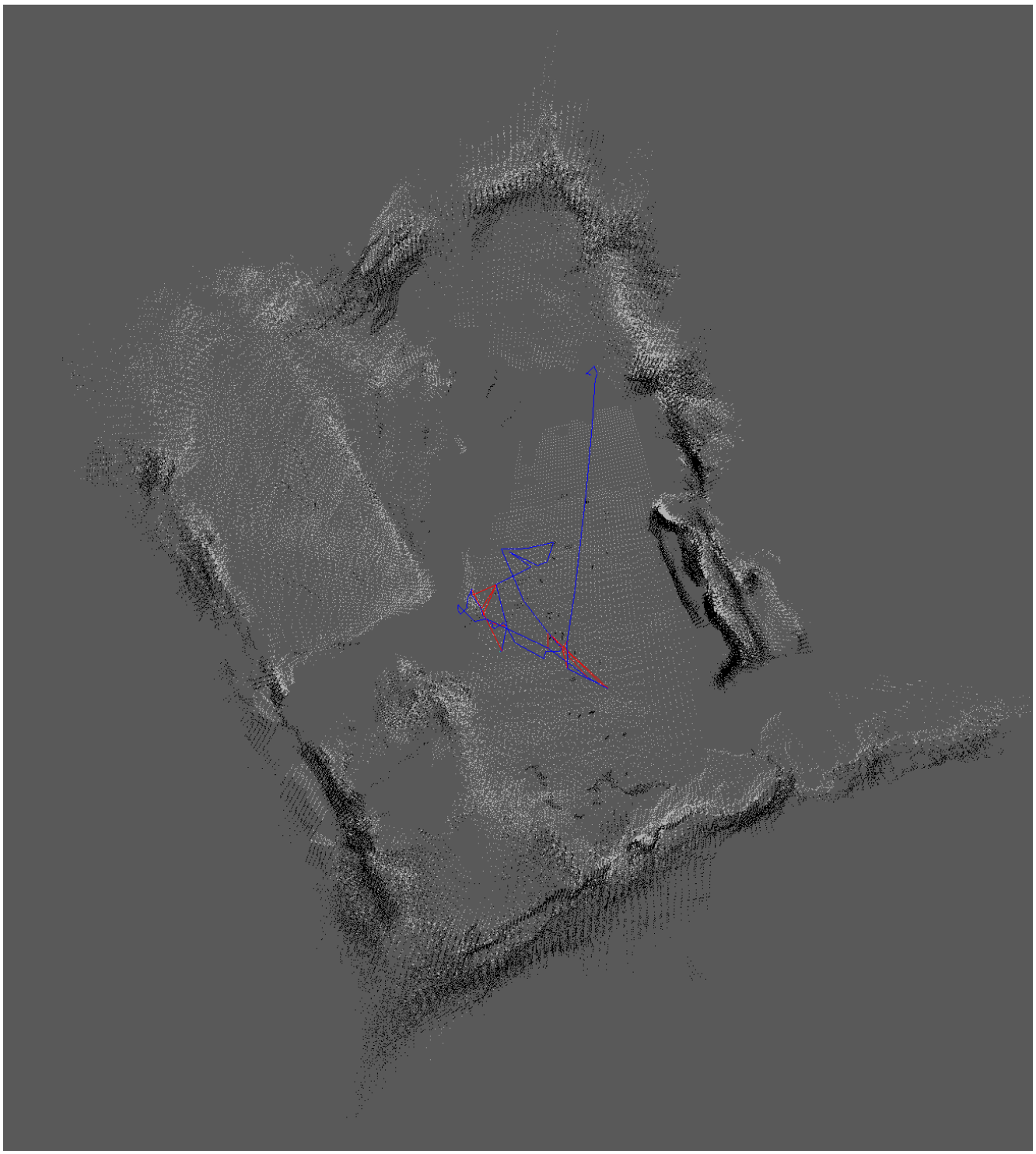

Рисунок 2.19: Occupancy grid снятой карты

На *Occupancy Grid* хорошо различима прямоугольная форма комнаты. *Occupancy grid* состоит из трёх цветов: чёрные области, белые и серые. Чёрные точки на ней – препятствия, которые помешаю роботу пройти. Белые точки – свободные для передвижения области. Например, белые точки в центре – точки плоскости, которая находится на некотором расстоянии от пола. Области определяются как свободные, если на определённом расстоянии от них нет никаких объектов, но определение свободности области произойдёт только в тот момент, когда робот посмотрит в направлении данной области. Так на рисунке 3.19 видны серые области в некоторых частях комнаты – в данные области робот не смотрел и поэтому свободны ли они или заняты неизвестно.

Также на *Occupancy Grid* хорошо видны шумы – сильно удалённые небольшие облака точек. От шумов можно избавиться, улучшив параметры стереосистемы, а также фильтруя данные. Например, установив максимальный радиус вокруг робота и, если расстояние до объекта превышает его, то такие данные отбрасываются. Также, можно установить минимальное расстояние между точками, определяющее считаются ли они одним облаком, и фильтровать данные по минимальному количеству точек в облаках. Также после снятия карты можно запустить её обработку, что увеличит количество найденных замыканий и сильно увеличит точность карты.

На рисунке 3.19 правая стена комнаты не доходит до левой – это происходит потому, что на данном месте расположено зеркало и стереосистема не видит разницы между отражением и реальным миром. Впрочем, человек иногда тоже может не распознать зеркало.

При снятии карты, *RTAB-Map* через определённые промежутки времени и по “решению” заложенного в него алгоритма сохраняет данные в виде узлов графа – на рисунке 3.19 находятся на концах красных и синих прямых линий (синие линии – прямые, идут в узлы и из узлов).

Создаваемые подряд узлы, соединяются между собой рёбрами – на том же рисунке являются синими и красными линиями. Рёбра описывают перемещение (перенос и поворот) системы координат робота в глобальной системе. Т.е. чтобы переместиться из узла (i) в узле (i+1) необходимо к системе координат робота узла (i) применить трансформацию, хранящуюся в ребре, соединяющем узлы (i) и (i+1). Рёбра бывают трёх типов:

1. Между соседями – соединяют два последовательно созданных узла (синие линии)
2. Созданные между узлами, выбранными на этапе поиска цикла замыканий (красные линии)
3. Созданные между близкими узлами

Каждый раз при добавлении ребра 2-ого или 3-ого типа вычисляется и распространяется через весь граф число, корректирующее накапливающуюся от одометрии ошибку. Данный процесс называется оптимизацией графа.

На карте узлы графа находятся ровно в том месте, в котором был робот при снятии данных. Таким образом на рисунке 3.19 видно его перемещения по комнате, какие позиции он занимал (он был на концах синих и красных прямых).

Благодаря процессу оптимизации графа, который происходил на позициях, в которых *RTAB-Map* “понимал”, что данный вид уже видел – позиции на концах красных линий, комната имеет прямоугольный вид. Без этого цикла поисков замыканий имелась бы накапливающаяся ошибка аналогичная дрейфу нуля гироскопа, что приводило бы к неправильному построению карты.

Следующим этапом является запуск всех программ в *ROS*, как если бы они запускались на роботе.

3. ЗАПУСК RTAB-MAP В ROS

В качестве программной среды робота будет использована симуляция в виртуальной машине *Virtual Box*, на которой установлена операционная система *Ubuntu* 16.04 с установленной *ROS*. Такая виртуализация негативно влияет на вычислительную мощность оборудования, но позволяет протестировать решение при отсутствии возможности использовать напрямую аппаратные компоненты.

Для подключения к камере используется *ROS* пакет *libuvc_camera* [36]. Для запуска пакета написан *launch*-файл (формат *XML*) *cameraOn.launch* [37], в котором установлены параметры частоты кадров, разрешения камеры и формат считываемых изображений. Используя данный пакет удалось получить изображения с камеры в гораздо белее высоком качестве вплоть до максимального заявленного производителем: 1280 на 960 пикселей для каждой камеры. Визуализация данных со стереокамеры при таком разрешении сильно нагружает виртуальную машину и действия на видео запаздывают.

Все *launch*-файлы расположены по адресу [38].

При запуске файла *cameraOn.launch* также запускается операционная система *ROS* и начинается процесс считывания данных с камеры и отправка их в соответствующие топики. Рисунок 4.1 демонстрирует что изображения с камеры отправляются в топик */stereocamera/image_raw*, откуда их может считать другая программа, что реализует удобное взаимодействие программ, созданных независимо. Представление взаимодействия программ в виде такого графа помогает легче понять, что происходит в системе. Направление потока данных отмечено стрелками. Для считывания данных программа должна подключиться к нужному топику. Топики могут хранить определённое максимальное количество данных одновременно, затем данные перезаписываются, программы, считывающие и записывающие данные в топики, ничего не знают друг о друге.

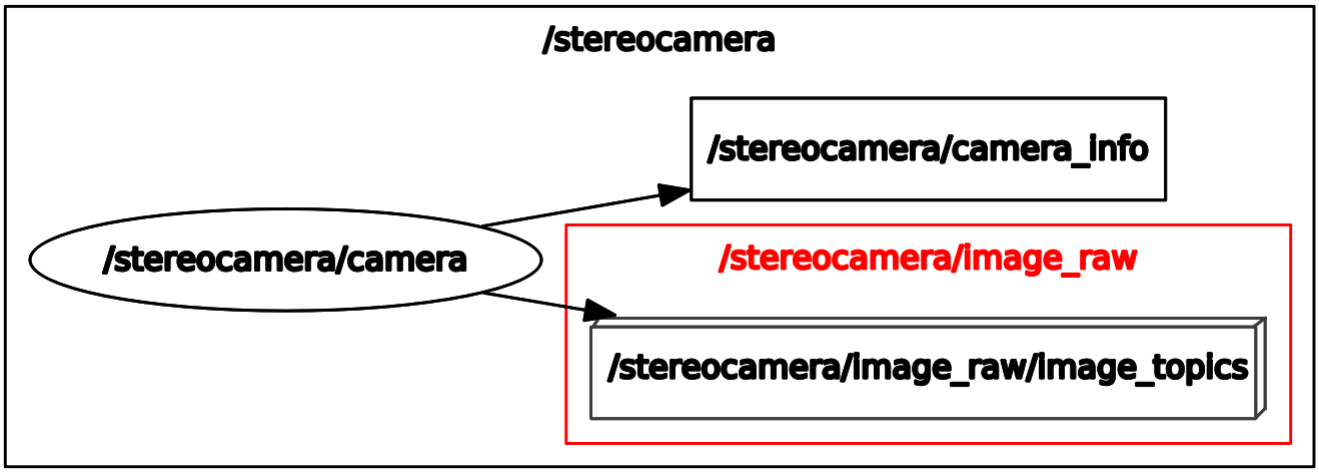

Рисунок 3.1: Структура взаимодействия программ. Чтение данных

Стереокамера успешно подключена к системе, теперь необходима её калибровка. Калибровка осуществлена с помощью *ROS* пакета *camera_calibration* [9]. Изображения считываются пакетом *libuvc_camera*, ректифицируются пакетом *image_proc* [39] и уже ректифицированные изображения будут передаваться *RTAB-Map* – такое решение позволит избежать проблем, возникших с калибровкой в прошлой главе и при этом использовать максимальные доступные разрешения изображений.

Пакеты *camera_calibration* и *image_proc* ожидают на вход два разделённых изображения: одно с левой камеры и одно с правой. Так как имеющаяся стереокамера передает эти изображения как единое целое, то необходимо их разделить. Для этого был написан *ROS* пакет *side_x_side_stereo* [40] (приложение 4) на языке *C++*. За его основу был взят код, выложенный на *GitHub* [41], он был модифицирован: теперь каждому изображению соответствует калибровочный файл, содержащий параметры камеры, с которой изображение было получено, а также отметка времени, когда изображение было сделано – данные изменения необходимы для корректной работы пакета *image_proc*, *RTAB-Map* и, в частности, визуальной одометрии (входит в *RTAB-Map*).

Калибровка с помощью пакета *camera_calibration* похожа на калибровку в *RTAB-Map*, например, результаты также сохраняются в файлы с расширением *yaml*. Но, в отличии от *RTAB-Map*, *camera_calibration* довольно быстро решает, что данных для калибровки достаточно, а её результаты более робастны. Результат калибровки представлен на рисунке 4.2. Качество калибровки получилось довольно хорошим.

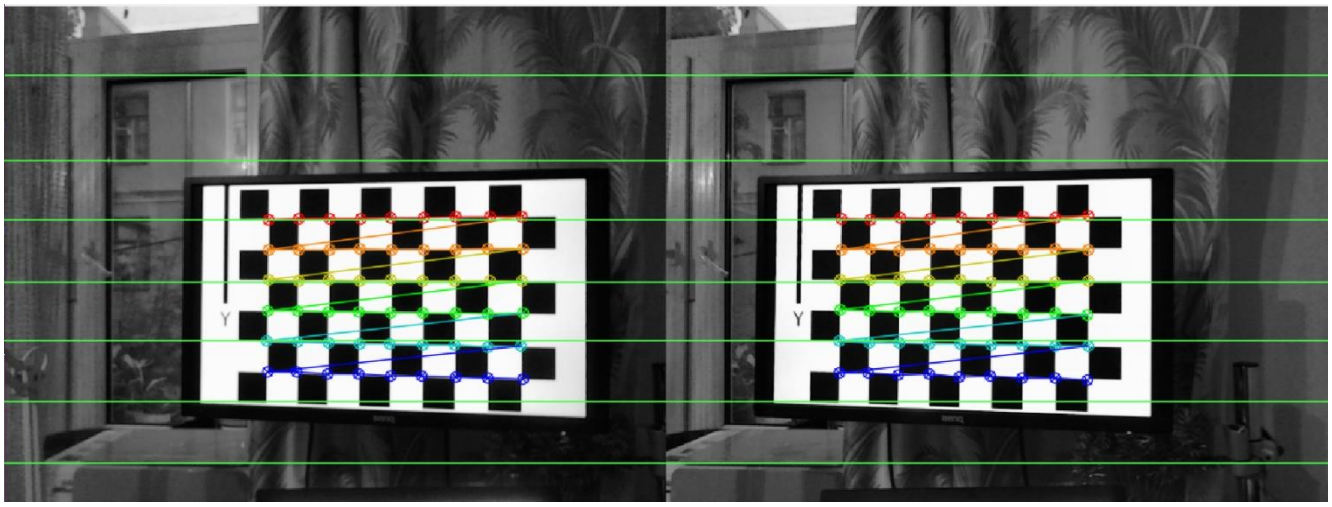

Рисунок 3.2: Результаты калибровки с помощью camera_calibration

Для удобно запуска написан файл *sxsOn.launch* [42], который запускает файл *cameraOn.launch* и затем запускает пакет *side_x_side_stereo*. Результат запуска файла *sxsOn.launch* представлен на рисунке 4.3.

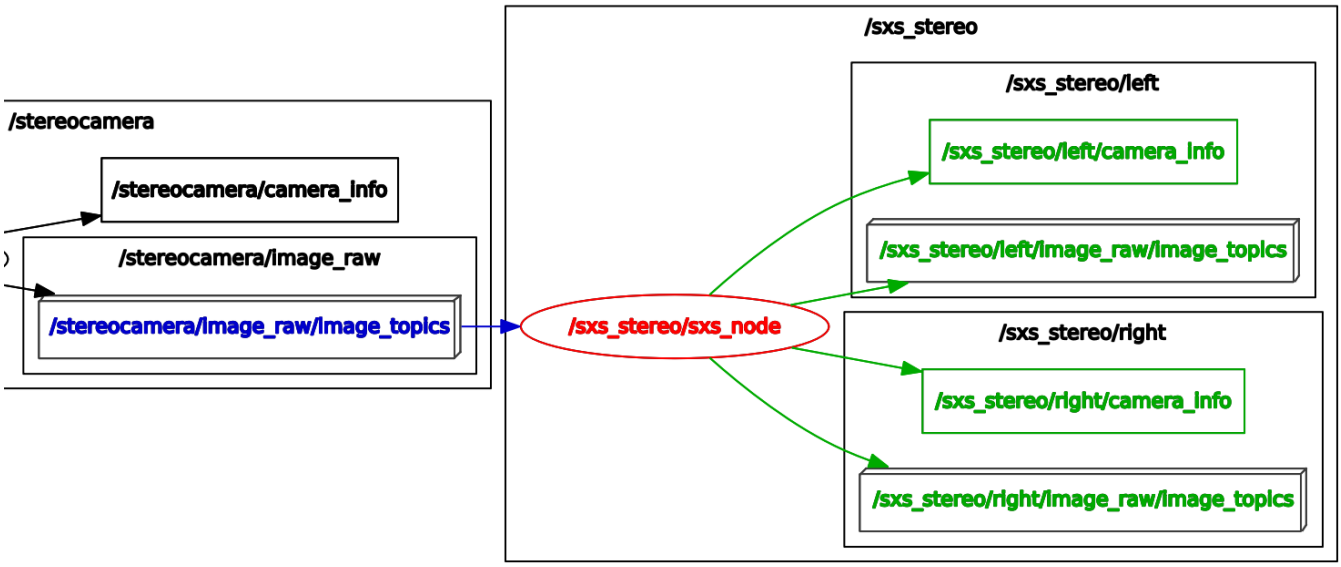

Рисунок 3.3: Структура взаимодействия программ. Разделение считываемых изображений

Из рисунка 4.3 видно, что считываемые с камеры изображения разделяются на изображения с левой камеры и изображения с правой, а также публикуется информация о камерах.

Данные изображения всё ещё не ректифицированы, для их ректификации нужно использовать пакет *camera_proc*, а также необходимо указать для робота системы координат: систему координат камеры, относительно центра робота. Данные действия произведены в файле *setup.launch* [43].

Для установки систем координат робота используется пакет *TF* [44], который устанавливает начальную позицию робота в глобальной системе координат и относительные положения его звеньев. Системы координат приведены на рисунке 4.4. Для испытаний стереокамера крепится к крышке ноутбука спроектированной и распечатанной деталью [45].

Рисунок 3.4: Система координат “робота”

В файле *setup.launch* описано расположение системы координат камеры в системе координат робота. Связи между системами координат *TF* представляет в виде графов. После запуска файла *setup.launch* граф *TF* будет иметь вид как на рисунке 4.5. Расположение робота в глобальной системе координат будет задано при запуске *RTAB-Map*.

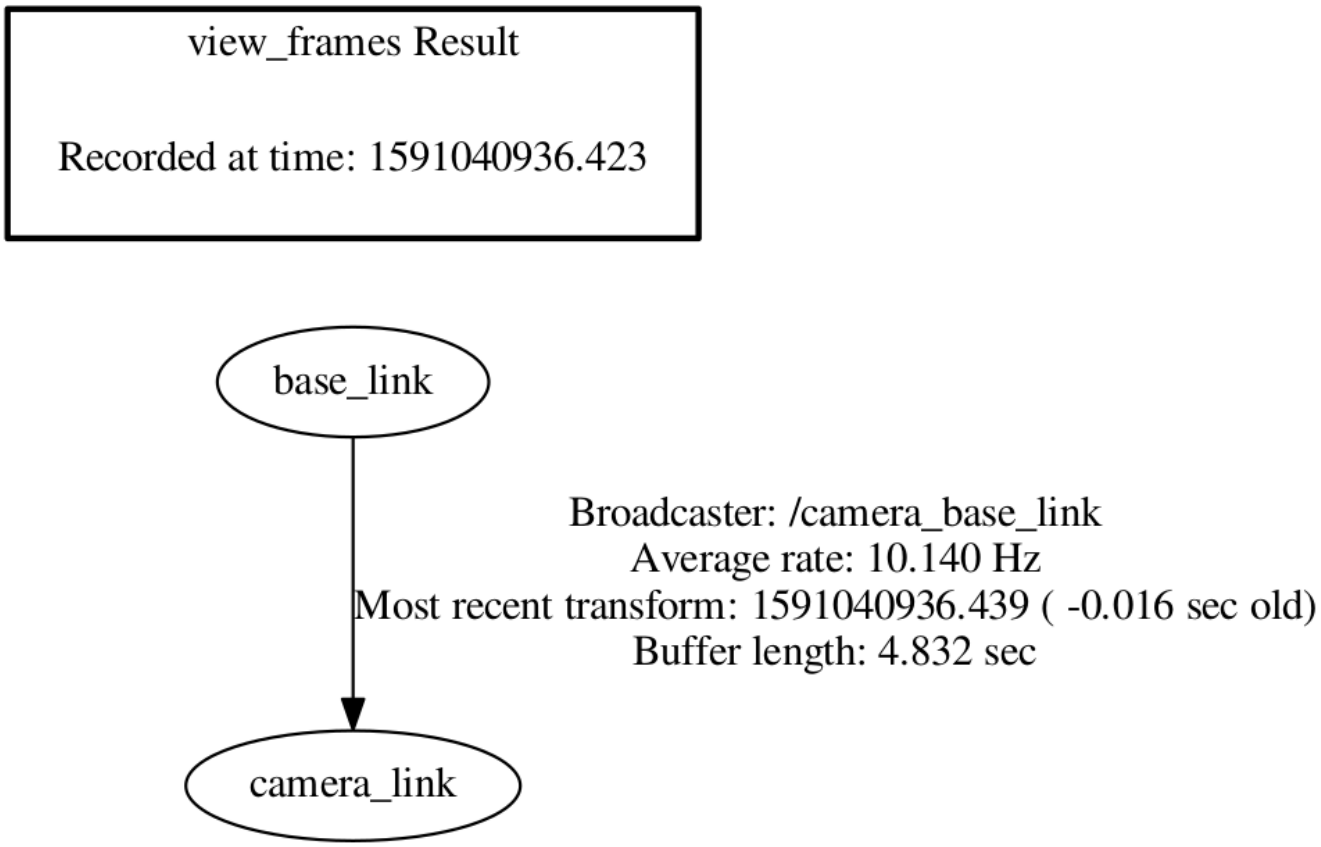

Рисунок 3.5: Представление отношений систем координат робота в ROS

Где *base_link* и *camera_link* соответственно система координат базы и стереокамеры.

Также будет произведена ректификация изображений. На рисунке 4.6 показано, что пакет *image_proc* читает изображения и данные камер, ректифицирует их и отправляет результаты в соответствующие топики. Результаты ректификации находятся в топиках */sxs_stereo/left/image_rect_color* и */sxs_stereo/right/image_rect*.

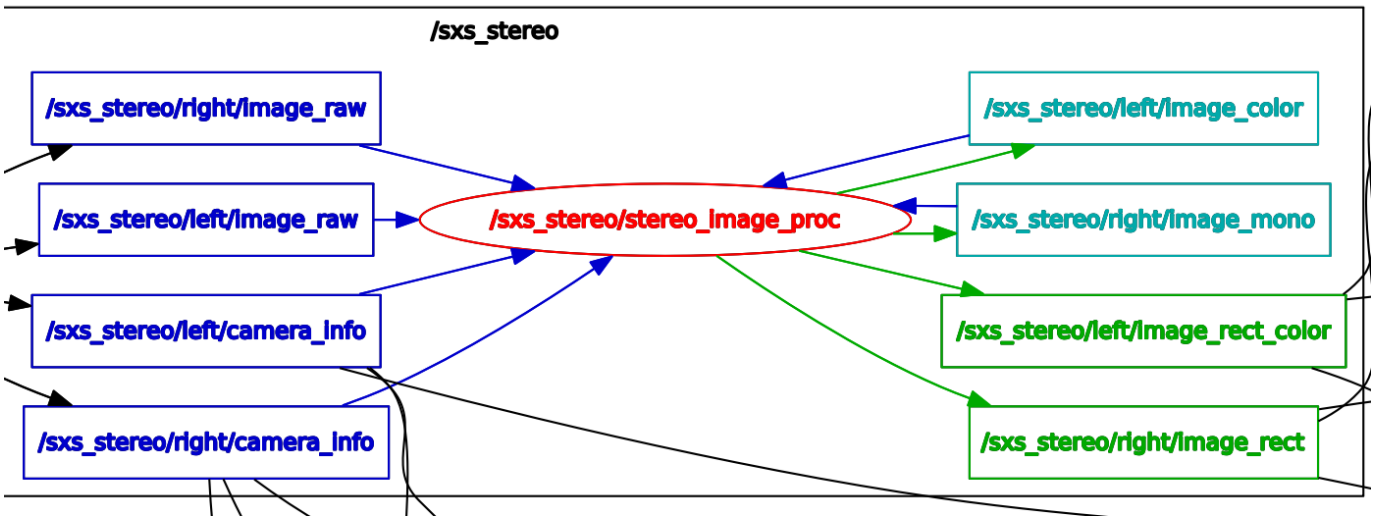

Рисунок 3.6: Структура взаимодействия программ. Ректификация изображений

Эти схемы получены пакетом *rqt_graph* [46], стоит отметить, что пакет не всегда отображает все связи между топиками и нодами, если к результату работы ноды никто не обращается. Дело в том, что некоторые пакеты специально устроены так, чтобы не отправлять сообщения в топики, если никто их не читает, а другие не читают сообщений из входных топиков, если результат их работы никому не требуется. Так, *image_proc* не подпишется на топики для чтения данных, пока кто-либо не подпишется на его выходные топики. Также на рисунке 4.6, в отличие от рисунка 4.3 и рисунка 4.1, отменена группировка топиков, что улучшает визуальное восприятие схемы.

После ректификации изображений и задания системы координат можно запускать пакет *RTAB-Map*. При запуске *RTAB-Map* нужно установить довольно много настроек, поэтому для него отдельно написан файл *rtabmap.launch* [47], а файл *main.launch* [48] запускает по очереди файлы *setup.launch* и *rtabmap.launch* после чего все программы становятся запущенными, а последний этап взаимодействия программ представлен на рисунке 4.7. Главной и конечной программой на нём является */rtabmap/rtabmap*, многие выходные топики которого скрыты для лучшей визуализации. С полным списком топиков можно ознакомиться на сайте библиотеки или введя в консоли команду *rostopic list*.

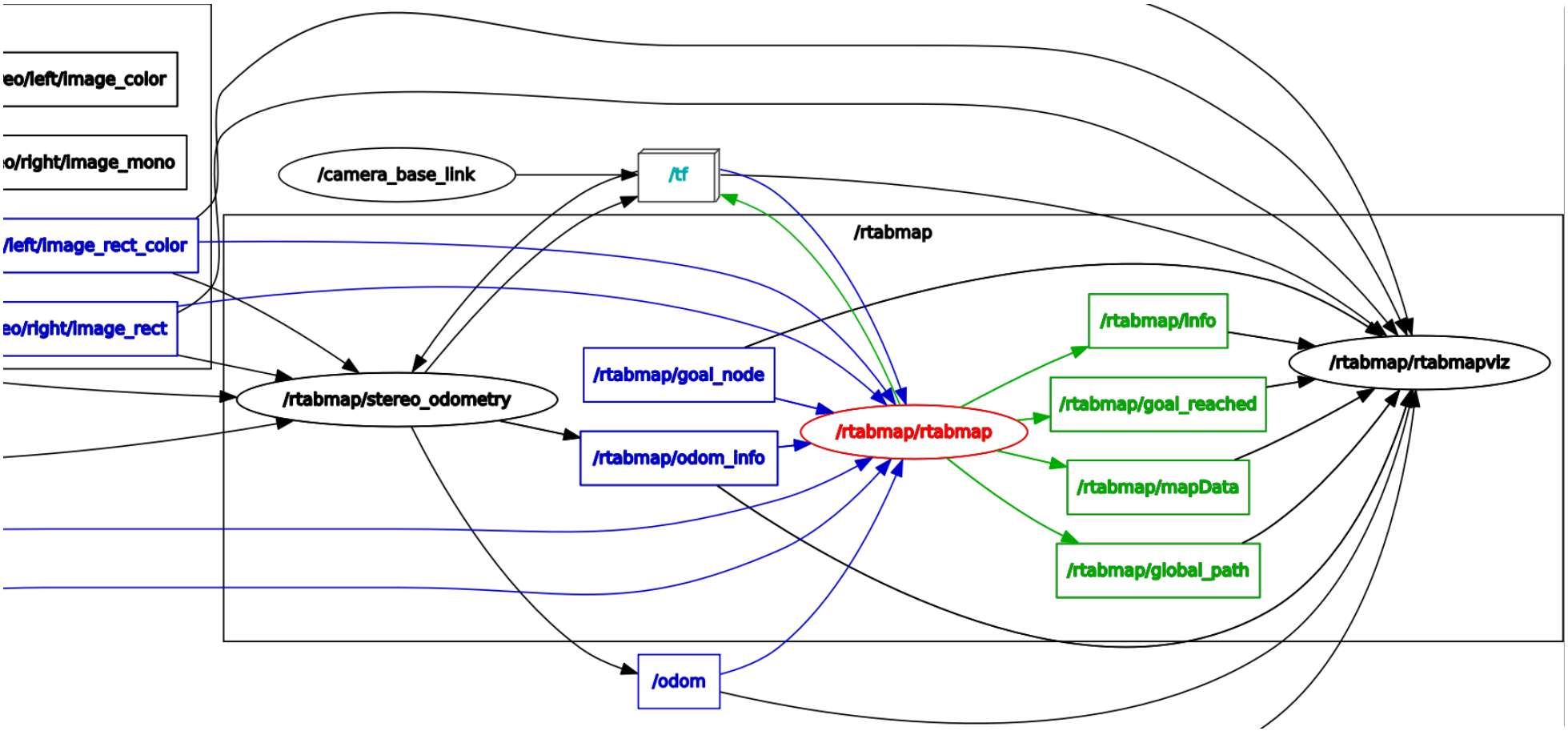

Рисунок 3.7: Структура взаимодействия программ. Визуальная одометрия и RTAB-Map

Программа */rtabmap/rtabmapviz* используется только для визуализации пользовательского интерфейса и на роботе её можно не запускать.

Из рисунка 4.7 видно, что для вычисления перемещений робота используется стереоодометря (*/rtabmap/stereo_odometry*), что из предыдущих схем *RTAB-Map* использует только ректифицированные изображения и параметры камеры, а также, что он активно использует модуль *TF*, первый раз инициализированный в файле *setup.launch*. Результирующее дерево отношений систем координат представлено на рисунке 4.8.

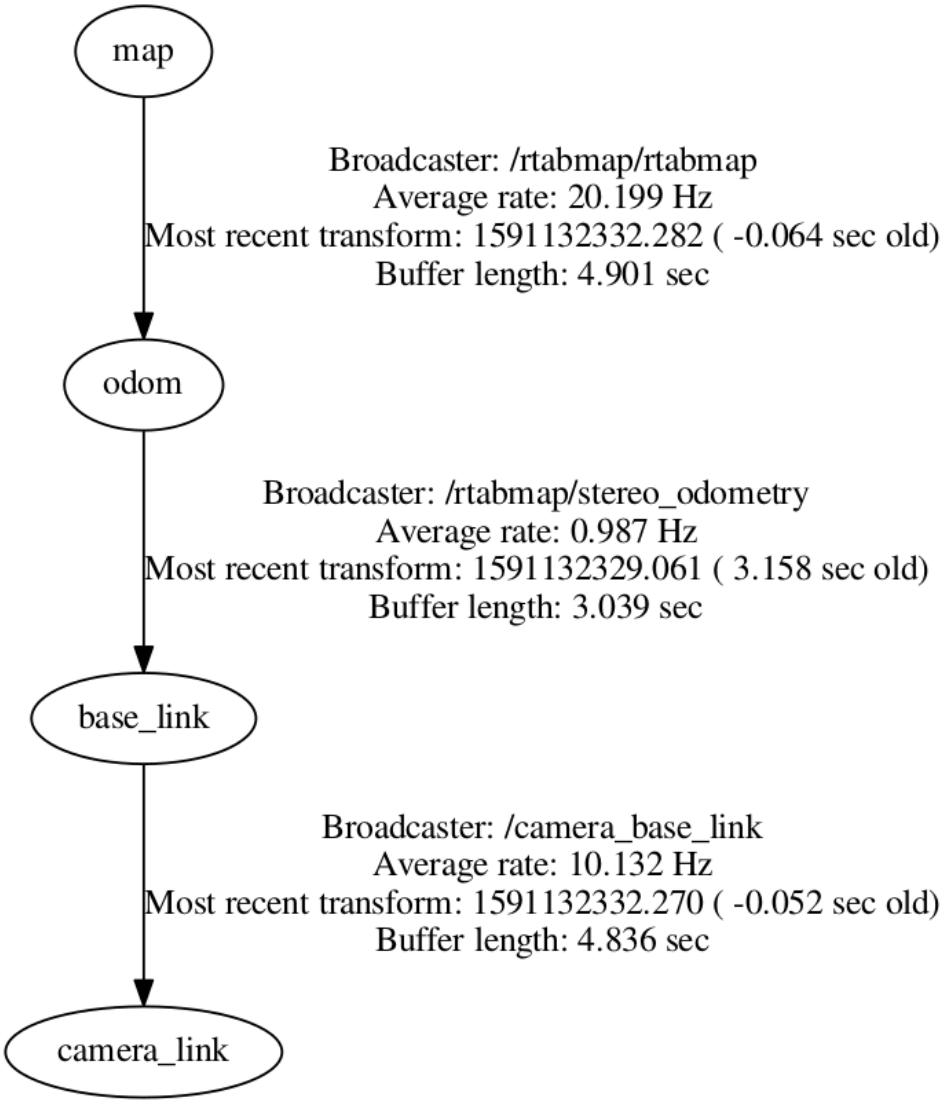

Рисунок 3.8: Отношения систем координат в ROS во время работы RTAB-Map

Видно, что появилось целых две новых систем координат: *map* и *odom*. Можно заметить, что трансформации между *map* и *odom* и *odom* и *base_link* последний раз были обновлены в разное и с разной частотой. Это связано с тем, что они из себя представляют и с тем, что производительность виртуальной машины очень низкая.

Когда робот включается и запускается *RTAB-Map,* то для вычисления его перемещений используется одометрия (в данном случае визуальная). Одометрию необходимо отсчитывать относительно какого-то начального положения, поэтому в момент включения робота создаётся система координат *odom* и за её начало берётся позиция робота при включении. В то же время необходимо знать положение робота в глобальной карте (уже имеющейся или всё ещё сканируемой), поэтому создаётся система координат map, которая представляет собой глобальную систему координат и находится в одном и тоже месте на карте.

При включении робота система координат map совпадает с *odom*, но, если уже имеется карта пространства, то, как только робот с помощью цикла замыканий обнаружит местность, которую уже “знает”, он перенесёт начало системы координат map на место, выбранное за начало глобальной системы координат на имеющейся карте. Таким образом станет точно известна позиция робота на глобальной карте, а также позиция робота относительно места его инициализации, так как *odom* изменена не будет (будут изменены вектор переноса и матрица поворота между map и *odom* из-за переноса *map*). Такое решение устраняет неоднозначности при работе робота на неполностью известной местности.

На этом этапе подготовка оборудования и программного обеспечения для поставленной задачи завершена и можно произвести тест решения. Так как вычислительных ресурсов виртуальной машины очень мало, то тест будет произведён демонстративный и на небольшом пространстве. Нужно отметить, что

в тесте рассматривается вариант, когда робот не “знает” местность, то есть её сканирование не было изначально произведено. Данный вариант является наименее точным с точки зрения позиционирования, но не требует больших вычислительных ресурсов, что позволяет его провести с использованием виртуальной машины. На рисунке 4.9 представлена схема перемещений робота: стартуя в позиции I, перемещается в позицию II и заканчивает своё движение в позиции III. Также на рисунке 4.9 отмечена глобальная система координат, относительно которой будет наблюдаться изменение положения робота.

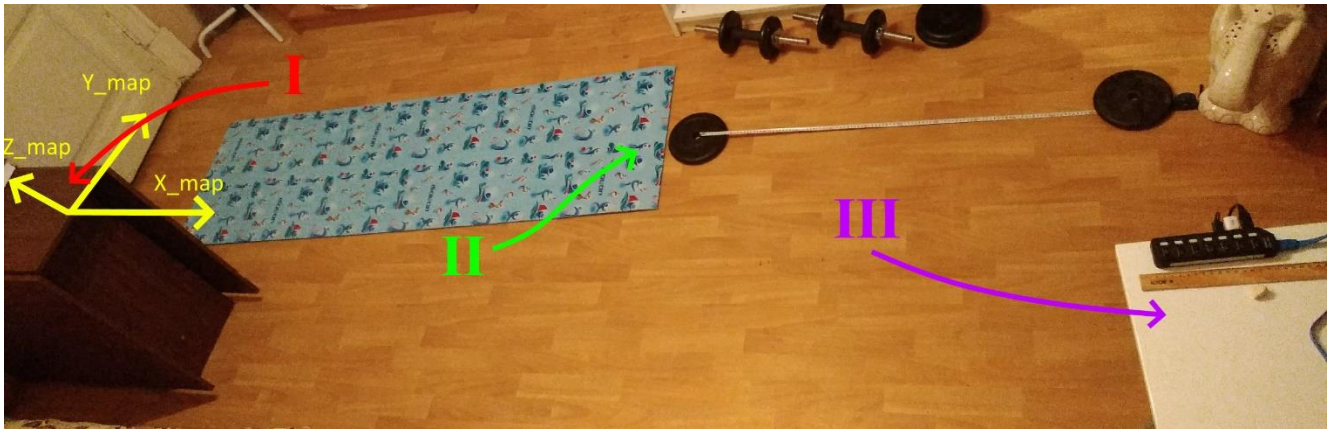

Рисунок 3.9: Испытательный полигон

При запуске робота в позиции I (рисунок 4.10) видно, что он стоит в начале глобальной системы координат и не повёрнут (небольшие значения вызваны смешением крышки ноутбука).

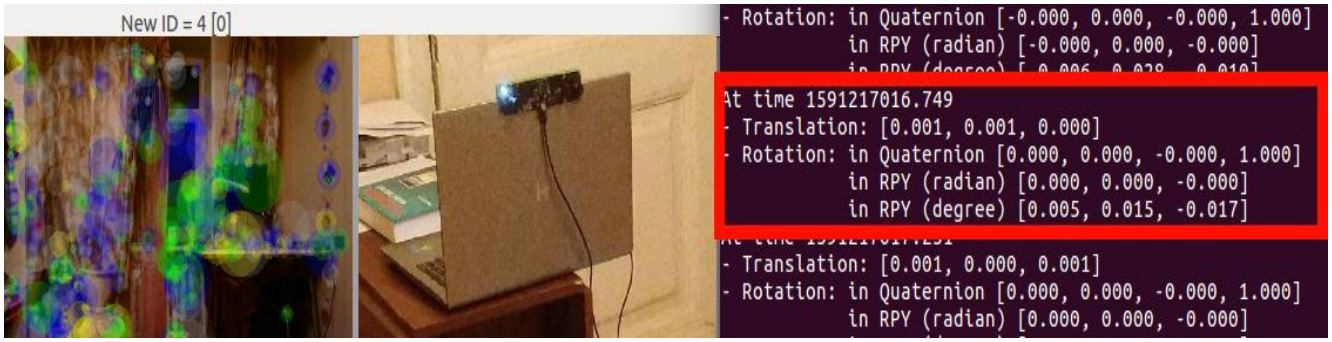

Рисунок 3.10: Робот на I позиции

На рисунке 4.10 слева на право изображено: что видит робот с обнаружением признаков детектором *SURF*, положение робота, вывод в консоль положения робота (вектора переноса [м] (*Translation*) и углов поворота в градусах (*RPY*: крен, тангаж и рыскание) между map и *base_link* (рисунок 4.8)).

Результат передвижения на позицию II представлен на рисунке 4.11.

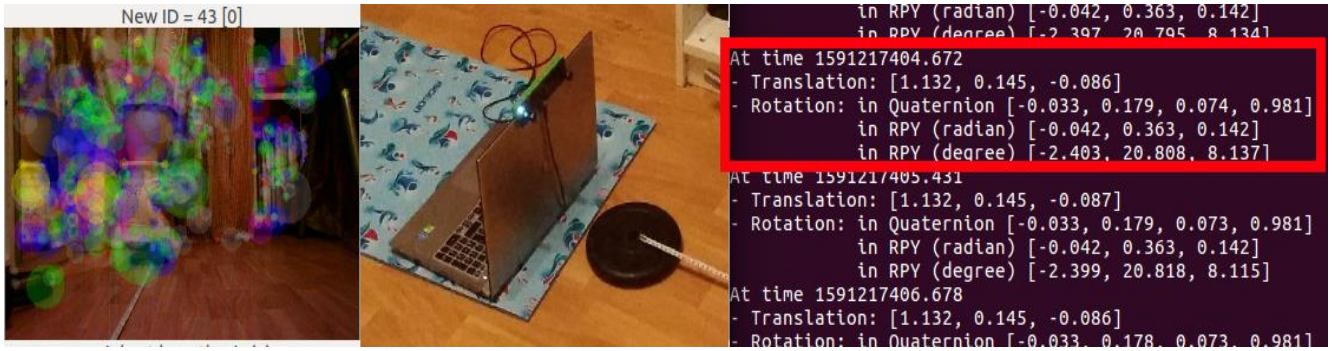

Рисунок 3.11: Робот на II позиции

Как видно по подсчётам программы, робот считает, что переместился на 1.132 м вдоль оси *X_map*, на 0.145 м вдоль *Y_map* и опустился на 0.086 м по оси *Z_map*. По всем осям перемещение было определено в верном направлении, но оказалось меньше действительного – это вызвано тем, что *RTAB-Map* для увеличения точности пытается предугадать куда переместится робот, но виртуальная машина не позволяет произвести считывание и обработку изображений достаточно быстро: считывание изображений происходит с запаздыванием, а частота обработки составляет порядка 0.5 – 1 Гц. Таким образом точность перемещений сильно падает. При работе без виртуальной машины (работа в прошлой главе на *Windows*) такой проблемы не наблюдается. Кроме того, неточности определения положения по одометрии компенсируются при обнаружении замыканий.

На рисунке 4.12 представлено последнее положение робота. Снова по всем осям было правильно определено направление движения, но сильно занижены их значения.

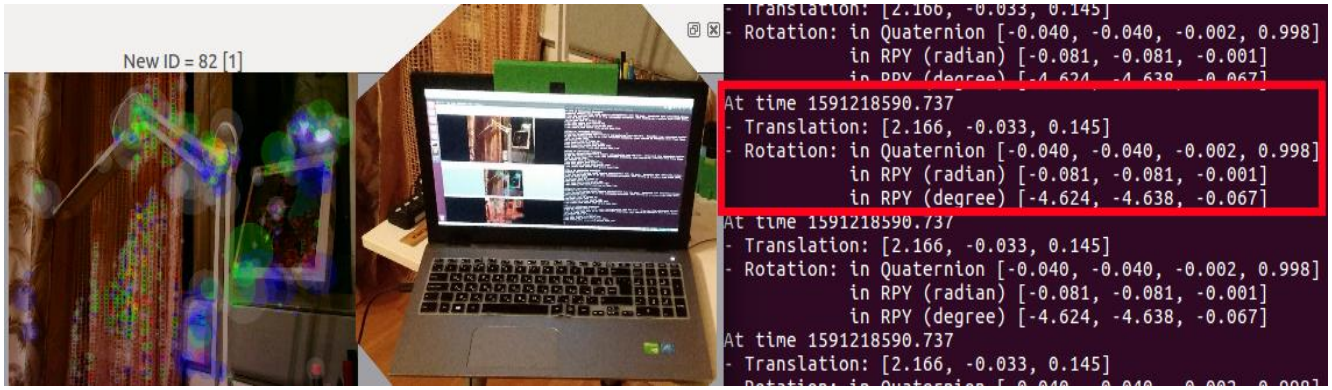

Рисунок 3.12: Робот на III позиции

Результаты эксперимента показали: робот способен определить своё положение в пространстве, что говорит о том, что представленная работа решает поставленную задачу. Все требования поставленной задачи были соблюдены. Дальнейшей работой является встраивание реализованного программного комплекса в аппаратную часть робота, что не вызывает проблем, так как в решении используется операционная система *ROS*, позволяющая создавать универсальные решения, работающие одинаково на любой аппаратной платформе, а используемые алгоритмы вычислительно затратными не являются: проблемы были вызваны сильно низкой эффективностью работы виртуальной среды, а не сложностью алгоритмов.

ЗАКЛЮЧЕНИЕ

В работе продемонстрировано применение операционной системы *ROS* в робототехнической задаче, показано, что на текущий момент делаются попытки создания универсальных средств разработки, таких как *ROS*, для удобного использования робототехниками в их задачах, но пока что эти средства имеют значительные недостатки: высокий порог вхождения, зависимость от основной операционной системы и, иногда, необходимость глубокого понимания особенностей программной среды, например, для компиляции. Также, часто можно встретить низкую задокументированность.

Существует некоторое разнообразие готовых решений, которые можно применить для робототехнических задач, но применение большинства таких решений ограничивается простыми задачами, а достаточно мощные решения могут быть платформозависимыми, поэтому часто приходится решать задачу с нуля.

В визуальных задачах *SLAM* качество источника визуальной информации является не менее важным, чем выбранный алгоритм решения. Так, желательными являются разрешение сенсора не менее 1280 на 720 пикселей и частота работы от 30 Гц. При выборе стереосистем необходимо отталкиваться от расстояний, на которых планируется их работа. Для небольших расстояний нужно использовать стереокамеры с меньшим расстоянием между объективами камер, а для больших – с большими. Также, для работы на больших расстояниях необходима большая разрешающая способность стереокамеры. Так, на выбранной камере с расстоянием между объективами 120 мм удаётся распознать объекты на расстояниях от 0.6-1 м. Можно заметить высокую стоимость на стереокамеры даже самого низкого класса.

В плане калибровки наиболее удобным оказался способ, реализованный в *ROS* пакете *camera_calibration*. Со встроенной в *RTAB-Map* системой калибровки

не удалось получить достаточно хорошие параметры стереокамеры. Оказалось, что *RTAB-Map* и *Matlab* некоторые параметры камер вычисляют в разных размерностях. Для успешной калибровки необходимо соблюдать баланс между размером калибровочной доски и расстоянием между камерами. Так, с увеличением расстояния калибровки, необходимо увеличивать размер доски. Вычисляемые калибровкой параметры являются оптимальными для набора изображений, участвующих в калибровке, но к их точности в условия применения стереокамеры нужно относиться с осторожностью. Было замечено, что при использовании минимально возможного разрешение стереокамеры её дисторсия более заметна, чем при использовании её максимального разрешения.

Одним из лучших готовых решений для задач *SLAM* является *RTAB-Map*, который проще всего использовать под *ROS*, но можно также скомпилировать на *Windows*. *RTAB-Map* не требователен к вычислительным мощностям и распространяется вместе с удобными инструментами, позволяющими исследовать качество решения. *RTAB-Map* оказался подходящим решением для поставленной задачи и достаточно гибким, поэтому можно рекомендовать использовать его в других *SLAM*-задачах. Точность позиционирования методом *RTAB-Map* напрямую зависит от: качества калибровки стереосистемы, плавности получения кадров, разрешающей способности стереосистемы, частоты работы *RTAB-Map* (устанавливается пользователем, не может превышать частоту получения кадров и должна соответствовать имеющимся вычислительным мощностям). Согласно проведённым экспериментам, при использовании *RTAB-Map* без виртуализации можно ожидать точности позиционирования выше 0.1 м в известной местности. В неизвестной местности будет накапливаться ошибка, но она будет устранена при обнаружении замыканий. Также её можно компенсировать внешними системами, такими, как *GPS*. После снятия *3D* карты местности можно рекомендовать произвести постобработку снятых данных, что сильно увеличит качество

полученных результатов благодаря нахождению пропущенных замыканий и возможности дополнительно отфильтровать данные.

СПИСОК ЛИТЕРАТУРЫ

[1] McKinsey&Company, «The future of parcel delivery: Drones and disruption,» 2019. [В Интернете]. Available: https://www.mckinsey.com/featured-insights/the-next-normal/parcel-delivery.

[2] B. Heid, «Technology delivered: Implications for cost, customers, and competition in the last-mile ecosystem,» McKinsey&Company, 2018. [В Интернете]. Available: https://www.mckinsey.com/industries/travel-transport-and-logistics/our-insights/technology-delivered-implications-for-cost-customers-and-competition-in-the-last-mile-ecosystem.

[3] MarketsandMarkets, «Delivery Robots Market by Load Carrying Capacity (Up to 10, 10.01–50.00, and More than 50 kg), Component (LiDAR Sensors, Control Systems), Number of Wheels (3, 4, and 6), End-User Industry (Food & Beverages, Retail), and Geography - Global Forecast 2024,» 2019. [В Интернете]. Available: https://www.marketsandmarkets.com/Market-Reports/delivery-robot-market-263997316.html.

[4] «Amazon Prime Air,» [В Интернете]. Available: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011. [Дата обращения: 2020].

[5] S. Scott, «Meet Scout,» 23 Январь 2019. [В Интернете]. Available: https://blog.aboutamazon.com/transportation/meet-scout.

[6] J. Wilke, «A drone program taking flight,» 5 Июнь 2019. [В Интернете]. Available: https://blog.aboutamazon.com/transportation/a-drone-program-taking-flight.

[7] «RTAB-Map,» 2020. [В Интернете]. Available: http://introlab.github.io/rtabmap/.

[8] MathWorks, «Matlab,» [В Интернете]. Available: https://www.mathworks.com/products/matlab.html. [Дата обращения: 2020].

[9] J. Bowman и P. Mihelich, «camera_calibration,» [В Интернете]. Available: http://wiki.ros.org/camera_calibration. [Дата обращения: 2020].

[10] А. В. Уразбахтин, «Алгоритм для системы позиционирования робота с одновременной навигацией и составлением карты SLAM,» *Энергетические и электротехнические системы,* pp. 277-283, 2016.

[11] M. Labbe и F. Michaud, «2014 IEEE/RSJ International Conference on Intelligent Robots and Systems,» в *Online global loop closure detection for large-scale multi-session graph-based SLAM*, Chicago, 2014.

[12] М. И. Собченко и В. И. Ухандеев, «Алгоритмы SLAM: обзор существующих решений,» *Электронные информационные системы,* т. 1(1), pp. 69-78, 2014.

[13] И. С. Балташов и Д. А. Демидов, «Неделя науки СПБПУ,» в *Сравнительное исследвание реализованных в ROS алгортимов SLAM*, Санкт-Петербург, 2016.

[14] M. Labbe и F. Michaud, «RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation,» *Journal of Field Robotics,* т. 36, № 2, p. 416–446, 2019.

[15] R. Dube, «SegMatch,» [В Интернете]. Available: https://github.com/ethz-asl/segmap. [Дата обращения: 2020].

[16] A. Harmat и M. Tribou, «MCPTAM,» [В Интернете]. Available: https://github.com/aharmat/mcptam. [Дата обращения: 2020].

[17] F. Endres, «RGBDSLAMv2,» [В Интернете]. Available:

https://wiki.ros.org/rgbdslam. [Дата обращения: 2020].

[18] А. В. Григорьевич, «Обзор технологии навигации мобильных роботов OMNIDIRECTIONAL VSLAM,» *Сборник научных трудов Новосибирского Государственного Технического Университета,* т. 2(92), pp. 81-92, 2018.

[19] M. Labbe и F. Michaud, «Long-term online multi-session graph-based SPLAM with memory management,» *Autonomous Robots,* т. 42, № 6, pp. 1133-1150, 2018.

[20] Open Robotics, «ROS,» [В Интернете]. Available: https://www.ros.org/. [Дата обращения: 2020].

[21] У. В. Михайлова, Е. А. Михайлов и А. С. Сарваров, «Использование фреймворка ROS для разработки архитектуры системы управления роботом,» *Электротехнические системы и комплексы,* № 21, pp. 117-121, 2013.

[22] Д. В. Копытов и К. В. Климов, «Вторая молодёжная конференция "Инновационная деятельность в науке и технике. Электромеханика, автоматика и робототехника",» в *Использование Robot Operating System (ROS) при создании робототехнических систем*, Истра, 2018.

[23] Kitware, «CMake,» [В Интернете]. Available: https://cmake.org/. [Дата обращения: 2020].

[24] 3Dvideo, «Камеры глубины — тихая революция (когда роботы будут видеть) Часть 2,» Июль 2019. [В Интернете]. Available: https://habr.com/ru/post/458458/. [Дата обращения: 2020].

[25] А. Г. Чибуничев, С. Б. Макаров и Е. В. Полякова, «Исследование возможности применения недорогих стереокамер для навигации роботизированных систем,» *Известия высших учебных заведений, геодезия и аэрофотосъёмка,* т. 63,

№ 6, pp. 645-649, 2019.

[26] Nvidia, «Встраиваемые системы для автономных машин нового поколения,» [В Интернете]. Available: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/. [Дата обращения: 2020].

[27] Oracle, «VirtualBox,» [В Интернете]. Available: https://www.virtualbox.org. [Дата обращения: 2020].

[28] Canonical, «Ubuntu,» [В Интернете]. Available: https://ubuntu.com/. [Дата обращения: 2020].

[29] The MathWorks, «Stereo Camera Calibrator App,» 2020. [В Интернете]. Available: https://www.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html.

[30] Е. П. Кружков, «StereoCamera.py,» 2020. [В Интернете]. Available: https://github.com/Quetalas/Stereo-Camera-Matlab/blob/master/StereoCamera.py.

[31] OpenCV team, «OpenCV,» [В Интернете]. Available: https://opencv.org/. [Дата обращения: 2020].

[32] Е. П. Кружков, «averageMinMax.m,» 2020. [В Интернете]. Available: https://github.com/Quetalas/Stereo-Camera-Matlab/blob/master/averageMinMax.m.

[33] Е. П. Кружков, «calibraate.m,» 2020. [В Интернете]. Available: https://github.com/Quetalas/Stereo-Camera-Matlab/blob/master/calibrate.m.

[34] The MathWorks, «estimateCameraParameters,» 2020. [В Интернете]. Available: https://www.mathworks.com/help/vision/ref/estimatecameraparameters.html.

[35] Р. Клетте, Компьютерное зрение. Теория и алгоритмы, Издательство "ДМК",

2019.

[36] K. Tossell, «libuvc_camera,» [В Интернете]. Available: http://wiki.ros.org/libuvc_camera. [Дата обращения: 2020].

[37] Е. П. Кружков, «cameraOn.launch,» 2020. [В Интернете]. Available: https://github.com/Quetalas/SLAM-launches/blob/master/cameraOn.launch.

[38] Е. П. Кружков, «SLAM-launches,» 2020. [В Интернете]. Available: https://github.com/Quetalas/SLAM-launches.

[39] V. Rabaud, «image_proc,» 2020. [В Интернете]. Available: http://wiki.ros.org/image_proc.

[40] Е. П. Кружков, «side_x_side_stereo,» 2020. [В Интернете]. Available: https://github.com/Quetalas/side_x_side_stereo.

[41] D. B. Curtis, «side_x_side_stereo,» 2019. [В Интернете]. Available: https://github.com/dbc/side_x_side_stereo/blob/master/src/side_by_side_stereo_node.cpp.

[42] Е. П. Кружков, «sxsOn.launch,» 2020. [В Интернете]. Available: https://github.com/Quetalas/SLAM-launches/blob/master/sxsOn.launch.

[43] Е. П. Кружков, «setup.launch,» 2020. [В Интернете]. Available: https://github.com/Quetalas/SLAM-launches/blob/master/setup.launch.

[44] T. Foote, «TF,» [В Интернете]. Available: http://wiki.ros.org/tf. [Дата обращения: 2020].

[45] Е. П. Кружков, «Держатель для стереокамеры, 3D модель,» 2020. [В Интернете]. Available: https://a360.co/2BCytkM.

[46] T. Dirk, «rqt_graph,» [В Интернете]. Available: wiki.ros.org/rqt_graph. [Дата обращения: 2020].

[47] Е. П. Кружков, «rtabmap.launch,» 2020. [В Интернете]. Available: https://github.com/Quetalas/SLAM-launches/blob/master/rtabmap.launch.

[48] Е. П. Кружков, «main.launch,» 2020. [В Интернете]. Available: https://github.com/Quetalas/SLAM-launches/blob/master/main.launch.

ПРИЛОЖЕНИЕ 1

Код Python для создания наборов калибровочных снимков (StereoCamera.py)

```
import cv2
import os
import numpy as np

def show_stereo_image(image, is_success=True):
    if is_success:
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        left_gray = gray_image[:, 0:320]
        right_gray = gray_image[:, 320:640]

        ret_left, corners_left = cv2.findChessboardCorners(left_gray, (9, 6), None)
        ret_right, corners_right = cv2.findChessboardCorners(right_gray, (9, 6), None)

        image_to_show = np.copy(image)
        left_image = cv2.drawChessboardCorners(image_to_show[:, 0:320, :], (9,6),
corners_left, ret_left)
        right_image = cv2.drawChessboardCorners(image_to_show[:, 320:640, :], (9, 6),
corners_right, ret_right)

        image_to_show = np.concatenate((left_image, right_image), axis=1)
        cv2.imshow('frame', image_to_show)


def find_last_dir(path_where_search):
    last_dir = -1
    for exist_dir in os.listdir(path_where_search):
        if last_dir <= int(exist_dir):
            last_dir = int(exist_dir)
    return str(last_dir)


def is_empty(dir_path):
    return len(os.listdir(dir_path)) == 0

def get_the_path_to_save(save_into_last_dir=False):
    working_dir = os.getcwd()
    samples_dir_name = "images_samples"
    path_to_samples = os.path.join(working_dir, samples_dir_name)

    if not os.path.isdir(path_to_samples):
        os.mkdir(path_to_samples)

    last_dir_number = find_last_dir(path_to_samples)
    if last_dir_number == '-1':
        last_dir_number = "0"
        path_to_save = os.path.join(path_to_samples, last_dir_number)
        os.mkdir(path_to_save)
    else:
        path_to_save = os.path.join(path_to_samples, last_dir_number)

    if save_into_last_dir or is_empty(path_to_save):
       return path_to_save

    new_dir_number = str(int(last_dir_number) + 1)
```

```
    path_to_save = os.path.join(path_to_samples, new_dir_number)
    os.mkdir(path_to_save)

    return path_to_save


def save_image(image, path_to_save, image_number):

    left_dir = os.path.join(path_to_save, "left")
    right_dir = os.path.join(path_to_save, "right")

    if not os.path.isdir(left_dir):
        os.mkdir(left_dir)

    if not os.path.isdir(right_dir):
        os.mkdir(right_dir)

    left_image_path = os.path.join(left_dir, "{}.jpg".format(image_number))
    right_image_path = os.path.join(right_dir, "{}.jpg".format(image_number))

    left_image = image[:,0:320,:]
    right_image = image[:,320:640,:]

    if cv2.imwrite(left_image_path, left_image) and \
            cv2.imwrite(right_image_path, right_image):

        print(image_number + 1, ' images are saved')
        return True
    print("image isn't saved")
    return False


if __name__ == '__main__':
    stereo_camera = cv2.VideoCapture(1)
    stereo_camera.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    stereo_camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

    path_to_save = get_the_path_to_save(save_into_last_dir=False)
    num_saves = 0
    while True:
        ret, frame = stereo_camera.read()
        show_stereo_image(frame, is_success=ret)
        key = cv2.waitKey(100)
        if key == ord('s'):
            if save_image(frame, path_to_save, num_saves):
                num_saves += 1

        if key == ord('q'):
            break
```

ПРИЛОЖЕНИЕ 2

Код для вычисления средних значений (averageMinMax.m)

```
function [average] = averageMinMax(array, min, max)

n = length(array);
sum =0;

for i = 1 : n
    if array(i) >= min && array(i) <= max
        sum = sum + array(i);
    else
        n = n - 1;
    end
end

average = sum / n;
end
```

ПРИЛОЖЕНИЕ 3

Код Matlab для калибровки стереокамеры (calibrate.m)

```
squareSize = 37.3;  % in units of 'millimeters'
num_samples = 20;
num_images = 20;

stereo_params_array = {};
for i = 0 : num_samples - 1
    imageFileNames1 = {'','','','','','','','','','',...
        '','','','','','','','','','',''};
    imageFileNames2 = {'','','','','','','','','','',...
        '','','','','','','','','','',''};
    disp(i)
    for j = 0 : num_images - 1
        path1 = "C:\Users\Evgen\Documents\My\Projects\stereocamera\images_samples\" + i + "\left\" + j
+ ".jpg";
        imageFileNames1{j+1} = char(path1);

        path2 = "C:\Users\Evgen\Documents\My\Projects\stereocamera\images_samples\" + i + "\right\" +
j + ".jpg";
        imageFileNames2{j+1} = char(path2);
    end

    % Detect checkerboards in images
    [imagePoints, boardSize, imagesUsed] = detectCheckerboardPoints(imageFileNames1, imageFileNames2);

    % Generate world coordinates of the checkerboard keypoints
    worldPoints = generateCheckerboardPoints(boardSize, squareSize);

    % Read one of the images from the first stereo pair
    I1 = imread(imageFileNames1{1});
    [mrows, ncols, ~] = size(I1);

    % Calibrate the camera
    [stereoParams, pairsUsed, estimationErrors] = estimateCameraParameters(imagePoints, worldPoints,
...
        'EstimateSkew', true, 'EstimateTangentialDistortion', true, ...
        'NumRadialDistortionCoefficients', 3, 'WorldUnits', 'millimeters', ...
        'InitialIntrinsicMatrix', [], 'InitialRadialDistortion', [], ...
        'ImageSize', [mrows, ncols]);

    stereo_params_array{i+1} = stereoParams;

end

fx_L = zeros(1,20);
fy_L = zeros(1,20);
cx_L = zeros(1,20);
cy_L = zeros(1,20);

k1_L = zeros(1,20);
k2_L = zeros(1,20);
k3_L = zeros(1,20);
p1_L = zeros(1,20);
```

```
p2_L = zeros(1,20);


fx_R = zeros(1,20);
fy_R = zeros(1,20);
cx_R = zeros(1,20);
cy_R = zeros(1,20);

k1_R = zeros(1,20);
k2_R = zeros(1,20);
k3_R = zeros(1,20);
p1_R = zeros(1,20);
p2_R = zeros(1,20);

tx = zeros(1,20);
ty = zeros(1,20);
tz = zeros(1,20);

r11 = zeros(1,20); r12 = zeros(1,20); r13 = zeros(1,20);
r21 = zeros(1,20); r22 = zeros(1,20); r23 = zeros(1,20);
r31 = zeros(1,20); r32 = zeros(1,20); r33 = zeros(1,20);

for i = 1 : num_samples

    fx_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.FocalLength(1);
    fy_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.FocalLength(2);
    cx_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.PrincipalPoint(1);
    cy_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.PrincipalPoint(2);

    k1_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.RadialDistortion(1);
    k2_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.RadialDistortion(2);
    k3_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.RadialDistortion(3);
    p1_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.TangentialDistortion(1);
    p2_L(i) = stereo_params_array{i}.CameraParameters1.Intrinsics.TangentialDistortion(2);

    fx_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.FocalLength(1);
    fy_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.FocalLength(2);
    cx_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.PrincipalPoint(1);
    cy_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.PrincipalPoint(2);

    k1_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.RadialDistortion(1);
    k2_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.RadialDistortion(2);
    k3_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.RadialDistortion(3);
    p1_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.TangentialDistortion(1);
    p2_R(i) = stereo_params_array{i}.CameraParameters2.Intrinsics.TangentialDistortion(2);

    tx(i) = stereo_params_array{i}.TranslationOfCamera2(1);
    ty(i) = stereo_params_array{i}.TranslationOfCamera2(2);
    tz(i) = stereo_params_array{i}.TranslationOfCamera2(3);

    r11(i) = stereo_params_array{i}.RotationOfCamera2(1,1);
    r12(i) = stereo_params_array{i}.RotationOfCamera2(1,2);
    r13(i) = stereo_params_array{i}.RotationOfCamera2(1,3);

    r21(i) = stereo_params_array{i}.RotationOfCamera2(2,1);
    r22(i) = stereo_params_array{i}.RotationOfCamera2(2,2);
    r23(i) = stereo_params_array{i}.RotationOfCamera2(2,3);
```

```
    r31(i) = stereo_params_array{i}.RotationOfCamera2(3,1);
    r32(i) = stereo_params_array{i}.RotationOfCamera2(3,2);
    r33(i) = stereo_params_array{i}.RotationOfCamera2(3,3);
end


% average_fx_L = averageMinMax(fx_L, 274.5, 276)
% average_fx_R = averageMinMax(fx_R, 275, 276.5)
%
% average_fy_L = averageMinMax(fy_L, 274.5, 275.6)
% average_fy_R = averageMinMax(fy_R, 274.9, 276.5)
%
% average_cx_L = averageMinMax(cx_L, 0, 157)
% average_cx_R = averageMinMax(cx_R, 0, 169.3)
%
% average_cy_L = averageMinMax(cy_L, 252.3, 254)
% average_cy_R = averageMinMax(cy_R, 258, 259.3)
%
% average_k1_L = averageMinMax(k1_L, -0.435, -0.425)
% average_k2_L = averageMinMax(k2_L, 0.21, 0.24)
% average_k3_L = averageMinMax(k3_L, -1, -0.05)
%
% average_k1_R = averageMinMax(k1_R, -1, -0.4268)
% average_k2_R = averageMinMax(k2_R, 0.21, 1)
% average_k3_R = averageMinMax(k3_R, -1, -0.005)
%
%
% average_p1_L = averageMinMax(p1_L, -1, 0)
% average_p2_L = averageMinMax(p2_L, 0, 1)
%
% average_p1_R = averageMinMax(p1_R, -1, 0)
% average_p2_R = averageMinMax(p2_R, -1*10^(-3), 1)
average_fx_L = averageMinMax(fx_L, -1000, 1000)
average_fx_R = averageMinMax(fx_R, -1000, 1000)

average_fy_L = averageMinMax(fy_L, -1000, 1000)
average_fy_R = averageMinMax(fy_R, -1000, 1000)

average_cx_L = averageMinMax(cx_L, -1000, 1000)
average_cx_R = averageMinMax(cx_R, -1000, 1000)

average_cy_L = averageMinMax(cy_L, -1000, 1000)
average_cy_R = averageMinMax(cy_R, -1000, 1000)

average_k1_L = averageMinMax(k1_L, -1000, 1000)
average_k2_L = averageMinMax(k2_L, -1000, 1000)
average_k3_L = averageMinMax(k3_L, -1000, 1000)

average_k1_R = averageMinMax(k1_R, -1000, 1000)
average_k2_R = averageMinMax(k2_R, -1000, 1000)
average_k3_R = averageMinMax(k3_R, -1000, 1000)


average_p1_L = averageMinMax(p1_L, -1000, 1000)
average_p2_L = averageMinMax(p2_L, -1000, 1000)
```

```
average_p1_R = averageMinMax(p1_R, -1000, 1000)
average_p2_R = averageMinMax(p2_R, -1000, 1000)

figure();
subplot(2,1,1);
plot(1:num_samples, fx_L, "red", 1:num_samples, fx_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_fx_L, "--red", 1:num_samples,
ones(num_samples)*average_fx_R, "--blue");
grid('on');
title('Фокусное расстояние по направлению x');
ylabel('f_x [пиксели]');
xlabel('Набор №');
legend('левая камера', 'правая камера');

subplot(2,1,2);
plot(1:num_samples, fy_L, "red", 1:num_samples, fy_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_fy_L, "--red", 1:num_samples,
ones(num_samples)*average_fy_R, "--blue");
grid('on');
title('Фокусное расстояние по направлению y');
ylabel('f_y [пиксели]');
xlabel('Набор №');
legend('левая камера', 'правая камера');

figure();
subplot(2,1,1);
plot(1:num_samples, cx_L, "red", 1:num_samples, cx_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_cx_L, "--red", 1:num_samples,
ones(num_samples)*average_cx_R, "--blue");
grid('on');
title('x координата главной точки (c_x)');
ylabel('c_x [пиксели]');
xlabel('Набор №');
legend('левая камера', 'правая камера');

subplot(2,1,2);
plot(1:num_samples, cy_L, "red", 1:num_samples, cy_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_cy_L, "--red", 1:num_samples,
ones(num_samples)*average_cy_R, "--blue");
grid('on');
title('y координата главной точки (c_y)');
ylabel('c_y [пиксели]');
xlabel('Набор №');
legend('левая камера', 'правая камера');


figure();
subplot(3,1,1);
plot(1:num_samples, k1_L, "red", 1:num_samples, k1_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_k1_L, "--red", 1:num_samples,
ones(num_samples)*average_k1_R, "--blue");
grid('on');
```

```
title('Коэффициент радиальной дисторсии k_1');
ylabel('k_1');
xlabel('Набор №');
legend('левая камера', 'правая камера');

subplot(3,1,2);
plot(1:num_samples, k2_L, "red", 1:num_samples, k2_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_k2_L, "--red", 1:num_samples,
ones(num_samples)*average_k2_R, "--blue");
grid('on');
title('Коэффициент радиальной дисторсии k_2');
ylabel('k_2');
xlabel('Набор №');
legend('левая камера', 'правая камера');

subplot(3,1,3);
plot(1:num_samples, k3_L, "red", 1:num_samples, k3_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_k3_L, "--red", 1:num_samples,
ones(num_samples)*average_k3_R, "--blue");
grid('on');
title('Коэффициент радиальной дисторсии k_3');
ylabel('k_3');
xlabel('Набор №');
legend('левая камера', 'правая камера');


figure();
subplot(2,1,1);
plot(1:num_samples, p1_L, "red", 1:num_samples, p1_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_p1_L, "--red", 1:num_samples,
ones(num_samples)*average_p1_R, "--blue");
grid('on');
title('Коэффициент тангенциальной дисторсии p_1');
ylabel('p_1');
xlabel('Набор №');
legend('левая камера', 'правая камера');

subplot(2,1,2);
plot(1:num_samples, p2_L, "red", 1:num_samples, p2_R, "blue");
hold on;
plot(1:num_samples, ones(num_samples)*average_p2_L, "--red", 1:num_samples,
ones(num_samples)*average_p2_R, "--blue");
grid('on');
title('Коэффициент тангенциальной дисторсии p_2');
ylabel('p_2');
xlabel('Набор №');
legend('левая камера', 'правая камера');

figure();
subplot(2,1,1);
plot(1:num_samples, tx, 1:num_samples, ty, 'green', 1:num_samples, tz, 'red');
grid('on');
title('Элементы вектора переноса от правой камеры к левой');
ylabel('Миллиметры');
```

```
xlabel('Набор №');
legend('t_x', 't_y', 't_z');

subplot(2,1,2);
plot(1:num_samples, r11, 1:num_samples, r12, 1:num_samples, r13,...
    1:num_samples, r21, 1:num_samples, r22, 1:num_samples, r23,...
    1:num_samples, r31, 1:num_samples, r32, 1:num_samples, r33);
grid('on');
title('Элементы матрицы поворота от правой камеры к левой');
xlabel('Набор №');
legend('r11', 'r12', 'r13',...
    'r21', 'r22', 'r23',...
    'r31', 'r32', 'r33');

average_tx = averageMinMax(tx, -1000, 1000)
average_ty = averageMinMax(ty, -1000, 1000)
average_tz = averageMinMax(tz, -1000, 1000)

average_r11 = averageMinMax(r11, -1000, 1000)
average_r12 = averageMinMax(r12, -1000, 1000)
average_r13 = averageMinMax(r13, -1000, 1000)
average_r21 = averageMinMax(r21, -1000, 1000)
average_r22 = averageMinMax(r22, -1000, 1000)
average_r23 = averageMinMax(r23, -1000, 1000)
average_r31 = averageMinMax(r31, -1000, 1000)
average_r32 = averageMinMax(r32, -1000, 1000)
average_r33 = averageMinMax(r33, -1000, 1000)
```

ПРИЛОЖЕНИЕ 4

Код C++ для разделения сшитых изображений стереокамеры (*side_x_side_stereo*)

```
#include <ros/ros.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <image_transport/image_transport.h>
#include <camera_info_manager/camera_info_manager.h>
#include <tf/transform_broadcaster.h>
#include <ros/console.h>
// If non-zero, outputWidth and outputHeight set the size of the output images.
// If zero, the outputWidth is set to 1/2 the width of the input image, and
// outputHeight is the same as the height of the input image.
int outputWidth, outputHeight;

// Input image subscriber.
ros::Subscriber imageSub;

// Left and right image publishers.
image_transport::Publisher leftImagePublisher, rightImagePublisher;

// Left and right camera info publishers and messages.
ros::Publisher leftCameraInfoPublisher, rightCameraInfoPublisher;
sensor_msgs::CameraInfo leftCameraInfoMsg, rightCameraInfoMsg;

// Camera info managers.
camera_info_manager::CameraInfoManager *left_cinfo_;
camera_info_manager::CameraInfoManager *right_cinfo_;


// Image capture callback.
void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
	// Get double camera image.
	cv_bridge::CvImagePtr cvImg = cv_bridge::toCvCopy(msg, "rgb8");
	cv::Mat image = cvImg->image;

	// If there are any subscribers to either output topic then publish images
	// on them.
	if (leftImagePublisher.getNumSubscribers() > 0 ||
		rightImagePublisher.getNumSubscribers() > 0)
	{
		// Define the relevant rectangles to crop.
		cv::Rect leftROI, rightROI;
		leftROI.y = rightROI.y = 0;
		leftROI.width = rightROI.width = image.cols / 2;
		leftROI.height = rightROI.height = image.rows;
		leftROI.x = 0;
		rightROI.x = image.cols / 2;

		// Crop images.
		cv::Mat leftImage = cv::Mat(image, leftROI);
		cv::Mat rightImage = cv::Mat(image, rightROI);

		// Apply scaling, if specified.
		bool use_scaled;
```

```
		cv::Mat leftScaled, rightScaled;
		if (use_scaled = (outputWidth > 0 && outputHeight > 0))
		{
			cv::Size sz = cv::Size(outputWidth, outputHeight);
			cv::resize(leftImage, leftScaled, sz);
			cv::resize(rightImage, rightScaled, sz);
		}

		// Publish.
		cv_bridge::CvImage cvImage;
		sensor_msgs::ImagePtr img;
		cvImage.encoding = "rgb8";
		cvImage.header.frame_id = "camera_link";
		//cvImage.header.stamp =  ros::Time::now();
		cvImage.header.stamp = cvImg->header.stamp;
		if (leftImagePublisher.getNumSubscribers() > 0
			|| leftCameraInfoPublisher.getNumSubscribers() > 0)
		{
			cvImage.image = use_scaled ? leftScaled : leftImage;
			img = cvImage.toImageMsg();
			leftImagePublisher.publish(img);
			leftCameraInfoMsg.header.stamp = img->header.stamp;
			leftCameraInfoPublisher.publish(leftCameraInfoMsg);
		}
		if (rightImagePublisher.getNumSubscribers() > 0
			|| rightCameraInfoPublisher.getNumSubscribers() > 0)
		{
			cvImage.image = use_scaled ? rightScaled : rightImage;
			img = cvImage.toImageMsg();
			rightImagePublisher.publish(img);
			rightCameraInfoMsg.header.stamp = img->header.stamp;
			rightCameraInfoPublisher.publish(rightCameraInfoMsg);
		}
	}
}


int main(int argc, char** argv)
{
	ros::init(argc, argv, "sxs_stereo");
	ros::NodeHandle nh("");
	ros::NodeHandle nh_left(nh, "left");
	ros::NodeHandle nh_right(nh, "right");

	image_transport::ImageTransport it(nh);

	// Allocate and initialize camera info managers.
	left_cinfo_ =
		new camera_info_manager::CameraInfoManager(nh_left);
	right_cinfo_ =
		new camera_info_manager::CameraInfoManager(nh_right);
	left_cinfo_->loadCameraInfo("file:///home/evgeny/.ros/left/camera_info/camera.yaml");
	right_cinfo_->loadCameraInfo("file:///home/evgeny/.ros/right/camera_info/camera.yaml");

	// Pre-fill camera_info messages.
	leftCameraInfoMsg = left_cinfo_->getCameraInfo();
	rightCameraInfoMsg = right_cinfo_->getCameraInfo();

	// Load node settings.
	std::string inputImageTopic, leftOutputImageTopic, rightOutputImageTopic,
		leftCameraInfoTopic, rightCameraInfoTopic;
	nh.param("input_image_topic", inputImageTopic,
```

```
            std::string("/stereocamera/image_raw"));
        ROS_INFO("input topic to stereo splitter=%s\n", inputImageTopic.c_str());
        nh.param("left_output_image_topic", leftOutputImageTopic,
            std::string("left/image_raw"));
        nh.param("right_output_image_topic", rightOutputImageTopic,
            std::string("right/image_raw"));
        nh.param("left_camera_info_topic", leftCameraInfoTopic,
            std::string("camera_info"));
        nh.param("right_camera_info_topic", rightCameraInfoTopic,
            std::string("camera_info"));
        nh.param("output_width", outputWidth, 0);  // 0 -> use 1/2 input width.
        nh.param("output_height", outputHeight, 0);  // 0 -> use input height.


        // Register publishers and subscriber.
        imageSub = nh.subscribe(inputImageTopic.c_str(), 2, &imageCallback);
        leftImagePublisher = it.advertise(leftOutputImageTopic.c_str(), 1);
        rightImagePublisher = it.advertise(rightOutputImageTopic.c_str(), 1);
        leftCameraInfoPublisher = nh_left.advertise<sensor_msgs::CameraInfo>
            (leftCameraInfoTopic.c_str(), 1, true);
        rightCameraInfoPublisher = nh_right.advertise<sensor_msgs::CameraInfo>
            (rightCameraInfoTopic.c_str(), 1, true);

        // Run node until cancelled.
        ros::spin();

        // De-allocate CameraInfoManagers.
        left_cinfo_ -> ~CameraInfoManager();
        right_cinfo_ -> ~CameraInfoManager();
}
```