

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Волгоградский государственный университет»
Кафедра *Компьютерных наук и экспериментальной математики*

**СИСТЕМА УПРАВЛЕНИЯ И ФОРМИРОВАНИЯ
ПОВЕДЕНЧЕСКОЙ СТРАТЕГИИ АВТОНОМНОГО МОБИЛЬНОГО
РОБОТА НА ОСНОВЕ ВИЗУАЛЬНОГО АНАЛИЗА
ОКРУЖАЮЩЕГО ПРОСТРАНСТВА
ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(бакалаврская работа)**

по направлению подготовки 02.03.03 Математическое обеспечение и
администрирование информационных систем
профиль «Параллельное программирование»

ВЫПОЛНИЛ:

студент гр. МОС-161
Курбанов Эльдар Ровшанович

НАУЧНЫЙ РУКОВОДИТЕЛЬ:

зав. кафедрой *КНЭМ*
д.ф.-м.н., профессор
Клячин Владимир Александрович

КОНСУЛЬТАНТ:

ст. преп. кафедры *КНЭМ*
Гордеев Алексей Юрьевич

РАБОТА ДОПУЩЕНА К ЗАЩИТЕ:

зав. кафедрой *КНЭМ*
д.ф.-м.н., профессор
Клячин Владимир Александрович

«30» мая 2020 г.

(протокол № ____ заседания кафедры)

Волгоград 2020

Оглавление

| | Стр. |
|---|-----------|
| Введение | 4 |
| Глава 1. Теория | 6 |
| 1.1 Поведение робота | 6 |
| 1.2 Анализ окружающего пространства | 7 |
| 1.3 Об управлении | 9 |
| Глава 2. Анализ | 10 |
| 2.1 Анализ окружающего пространства | 10 |
| 2.2 Шасси и система управления | 11 |
| 2.3 Поведенческая стратегия робота | 11 |
| 2.4 Вычислительная составляющая | 12 |
| 2.5 Известные аналоги | 13 |
| 2.5.1 Nvidia Kaya | 13 |
| 2.5.2 Nvidia JetBot | 15 |
| 2.5.3 Сравнение с аналогами | 16 |
| Глава 3. Практика | 17 |
| 3.1 Мобильный автономный робот | 17 |
| 3.1.1 Подбор шасси | 17 |
| 3.1.2 Движение шасси | 18 |
| 3.2 Визуальный анализ пространства | 20 |
| 3.3 Формирование поведенческой стратегии робота | 21 |
| 3.3.1 Исследование пространства | 21 |
| 3.3.2 Подъезд к целевому объекту | 24 |
| 3.4 Подробнее о программной части робота | 25 |
| 3.4.1 ROS | 25 |
| 3.4.2 Концепции ROS | 26 |
| 3.4.3 Узлы, используемые на роботе | 28 |
| Заключение | 40 |

| | |
|--|-----------|
| Список литературы | 41 |
| Список рисунков | 44 |
| Приложение А. Форматы сообщений ROS, используемые в ВКР . . | 45 |
| Приложение Б. Реализованный исходный код | 48 |

Введение

Актуальность данной работы обусловлена общей автоматизацией и «роботизацией» деятельности человека в условиях современной реальности [1]. Решение поставленной задачи позволит в дальнейшем создать робота, умеющего не только объезжать разного вида помещения, но и ещё выполняющего какую-либо полезную функцию. Например, распознавание опасных объектов окружающего пространства или исследование состава атмосферы в каком-либо замкнутом пространстве.

В настоящий момент поставленная данной работой задача формирования поведенческой стратегии и управления роботом выполнена полностью. Однако, она требует значительных улучшений для каких-либо конкретных условий работы. Например, если испытуемый робот окажется на улице, то может случиться так, что целевой объект может быть так и не найден, в связи с тем, что окружающее пространство окажется слишком широким для угла обзора камеры, установленной на робота. Соответственно, данный конкретный случай должен быть учтён в алгоритме движения робота, но это не является задачей данной работы.

Целью данной работы является создание системы автоматического управления робота с учётом данных, получаемых от окружающего пространства и прежде всего создание самого тестируемого образца робота и его аппаратной системы управления.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Исследовать предметную область робототехники¹ (аппаратную и программную часть);
2. Изучить существующие известные аналоги (в т.ч. зарубежные) и продумать как сделать робота ещё лучше;
3. Закупить необходимое оборудование, уложившись при этом в маленький бюджет;
4. Разработать схему управления роботом и соответствующее ПО;
5. Протестировать созданное изделие.

Научная новизна:

¹Робототехника не изучалась на протяжении всего курса обучения в университете.

1. Впервые в России был сделан робот с одновременным использованием технологии YDLIDAR, движением и распознаванием объектов окружающего пространства на базе платформы NVIDIA Jetson NANO²;
2. Создана программно-аппаратная база, на основе которой можно сделать робота, выполняющего иной функционал.

Практическая значимость данной работы заключается в том, что была решена задача создания своего собственного алгоритма движения для робота на базе относительно новой и ещё мало изученной платформы Jetson NANO со своим алгоритмом езды и следованием за целевыми объектами.

Методология и методы исследования. При разработке данной системы управления и формирования поведенческой стратегии автономного мобильного робота использовались такие методы эмпирического исследования, как наблюдение и эксперимент, а к методам теоретического исследования - анализ и синтез и восхождение от абстрактного к конкретному.

Объем и структура работы. Выпускная квалификационная работа состоит из введения, трёх глав, заключения и двух приложений. Полный объём ВКР составляет 53 страницы, включая 27 рисунков. Список литературы содержит 19 наименований.

²Возможно, это происходит не впервые, но других таких известных случаев не нашлось

Глава 1. Теория

Теоретическая часть данной работы будет описывать ту предметную область с которой пришлось столкнуться в ходе выполнения практической части.

1.1 Поведение робота

Прежде чем перейти к определению поведения будущего робота мы должны определить его главную задачу. А именно - поиск целевых объектов в замкнутом пространстве.

Для того чтобы выполнить данную задачу робот должен уметь объезжать то замкнутое пространство в котором он находится, распознавать объекты и уметь подъезжать к найденному целевому объекту. Здесь можно выделить две возможные стратегии, которые можно применять к данной задаче:

1. Сначала выполняется обезд всего доступного пространства, во время которого строится карта местности, а затем происходит выполнение на ней поиска целевых объектов;
2. Целевой объект ищется непосредственно во время обезда пространства. При этом обезд пространства происходит без составления карты.

К преимуществам первого подхода можно записать:

- Помимо поиска целевых объектов выполняется полное сканирование местности, что может пригодится для других задач;
- Возможно более «умное» построение маршрута при помощи, например, таких алгоритмов как A*¹;
- Можно найти все целевые объекты в данном замкнутом пространстве и примерно оценить их местоположение на отсканированной карте местности.

К недостаткам первого подхода относятся:

¹A* - алгоритм поиска по первому наилучшему совпадению на графике[2, с. 218].

- Долгое время работы алгоритма: сначала нужно все объездить, оценить обстановку, а затем искать объекты;
- Требуется более сложная алгоритмическая составляющая: как минимум роботу нужно научиться прокладывать маршруты на динамически строящейся карте и уметь определять себя и целевые объекты на ней²;

У второго подхода есть хоть и одно, но очень большое преимущество и это относительно «лёгкая» реализация: как в алгоритмическом, так и в плане производительности. Не требуется составлять карт, а значит и решать задачу SLAM³, в связи с этим уменьшается вычислительная нагрузка на робота.

Недостатки второго подхода:

- Время поиска целевого объекта будет зависеть от удачи, так как карты местности не строится и угадать когда робот поедет к целевому объекту не просто;
- Полное сканирование местности не выполняется, а значит не все целевые объекты могут быть найдены в пространстве;

1.2 Анализ окружающего пространства

Основным сенсором при решении задачи визуального анализа окружающего пространства является видеокамера. Для того чтобы анализировать сигнал с видеокамеры требуется решить задачу машинного зрения. А именно требуется каким-то образом обрабатывать полученное изображение и исходя из этого строить стратегию движения.

Например, можно обучить нейронную сеть, которая распознаёт различные объекты и классифицирует их как опасные или целевые. Если робот видит опасный объект, он немедленно должен перестроить свой маршрут так, чтобы не столкнуться с ним. И если робот видит целевой объект, то он наоборот должен подъехать к нему, удостовериться в том, что это именно

²По сути требуется решить задачу SLAM

³SLAM - метод для построения или обновления карты в пространстве с одновременным контролем местоположения и пройденного пути[3, с. 9].

нужный целевой объект и сохранить его местоположение в энергонезависимой памяти.

В качестве дополнительного сенсора для визуального анализа пространства можно использовать также технологию Лидар, которая позволяет также получить некоторое изображение, представляющее собой облако точек, поддающееся анализу. По сути прибор, реализующий технологию Лидар представляет собой дальномер оптического диапазона, который замеряет угол и расстояние до точки (получаются полярные координаты) при помощи лазерного сканирования. Существует два основных типа сканирующих лидаров [4, с. 702]:

1. 3D лидар;
2. 2D лидар.

Первый позволяет получить 3D картинку. Обычно такой лидар оснащён подвижным лазером, который довольно долго сканирует перед собой окружающую местность. Однако, уже сейчас можно найти 3D лидары, которые сканируют с довольно быстрой скоростью [5, с. 308]. Примером результата такой работы может стать картинка, изображённая на Рисунке 1.1. На сегодняшний день такие лидары являются довольно дорогими устройствами.



Рисунок 1.1 — Пример картинки, генерируемой 3D лидаром, представленном на выставке CEATEC 2017 компанией Panasonic в Японии.

Второй соответственно уже создаёт двухмерное облако точек, которое также можно визуализировать в виде картинки, пример которой изображён на Рисунке 1.2. Такой лидар обычно сканирует область вокруг себя и имеет угол обзора 360 градусов. Лазер также является подвижным, но только в этот раз он просто движется вокруг своей оси [6, с. 610].

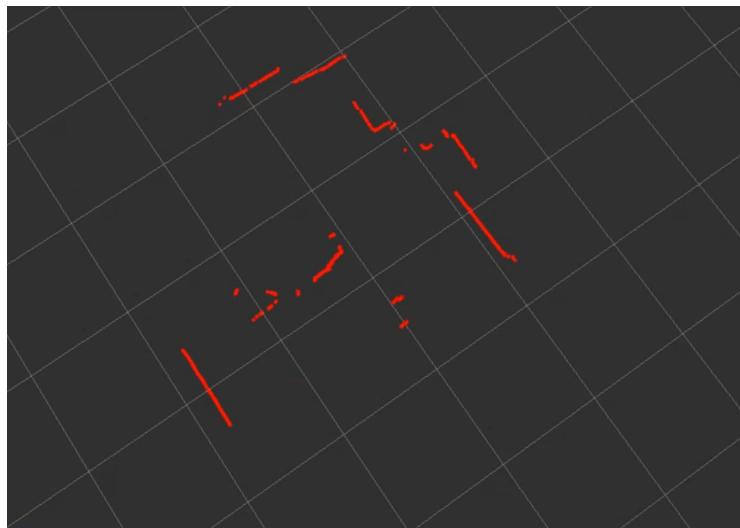


Рисунок 1.2 – Пример картинки, генерируемой 2D лидаром YDLIDAR X4.

1.3 Об управлении

Одна из задач, которую должен уметь решать мобильный автономный робот это задача передвижения. Потому как без него робот уже не будет полностью соответствовать своему критерию «мобильности». Движение для робота, применительно к конкретной задаче данной ВКР просто необходимо.

Для того чтобы робот двигался ему необходимо шасси. Шасси в основной своей массе по своей подвижной части подразделяются на те, что осуществляют езду при помощи гусеничной ленты и те, что ездят на колёсах. Шасси на гусеницах обладают большей проходимостью и мобильностью в следствии того, что гусеницы позволяют, например, разворачивать робота на месте.

Для того чтобы управлять шасси роботу необходимы электродвигатели и контроллер движения для них, но подробнее речь об этом зайдёт в пункте 3.1.2.

Глава 2. Анализ

Данная глава описывает решения, которые были приняты для выполнения основной задачи данной ВКР.

2.1 Анализ окружающего пространства

По ходу анализа задачи данной ВКР было принято решение установить на будущего робота два основных сенсора, речь о которых шла в пункте 1.2: это видеокамера и лазерный сканер, реализующий технологию Лидар¹.

Целью установки Лидара стала необходимость в сборе данных обо всём окружающем пространстве без необходимости совершать полный разворот. Такие данные можно было бы собирать и при помощи такого сенсора, как Xbox Kinect², изображённого на Рисунке 2.1, однако сбор информации об обстановке вокруг требовало бы полного оборота робота вокруг своей оси или установки сенсора на сервопривод. Разумеется, не во всех задачах роботу требуется видеть обстановку вокруг себя, в большинстве случаев роботу достаточно видеть то, что находится перед ним. Однако Лидар позволяет довольно быстро получать динамические данные вокруг робота в том числе, что также является преимуществом.

Целью установки видеокамеры является необходимость в выполнении роботом какой-то дополнительной полезной функции. В случае данной ВКР, в робот был встроен механизм поиска целевых объектов на окружающей местности.

¹Лидар - технология получения и обработки информации об удалённых объектах при помощи активной оптической системы[7, с. 20].

²Xbox Kinect - это бесконтактный сенсорный игровой контроллер, нашедший своё применение не только в игровой индустрии, но и в интерактивных экспозициях, и в робототехнике[8].



Рисунок 2.1 – Игровой сенсор Xbox Kinect, представленный в 2009 году в рамках выставки электронных развлечений Е3.

2.2 Шасси и система управления

В качестве шасси для робота был выбран вариант с гусеницами на ходу, так как это несло большую пользу в практическом плане: это не дорого и обладает преимуществами, которые были описаны в пункте 1.3. Система управления шасси робота должна характеризоваться следующим:

1. Каждая гусеница управляет отдельно;
2. Возможность двигаться вперёд и назад;
3. Управляется простым логическим сигналом (1 - выполнять движение, 0 - не выполнять движение);
4. Программный интерфейс системы управления должен полностью раскрывать возможности аппаратной части.

2.3 Поведенческая стратегия робота

В пункте 1.1 были описаны две возможные стратегии, которые можно применить к данной задаче. Для облегчения задачи на данном этапе разработки было принято решение реализовать более лёгкую стратегию при которой поиск целевого объекта будет выполняться во время объезда про-

странства роботом. При этом карта местности всё же будет строиться и она будет служить для распознавания застревания робота, что очень важно при езде на неровных и мягких поверхностях.

Таким образом, поведенческая стратегия робота в данной ВКР сводится к тому, что робот в общем случае будет ехать вперёд и искать две вещи:

1. Преграду перед собой (распознаётся Лидаром);
2. Целевой объект (распознаётся видеокамерой);

В случае, если впереди была обнаружена преграда, то роботу уже не стоит ехать вперёд (так как он просто ударится), а найти какой-то другой путь. Самым логичным решением в данной ситуации станет поворот налево или направо. О том в какую сторону поворачивать, робот принимает решение на основе облака точек³, которое предоставляет Лидар. В итоге поворот выполняется в ту сторону, где было найдено больше свободного пространства и меньше преград. Подробнее о том, как реализовано распознавание преград и свободного пространства будет описано в пункте 3.3.1.

2.4 Вычислительная составляющая

К вычислительной составляющей мобильного автономного робота предъявляются довольно сильные и строгие требования:

- Компактность (для размещения на корпусе);
- Мощность (требуется в реальном времени обрабатывать показания со всех сенсоров и выполнять движение);
- Энергоэффективность (для большего времени автономной работы);
- Бюджетность (в рамках данного проекта больших финансовых затрат не планировалось).

Было принято решение о том, что вычислительной составляющей будет одноплатный компьютер Nvidia Jetson NANO, изображённый на Рисунке 2.2, так как он соответствует всем изложенным выше требованиям.

К основным характеристикам данного компьютера можно отнести следующие[9]:

³Облако точек в данном конкретном случае будет представлять собой двумерный массив с числами типа float.



Рисунок 2.2 – NVIDIA Jetson Nano - компактный и мощный одноплатный компьютер, представленный в 2019 году.

- Создан специально для встраиваемых систем;
- Архитектура NVIDIA MaxwellTM с 128 ядрами NVIDIA CUDA[®];
- Четырехъядерный процессор ARM[®] Cortex[®]-A57 MPCore;
- Размер 69,6 мм x 45 мм;
- Имеет разъём GPIO и Ethernet.

2.5 Известные аналоги

К известным аналогам разрабатываемого робота, созданных на базе такого же одноплатного компьютера Nvidia Jetson NANO можно причислить роботов от самой компании Nvidia: Jetbot и Kaya. Оба эти робота были созданы для демонстрации возможностей данного одноплатного компьютера.

2.5.1 Nvidia Kaya

Данная модель компактного мобильного автономного робота была представлена на технологической конференции GTC 2019 и в первую очередь предназначается для работы с программным обеспечением Isaac SDK. Робот представлен на Рисунке 2.3.



Рисунок 2.3 – Внешний вид робота NVIDIA Kaya.

Аппаратно данный робот помимо самого Jetson NANO включает в себя пластиковый корпус на трёх колёсах (печатаемый на 3D принтере), 3D камеру LiDAR Intel Real Sense и систему управления. Общая стоимость аппаратной части на момент написания данной ВКР⁴ составляет \$812.87[10].

На компьютер Jetson NANO помимо ОС Ubuntu 18.04 LTS устанавливается ПО Isaac SDK и Isaac SIM. Isaac SDK - это открытая платформа NVIDIA для интеллектуальных роботов. Она предоставляет большой набор мощных алгоритмов, базирующихся на GPU вычислениях⁵ для навигации и управления.

На данном роботе можно запускать различные готовые примеры такие как ручное управление с геймпада Playstation 4, автономное следование за AprilTag, распознавание объектов на нейронной сети DetectNetv2 и алгоритм SLAM (основан на GMapping)[11].

⁴Май 2020 г.

⁵вычисления на видеокарте

2.5.2 Nvidia JetBot

JetBot был представлен на той же конференции, что Nvidia Kaya и является гораздо более доступным вариантом (цена \$226.15) для создания DIY робота (также он в отличии от Kaya имеется в розничной продаже одним комплектом и его не нужно собирать по частям из разных магазинов). Nvidia JetBot изображён на Рисунке 2.4[12].



Рисунок 2.4 – Набор инструментов NVIDIA JetBot от Waveshare.

Аппаратно он состоит из всё той же Nvidia Jetson Nano, двух электромоторов с драйвером в комплекте и CSI видеокамеры Sony IMX219.

Программная часть поставляется готовым образом на базе Ubuntu 18.04 в формате ISO для прошивки MicroSD карты.

Из доступных примеров имеется простое ручное управление через кнопки на экране с возможностью прямой трансляции изображения видеокамеры на экран в браузере и нейросеть, автономное движение по поверхности с распознанием препятствий и пропастей в окружающем пространстве при помощи нейросети на основе получаемого видеосигнала, также имеется функция следования робота за определённым целевым объектом[13].

2.5.3 Сравнение с аналогами

Робот, разрабатываемый в рамках данной ВКР по большей части сходится с Nvidia JetBot, однако подход к решению задач в корне изменён. Стратегия движения робота в данной ВКР полностью определяется показаниями Лидара, что даёт большую гибкость за счёт того, что Лидар сканирует всю поверхность вокруг себя тогда как видеокамера позволяет видеть только то, что находится непосредственно перед роботом. Таким образом робот, создаваемый в рамках данной ВКР решает уже решённую задачу другим более гибким способом.

Что касается Nvidia Kaaya, то данная модель хоть и оснащена Лидаром, но обладает довольно большим минусом в виде цены за данный продукт. Также установленный там Лидар не может просматривать пространство вокруг себя в силу того что Лидаром является видеокамера, поворот которой занимает гораздо дольше времени по сравнению с лазерным сканером.

Глава 3. Практика

3.1 Мобильный автономный робот

Для решения задачи данной работы необходимо было проводить «живые» тестирования работы алгоритмов. Такая необходимость обусловлена прежде всего тем, что помимо существующей задачи данной ВКР стояла задача в создании робота для распознавания объектов. По этой причине было принято решение делать алгоритмы на реальном роботе¹ с пребыванием данного робота во вполне реальных условиях.

3.1.1 Подбор шасси

Как было сказано в пункте 1.3 данной работы, существует большое количество различных шасси, на которых можно располагать различное оборудование. Наш выбор остановился на гусеничном шасси TS100, заказанном с платформы AliExpress, которое изображено на Рисунке 3.1.

Данное шасси за счёт своих размеров является очень мобильным средством передвижения робота и может проникнуть в относительно узкие для роботов пространства и без проблем оттуда выбраться, не повредившись. Для дополнительного оборудования на данном шасси место тоже нашлось: для этого было принято решение заказать дополнительную металлическую пластину, которая в последствии роль сыграла второго этажа. Шасси с уже установленным вторым этажом можно увидеть на Рисунке 3.2.

¹Задание данной ВКР можно было бы сделать и в любом симуляторе или игровом движке. Однако решение делать всё в реальной жизни сильно усложнило данную задачу.

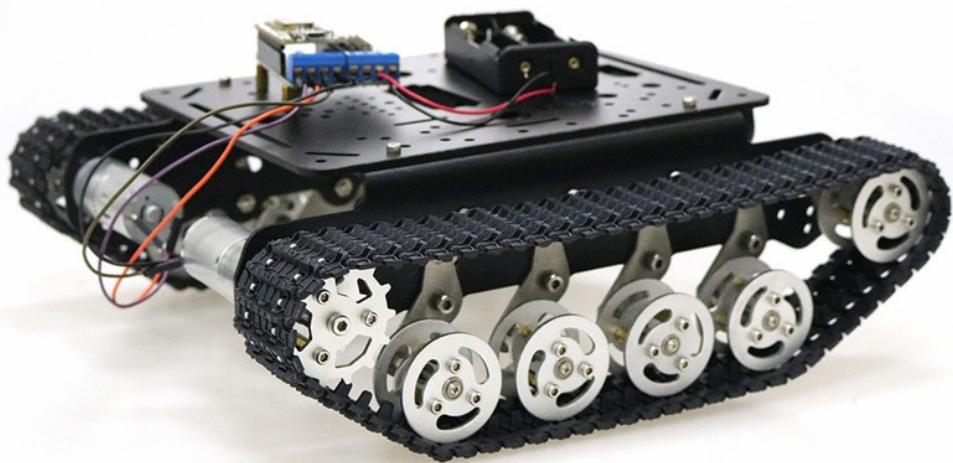


Рисунок 3.1 – Шасси TS100 для самодельного робота.

3.1.2 Движение шасси

К сожалению, по неизвестной причине к данному шасси не присоединился комплектный контроллер движения, который бы принимал команды от компьютера и заставлял двигаться два электродвигателя, установленные на шасси. Поэтому пришлось немного изучить ещё одну предметную область, которая не изучалась в течении университетского курса - электротехнику.

Контроллер двигателей

Компьютер, который будет в последствии установлен на робота будет управлять роботом посредством сигналов с напряжением 3.3В через порт

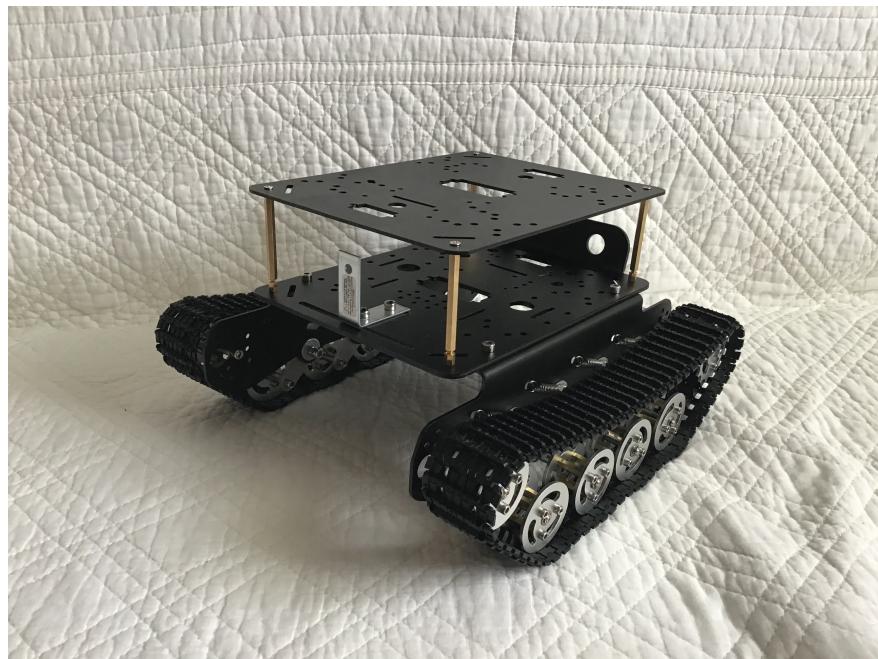


Рисунок 3.2 – Шасси робота без установленного на него оборудования.

GPIO, где 0 (или по-другому нет напряжения) - это движение не требуется и 1 (когда есть напряжение +3.3В), когда движение требуется.

Контроллер должен, также, уметь по отдельности управлять двумя гусеницами, заставлять их ездить вперёд и назад. Не мало важна и компактность решения, и энергоэффективность. Это основные требования. Из дополнительных требований можно выделить умение каким-то образом регулировать скорость движения. Общая структура желаемой модели контроллера изображена на Рисунке 3.3.

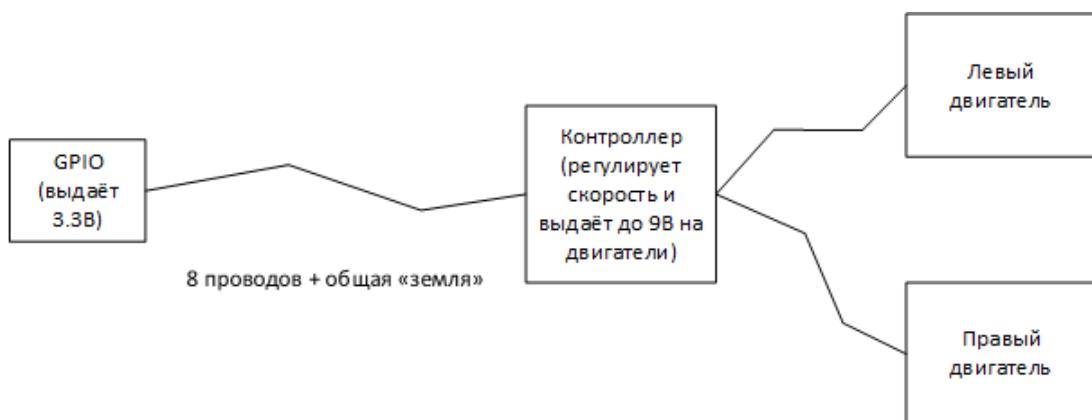


Рисунок 3.3 – Общая структура желаемой модели контроллера.

Таким образом спустя пару экспериментов с текстолитом и монолитными платами получился полноценный контроллер, который умеет управлять роботом с медленной и быстрой скоростями, однако у него были

свои сильные недостатки речь о которых в данной ВКР не зайдёт. Получившийся контроллер изображён на Рисунке 3.4.

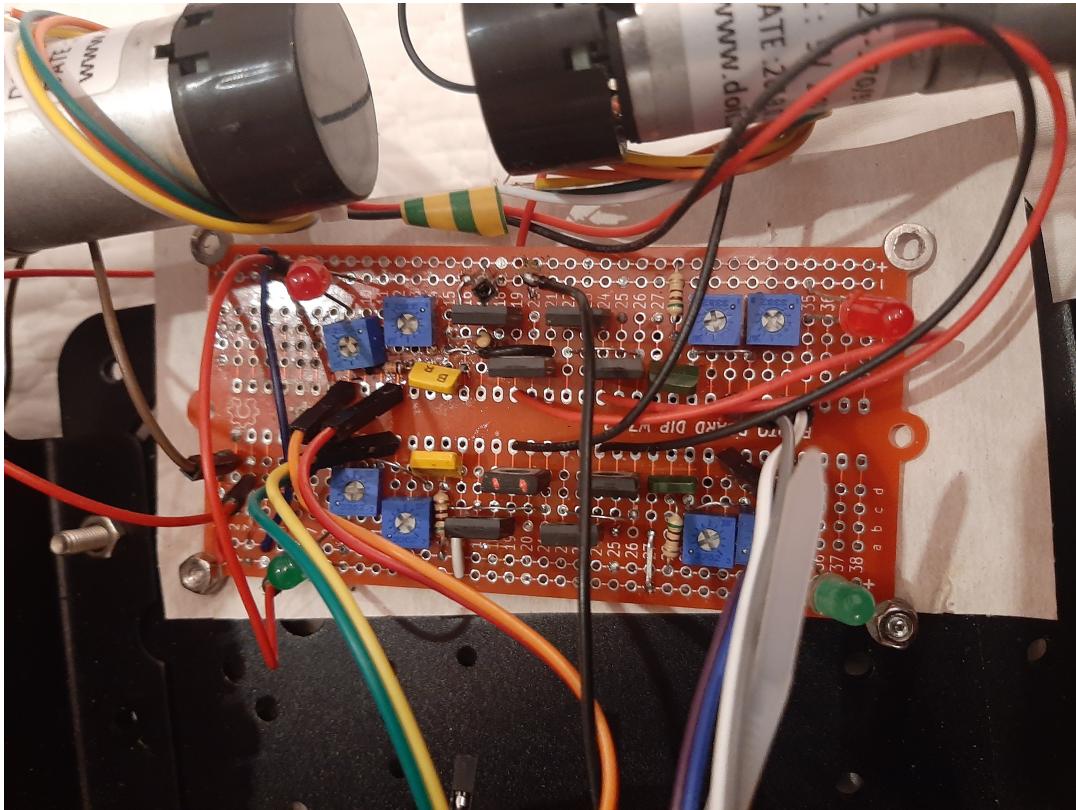


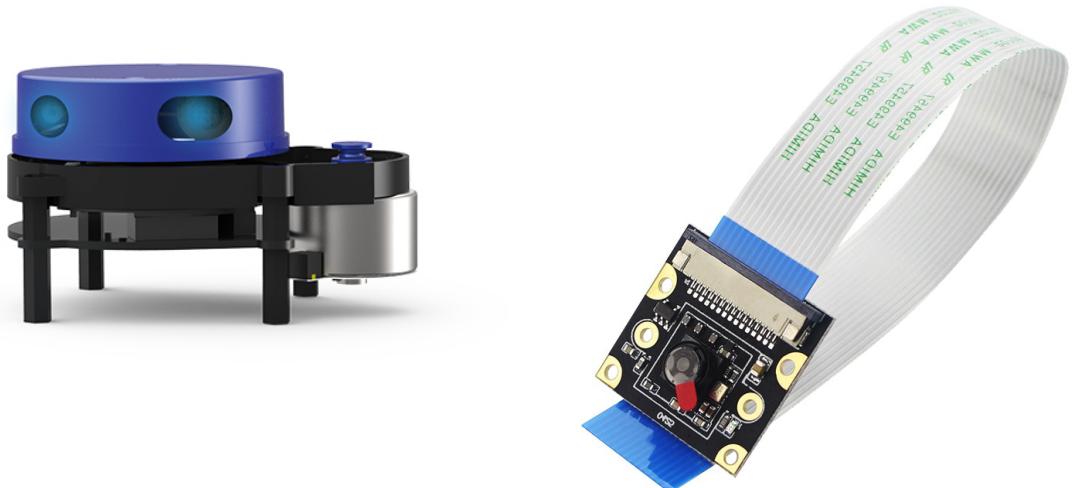
Рисунок 3.4 — Готовый экземпляр контроллера двигателей.

3.2 Визуальный анализ пространства

Для анализа окружающего пространства существует довольно большое количество различных датчиков и прочего оборудования[14]. Но закупать сразу всё не выгодно экономически, затратно в плане места размещения на роботе и расточительно в плане потребления электроэнергии этими самыми датчиками. Также для обработки всех этих сигналов нужны соответствующие вычислительные мощности.

Таким образом робот должен иметь совсем небольшое количество сенсоров и при этом не быть «слепым». Исходя из этих соображений, было решено установить на робота два основных сенсора: лазерный сканер YDLIDAR X4 (изображён на Рисунке 3.5а) и CSI камеру Sony IMX219 (изображена на Рисунке 3.5б). Первый поможет видеть препятствия вокруг

робота, второй сможет «видеть» целевые объекты, размещённые перед роботом.



a) 2D лидар YDLIDAR X4.

б) CSI камера с сенсором Sony IMX219.

3.3 Формирование поведенческой стратегии робота

Основная задача робота - ездить и искать целевые объекты делится на две подзадачи: исследование пространства и подъезд к целевому объекту.

3.3.1 Исследование пространства

Данный режим будет подразумевать под собой то, что робот будет просто ехать вперёд, параллельно разыскивая целевые объекты и объезжать возникшие перед ним препятствия.

Объезд препятствий

Робот должен уметь объезжать хотя-бы самые простейшие препятствия, по типу стен, диванов или прочих перегородок. В идеале, он должен уметь справляться и с тонкими препятствиями по типу ножек стула и мягкие поверхности.

Алгоритм объезда препятствий, представленный на данном роботе сводится к схеме, изображённой на Рисунке 3.6.

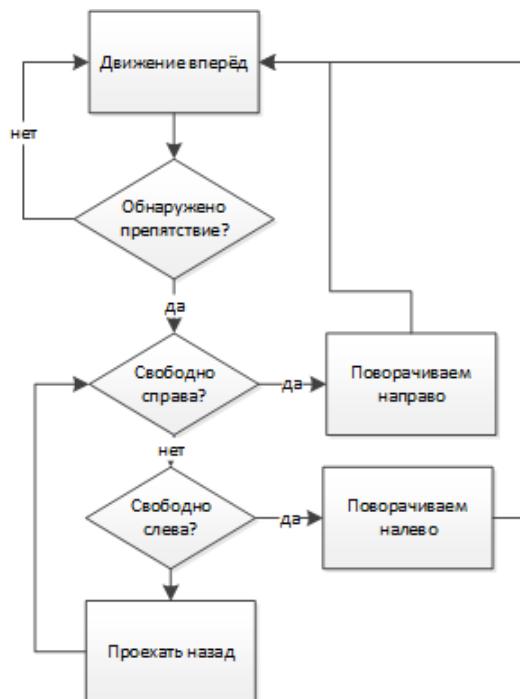


Рисунок 3.6 — Общая схема алгоритма объезда препятствий.

Подсчёт того, где свободнее: слева или справа идёт из соображений того, где находится больше препятствий. Как это считается? Условно робот поделён на несколько направлений. В данном случае он подразделён на «перед», «лево», «право» и «зад». Обозначим значения, которые подсчитываются на этих направлениях, как f , l , r и b соответственно.

Лидар выдаёт данные в формате массива значений float: обозначим его как числовой ряд a . В этом числовом ряду находятся числа, обозначающие расстояние до точки об которое отразился лазер. Чем больше число, тем дальше находится объект об который отразился лазер. Если лазерный сканер не нашёл в этом месте отражения, то он возвращает значение -1.

Фактически, данный массив является аналогом полярных координат, где позиция значения в массиве - это угол, а само значение является расстоянием. Всего этих чисел 720, из чего можно сделать вывод, что цена деления лидара это полградуса.

Таким образом каждый поворот лидара вычисляются 4 переменные²:

$$l = \frac{\sum_{i=90}^{270} a_i}{180} \quad b = \frac{\sum_{i=270}^{450} a_i}{180} \quad r = \frac{\sum_{i=450}^{630} a_i}{180} \quad f = \frac{\sum_{i=630}^{720} a_i + \sum_{i=0}^{90} a_i}{180}$$

После вычисления этих средних арифметических значений по каждой из сторон проверяются значения массива a_i , где $630 < i < 720$ и $0 < i < 90$ (передняя сторона робота) и если среди этих чисел находится хоть одно удовлетворяющее условию $0 < a_i < 0,3$, то считается в данный момент перед роботом находится какое-то препятствие.

Если это так, то далее сравниваются значения ранее высчитанных переменных l и r . Если значение $l > r$, то робот поедет налево, так как слева нашлось меньше препятствий, чем справа. Иначе, роботу следует ехать направо. Однако представленного выше алгоритма ещё недостаточно чтобы объезжать часто встречающиеся препятствия.

Обнаружение застревания

Робот может попасть в ситуацию, когда впереди внезапно образовалась преграда, невидимая для лазерного сканера (например, очень низкая преграда). Для обнаружения застревания при столкновении с такими преградами необходимо как-то понять, что робот перестал двигаться.

Одним из способов понять и распознать застревание может стать анализ облака точек, которые выдаёт LIDAR. Если вектор движения большинства точек на плоскости облака стал достаточно мал, то можно сделать вывод о том, что робот либо плохо двигается, либо вообще застрял.

²Важное замечание: значения -1, когда лазерный сканер не нашёл отражения заменяются на значение 1 для того чтобы значения переменных прибавлялись, а не уменьшались.

В данный проект была встроена система Google Cartographer, которая по облаку точек может строить окружающую карту местности (пример такой карты изображён на Рисунке 3.7), а также определять местоположение робота на ней[15, с. 1]. Информацию о местоположении можно использовать как раз в целях определения застревания. Если в течении секунды координаты робота менялись недостаточно сильно, значит робот застрял.

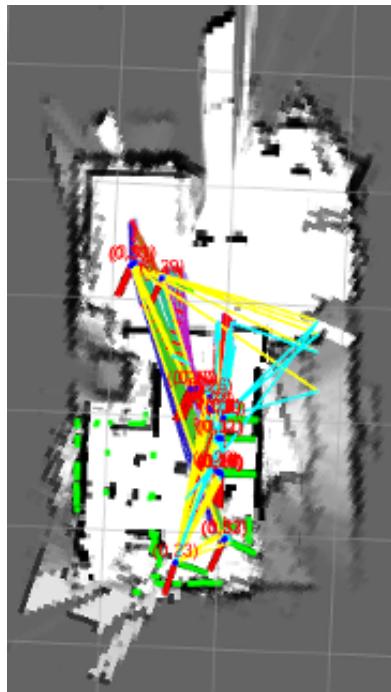


Рисунок 3.7 – Пример карты, сгенерированной Google Cartographer.

Для выезда из застревания используется простой алгоритм, который состоит из 3 шагов:

1. Ехать назад 2 секунды;
2. Выбрать сторону, в которую будет совершён поворот по значениям выше упомянутых переменных l и r ;
3. Ехать дальше.

Как показала практика, этот алгоритм, работает достаточно эффективно для того чтобы не застревать в большинстве ситуаций.

3.3.2 Подъезд к целевому объекту

Данный режим предполагается включать только в случае, если на видеосигнале, получаемом от CSI камеры был распознан целевой объект. В

в этом случае робот останавливается и поворачивается в ту сторону, где расположен центр предполагаемого целевого объекта. Далее робот начинает ехать вперёд и по мере необходимости продолжает центрировать шасси до тех пор пока не подъедет к объекту. Далее робот останавливается, конечная цель робота выполнена: целевой объект найден.

Определение того, что робот подъехал к объекту происходит по размеру прямоугольника, на котором обозначен целевой объект. Если прямоугольник уже достиг краёв кадра видеосигнала, значит робот приблизился к объекту максимально близко. Подъезд вплотную к объекту является не самой лучшей идеей, так как целевым объектом может быть стеклянная бутылка, которую можно просто сбить и разбить.

3.4 Подробнее о программной части робота

На одноплатный компьютер Nvidia Jetson Nano была установлена операционная система Ubuntu LTS 18.04 со специальным от Nvidia программным обеспечением JetPack 4.3, которое предоставляет удобные инструменты для вычислений в области искусственного интеллекта при помощи встроенного в Jetson NANO видеочипа и ядер CUDA. Также на компьютер был установлен фреймворк для программирования роботов ROS, аббревиатура которого расшифровывается как «Операционная система для роботов».

3.4.1 ROS

ROS предоставляет удобные и мощные функции, помогающие разработчикам в таких задачах, как передача сообщений различного типа, распределение вычислений между компьютерами, повторное использование кода и реализация современных алгоритмов для роботизированных приложений[16, с. 7]. В общем случае, ROS представляет собой инструмент, позволяющий связывать несколько независимых программных модулей при

помощи сервисов и узлов, которые могут передавать друг другу сообщения в различном формате. Структура ROS представлена на Рисунке 3.8[16, с. 19].



Рисунок 3.8 – Общая структура Robot Operating System.

Большими преимуществами использования данного фреймворка является возможность передачи сообщений по локальной сети и обширная библиотека уже реализованного ПО, которое можно без относительно больших затрат по времени интегрировать в свой собственный проект. На момент написания данной ВКР глобальный репозиторий ROS Index насчитывает 2120 подключенных к нему сторонних репозиториев и 5827 пакетов. Диаграмму соответствия пакетов в репозитории с версиями ROS можно увидеть на Рисунке 3.9³[17].

3.4.2 Концепции ROS

Ниже приведён список концепций рассматриваемого фреймворка

- **Узел** - это процесс, выполняющий вычисления. Каждый узел написание с использованием клиентских библиотек ROS. Используя

³Версия, используемая на роботе - Melodic

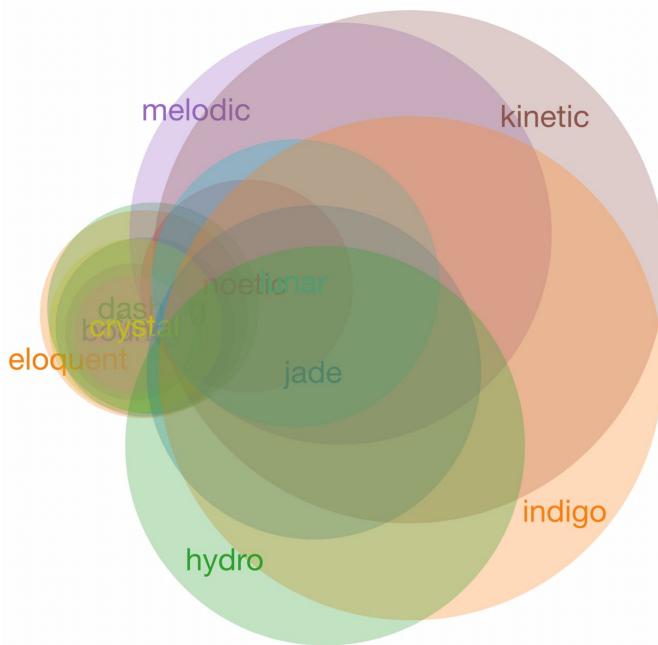


Рисунок 3.9 — Глобальный репозиторий ROS Index и версии ROS.

методы связи, узлы могут общаться друг с другом заранее определённым форматом сообщений и обмениваться данными. Для этого создаются узлы-подписчики, и узлы-публикаторы.

- **Мастер** - обеспечивает регистрацию и работоспособность запущенных узлов.
- **Сообщение** - простая структура данных, содержащая типизированное поле, которое может содержать целый набор данных, отправляемых на другой узел. Помимо стандартных типов сообщений⁴ возможна отправка заранее обозначенных собственных типов сообщений.
- **Тема** - именованная шина данных, используемая узлами для отправки сообщений. Публикующий и подписанный узел не знают о существовании друга друга. Благодаря тому что каждая тема имеет уникальное имя, любой узел может получить доступ к данной теме и отправляет через неё данные, при условии соблюдении заранее оговорённых передаваемых типов, данной темой.
- **Сервисы** - реализация удалённого вызова процедур⁵ в ROS. В некоторых случаях модель связи публикации и подписки может не подходить. В этих случаях и применяют взаимодействия в виде сервисов (схема запрос/ответ), при котором один узел может запросить

⁴Такие как целые, с плавающей точкой, логические, строковые...

⁵RPC

выполнение процедуры для другого узла, ожидая какого-то обязательного ответа⁶[16, с. 20].

3.4.3 Узлы, используемые на роботе

В рамках работы над данной ВКР были реализованы следующие узлы и сервисы:

- Сервис, управляющий сигналами на разъёме GPIO;
- Узел записи видео с видеокамеры;
- Узел распознавания объектов;
- Узел, управляющий движением робота и формирующий поведенческую стратегию робота.

Также в работе используются следующие сторонние узлы:

- Узел передачи изображения с CSI видеокамеры;
- Google Cartographer;
- Узел YDLIDAR.

Общую схему взаимодействия всех узлов можно увидеть на Рисунке 3.10.

Узел видеокамеры

Данный узел был заимствован из репозитория робота JetBot и он публикует изображения в формате сообщения, описанного стандартом ROS `sensor_msgs/Image` (содержание сообщения можно увидеть в Листинге A.1), получаемые из CSI камеры IMX219, подключенной к Nvidia Jetson NANO.

Для получения такого видеосигнала используется библиотека GStreamer и встроенные в образ Linux драйвера на данный сенсор. Пример получаемого изображения показан на Рисунке 3.11.

⁶В случае использования схемы с подписчиками и публикаторами доставка сообщений и ответ не гарантируются

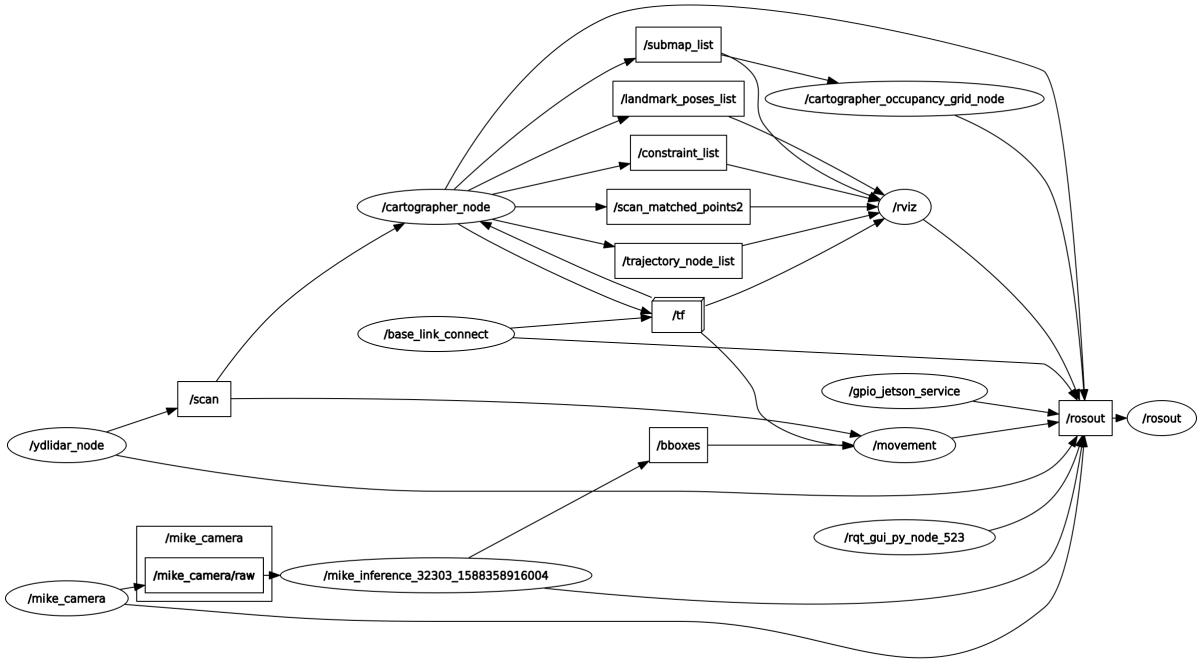


Рисунок 3.10 – Общая схема взаимодействия всех узлов и сервисов робота, сгенерированная ROS Graph.

Таким образом на выходе данного узла получается топик по имени raw, содержащий изображения. Общую схему работы данного узла можно увидеть на Рисунке 3.12.

Узел YDLIDAR

Данный узел представляет собой драйвер для YDLIDAR X4 и занимается его непосредственным запуском, остановкой, а также публикацией облака точек, формируемым лазерным сканером. Формат сообщения определён стандартом ROS sensor_msgs/LaserScan. Содержимое данного сообщения можно посмотреть в Листинге A.2. Пример получаемого изображения, создающегося из облака точек можно увидеть на Рисунке 1.2.

Таким образом на выходе данного узла получается топик с именем scan содержащий sensor_msgs/LaserScan. Общую схему работы данного узла можно увидеть на Рисунке 3.13.



Рисунок 3.11 – Пример получаемого изображения с CSI камеры Sony IMX219.

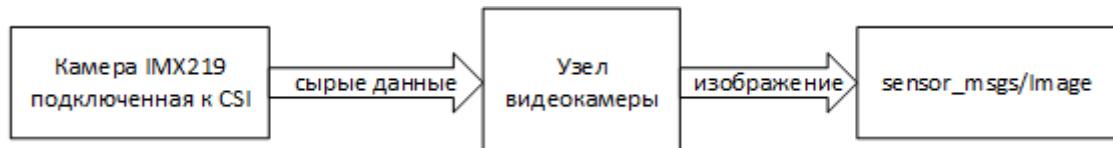


Рисунок 3.12 – Общая схема работы узла видеокамеры.



Рисунок 3.13 – Общая схема работы узла YDLIDAR.

Google Cartographer

Данный узел занимается обработкой узла scan, публикуемого узлом YDLIDAR. Основной задачей Google Cartographer является SLAM - то есть одновременная локализация и построение карты окружающей местности, для этого данной системе нужно выполнять очень много задач, а потому на выходе мы имеем сразу несколько топиков[18]:

1. `scan_matched_points`: данный топик определяется стандартом `sensor_msgs/PointCloud2` (структуру смотрите в Листинге A.3) и представляет собой облако точек в том виде, в котором оно использовалось для сопоставления сканирования с подкартами, создающимися Google Cartographer. Это облако отфильтровано и спроецировано так как это описывает конфигурационный файл Lua;
2. `submap_list`: этот топик является список всех вложенных карт, включая позу и номер последней версии каждой вложенной карты, по всем пройденным траекториям робота. Формат сообщений описан собственным стандартом `cartographer_ros_msgs/SubmapList`, его содержимое увидеть в Листинге A.4;
3. `map`: этот топик появляется только если указать это в конфигурационном файле и представляет собой цельную карту, которую сгенерировал Google Cartographer в виде двумерной матрицы. Формат этого топика определён сообщением стандарта ROS `nav_msgs/OccupancyGrid`, его содержимое увидеть в Листинге A.5.

Общую схему работы данного узла можно увидеть на Рисунке 3.14.

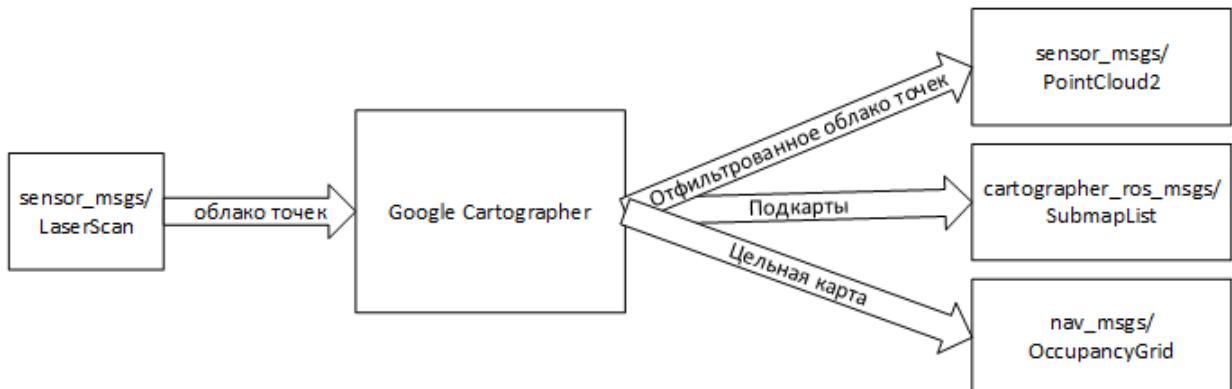


Рисунок 3.14 — Общая схема работы узла YDLIDAR.

Сервис GPIO

Данный сервис был создан с нуля на языке C++ в целях управления контроллером электродвигателей робота при помощи установленного

на Nvidia Jetson NANO разъёма стандарта GPIO, общую структуру которого можно увидеть на Рисунке 3.15.

| Jetson Nano J41 Header | | | | | |
|------------------------|----------------------------|-----|-----|-------------------------------|------------|
| Sysfs GPIO | Name | Pin | Pin | Name | Sysfs GPIO |
| | 3.3 VDC Power | 1 | 2 | 5.0 VDC Power | |
| | I2C_2_SDA I2C Bus 1 | 3 | 4 | 5.0 VDC Power | |
| | I2C_2_SCL I2C Bus 1 | 5 | 6 | GND | |
| gpio216 | AUDIO_MCLK | 7 | 8 | UART_2_TX /dev/ttyTHS1 | |
| | GND | 9 | 10 | UART_2_RX /dev/ttyTHS1 | |
| gpio50 | UART_2_RTS | 11 | 12 | I2S_4_SCLK | gpio79 |
| gpio14 | SPI_2_SCK | 13 | 14 | GND | |
| gpio194 | LCD_TE | 15 | 16 | SPI_2_CS1 | gpio232 |
| | 3.3 VDC Power | 17 | 18 | SPI_2_CS0 | gpio15 |
| gpio16 | SPI_1_MOSI | 19 | 20 | GND | |
| gpio17 | SPI_1_MISO | 21 | 22 | SPI_2_MISO | gpio13 |
| gpio18 | SPI_1_SCK | 23 | 24 | SPI_1_CS0 | gpio19 |
| | GND | 25 | 26 | SPI_1_CS1 | gpio20 |
| | I2C_1_SDA I2C Bus 0 | 27 | 28 | I2C_1_SCL I2C Bus 0 | |
| gpio149 | CAM_AF_EN | 29 | 30 | GND | |
| gpio200 | GPIO_PZ0 | 31 | 32 | LCD_BL_PWM | gpio168 |
| gpio38 | GPIO_PE6 | 33 | 34 | GND | |
| gpio76 | I2S_4_LRCK | 35 | 36 | UART_2_CTS | gpio51 |
| gpio12 | SPI_2_MOSI | 37 | 38 | I2S_4_SDIN | gpio77 |
| | GND | 39 | 40 | I2S_4_SDOUT | gpio78 |

Рисунок 3.15 – Общая структура разъёма J41 на компьютере Nvidia Jetson NANO.

На вход сервиса приходит сообщение собственного стандарта `gpio_jetson_service/gpio_srv`, состоящее команды в виде числа в формате `uint8` и выходной `bool` переменной `success` (содержимое сообщение также приведено в Листинге A.6). Действие каждой команды закреплено в

заголовочном файле commands.hpp в пространстве имён MoveCommands. Всего доступно 20 различных команд, которые являются комбинацией двух характеристик: гусеница и скорость. Дополнительно имеются команды на движение вперёд и назад⁷ с возможностью выбора скорости. Всего гусениц на роботе установлено две: левая и правая. А скоростей доступно 4 штуки:

1. Остановка (нет скорости);
2. Медленная;
3. Средняя;
4. Быстрая.

На выходе сервис возвращает своим клиенту переменную в формате boolean, значение true которой говорит об успешности подачи или снятия напряжения 3.3В на ножки разъёма GPIO или false при возникновении какой-либо ошибки.

Управление пинами GPIO происходит при помощи выполнения следующих команд в оболочке bash, вызываемых при помощи стандартной функции system("команда"):

1. echo «номер пина» > /sys/class/gpio/export;
2. echo «номер пина» > /sys/class/gpio/unexport;
3. echo «in или out» > /sys/class/gpio/gpio«номер пина»/direction;
4. echo «значение 1 или 0» > /sys/class/gpio/gpio«номер пина»/value.

Первая команда служит для того активировать данную ножку на разъёме и разрешить управление над ней. Вторая команда, соответственно, выключает данную ножку. Третья команда выполняется для назначения «направления» данной ножки. Она может быть как входной, то есть ждать какого-то управляющего сигнала, так и выходной, то есть сама подавать напряжение +3.3В. Последняя команда управляет тем значением, которое будет на ножке[19].

Также, дополнительно были сделаны тестовые клиенты к данному сервису. Первый позволяет при помощи нажатий клавиш WASD управлять направлением движения робота в ручном режиме. Второй тестовый клиент также представляет интерфейс для ручного управления роботом, но уже в полном функционале, то есть нажатие определённой клавиши на клавиатуре вызывает определённую команду GPIO сервиса.

⁷Данные команды для движения задействуют одновременно все гусеницы робота.

Общую схему работы данного сервиса можно увидеть на Рисунке 3.16.

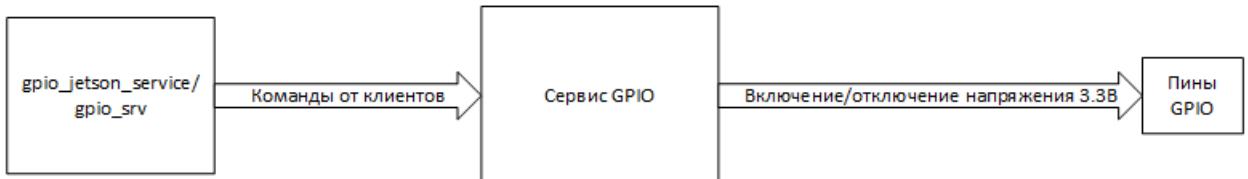


Рисунок 3.16 – Общая схема работы сервиса GPIO.

Узел записи видео

Данный узел был создан в целях сборки данных для обучения нейронной сети, которая отвечает за распознавание объектов и во время работы робота никак не используется.

Узел записи видео подписывается на топик raw, в который узел камеры публикует кадры, получаемые из подключенной CSI видеокамеры. Полученные кадры подгоняются под размер 640x480, преобразуются в формат bgr8, а затем передаются открытой библиотеке компьютерного зрения OpenCV⁸, которая настроена так чтобы записывать видеофайлы с частотой кадров 20 кадров в секунду каждые 1000 полученных кадров на подключенный к Jetson NANO по интерфейсу USB 3.0 внешний жёсткий диск в кодировке DIVX и формате avi.

Имя каждого файла уникально и состоит из базового имени (в данном случае mike-video-) и текущей даты с временем в формате «день-месяц-год-час-минута-секунда». Это позволяет не перезаписывать каждый раз одно и то же видео, а иметь сразу много кусков и не переживать за конфликт имён.

Общую схему работы данного узла можно увидеть на Рисунке 3.17.

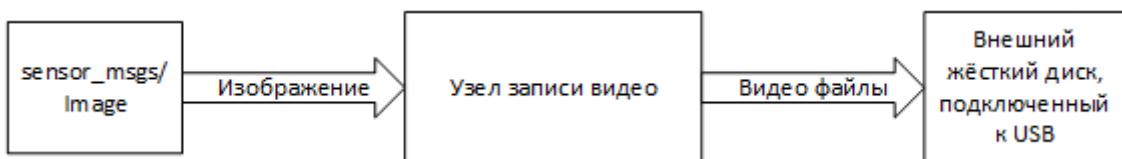


Рисунок 3.17 – Общая схема работы узла записи видео.

⁸Конкретно, в данном случае используется C++ класс cv::VideoWriter

Узел распознавания объектов

Узел распознавания объектов написан на языке Python версии 3. На входе он подписывается на топик видеокамеры raw, а на выходе предоставляет сообщения собственного стандарта inference/Bboxes (стандарт приведён в Листинге A.7).

Bboxes состоит из одного элемента - массива сообщений Bbox (стандарт сообщений Bbox приведён в Листинге A.8), который представляет собой массив так называемых bounding box. Bounding box - это по сути прямоугольник, генерируемый нейронной сетью, который указывает на распознанные объекты на входном видеоизображении.

Распознанных объектов в кадре может быть несколько, а значит и этих прямоугольников за один кадр может сгенерироваться несколько, поэтому важно передавать именно массив bounding box. В определённом в данном узле собственном стандарте сообщения inference/Bbox у каждого bounding box'а имеются следующие значения:

1. x_min - первая координата прямоугольника по оси *x* в формате float32;
2. y_min - первая координата прямоугольника по оси *y* в формате float32;
3. x_max - вторая координата прямоугольника по оси *x* в формате float32;
4. y_max - вторая координата прямоугольника по оси *y* в формате float32;
5. score - вероятность в формате float32 того, что распознанный объект распознан верно;
6. label в строковом формате string - название распознанного объекта ⁹.

Для распознавания объектов используется нейронная сеть, запускаемая на видеоядре компьютера Nvidia Jetson NANO, основанная на ssd_inception_v2_coco_2017_11_17, обученная на собранном с видеокамеры робота датасете, а также прошедшая оптимизация при помощи ПО от компании Nvidia - TensorRT. Данная нейронная сеть создавалась не в рамках

⁹по нему можно понять какого вида объект был обнаружен и понять является ли он целевым

работы над данной ВКР, поэтому подробности её создания не будут освещены в тексте данной работы. Из распознаваемых данной нейронной сетью объектов можно выделить:

- Прозрачная бутылка;
- Кухонный нож с белой ручкой;
- Пластиковый контейнер лапши быстрого приготовления;
- Глубокая фарфоровая тарелка;
- Фонарик с металлическим корпусом;
- Синяя шариковая авторучка.

Общую схему работы данного узла можно увидеть на Рисунке 3.18.

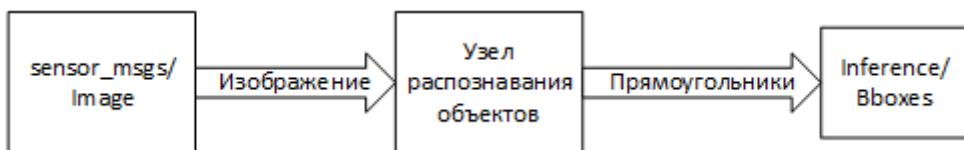


Рисунок 3.18 – Общая схема работы узла распознавания объектов.

Узел движения

Данный узел является самым главным узлом в данной работе и был также написан с нуля. Его задача принимать решения о том, куда поедет робот на основании тех данных, которые приходят из нейронной сети, лазерного сканера YDLIDAR, и Google Cartographer.

Узел движения подписывается на 2 топика: /scan - который содержит облако точек, выдаваемых лидаром, и /bboxes - топик, в который попадают bounding box из нейросети. Также, узел включает в себя слушателя изменений местоположения робота по имени «base_link». Именно по этому имени можно получить текущее местоположение робота на карте, создающейся Google Cartographer. Помимо этого узел является клиентом сервиса управления двигателями для того чтобы иметь возможность непосредственно отдавать команды для различных манёвров робота.

Обработка облака точек. Приходящее в узел облако точек из 720 элементов, как и было описано в пункте 3.3.1, делится на 4 части, по каждой

из которой считается среднее арифметическое число: перед f , зад b , лево l и право r .

В первую очередь анализируется передняя часть. Если на ней было обнаружено хоть одно число больше 0 и менее 0,3, то считается, что перед роботом есть препятствие и нужно поворачивать. Поворот в нужную сторону длится 1 секунду, затем алгоритм повторяется.

Обработка застреваний Для обработки застреваний устанавливается слушатель так называемого transform. Узел слушает два transform'a: «base_link» и «тар», которые генерируются Google Cartographer. В первом содержится информация об отклонении местоположения и вектора поворота от второго transform. Без второго transform нельзя было бы понять местоположение робота, так как не было бы «базового» местоположения (объекта transform) с которым и происходит сравнение.

Каждый раз программа запоминает последнее местоположение робота и момент времени в котором данное местоположение было запомнено. Как только проходит одна секунда, проверяется насколько сильно робот изменил местоположение. Если робот не «застрял», то запоминается новое время и местоположение робота.

Если местоположение изменилось недостаточно сильно ($dx < 0,3$, $dy < 0,3$ и отклонение вектора поворота $dr < 3$ градуса), то происходит перехват управления: робот останавливается, движется назад в течении секунды, затем ищется в какую сторону повернуть по высчитанным ранее средним арифметическим числам. Робот поворачивает в течении секунды, затем движение продолжается как обычно.

Следование за целевым объектом. При появлении какого-либо распознанного объекта в кадре, нейронная сеть посылает в топик /bboxes сообщение с координатами прямоугольника, в рамках которого и находится распознанный объект. Если распознанных объектов больше чем 1, то для дальнейшего анализа выбирается тот, у кого более высокая вероятность правильного совпадения имени распознанного объекта с действительностью. Это делается для того чтобы отфильтровать те различные мелкие фрагменты объекта, которые ввиду неточности работы нейронной сети появляются на распознанном объекте.

Следующим шагом проверяется название объекта и если оно есть в списке целевых объектов, то фактически происходит перехват управления. Робот останавливается и центр прямоугольника распознанного объекта выравнивается в кадре и становится стабильнее. После этого шасси роботаоворачивается таким образом, чтобы центр прямоугольника, в котором находится целевой объект оказался в середине (с небольшой допустимой погрешностью) кадра видеокамеры¹⁰. После этого робот начинает движение вперёд ровно до тех пор, пока какой-либо из краёв прямоугольника не достигнет края кадра. Если роботу удалось достичь данной точки, то движение останавливается, так как считается, что робот выполнил свою задачу, найдя целевой объект на местности.

Схема всего алгоритма работы узла движения представлена на Рисунке 3.19. Общую схему работы данного узла можно увидеть на Рисунке 3.20. Исходный код данного узла на языке программирования C++ приведён в Листинге Б.1.

¹⁰Для этих целей в исходном коде программы заранее записано разрешение видеокадра, с которым оперирует нейронная сеть.

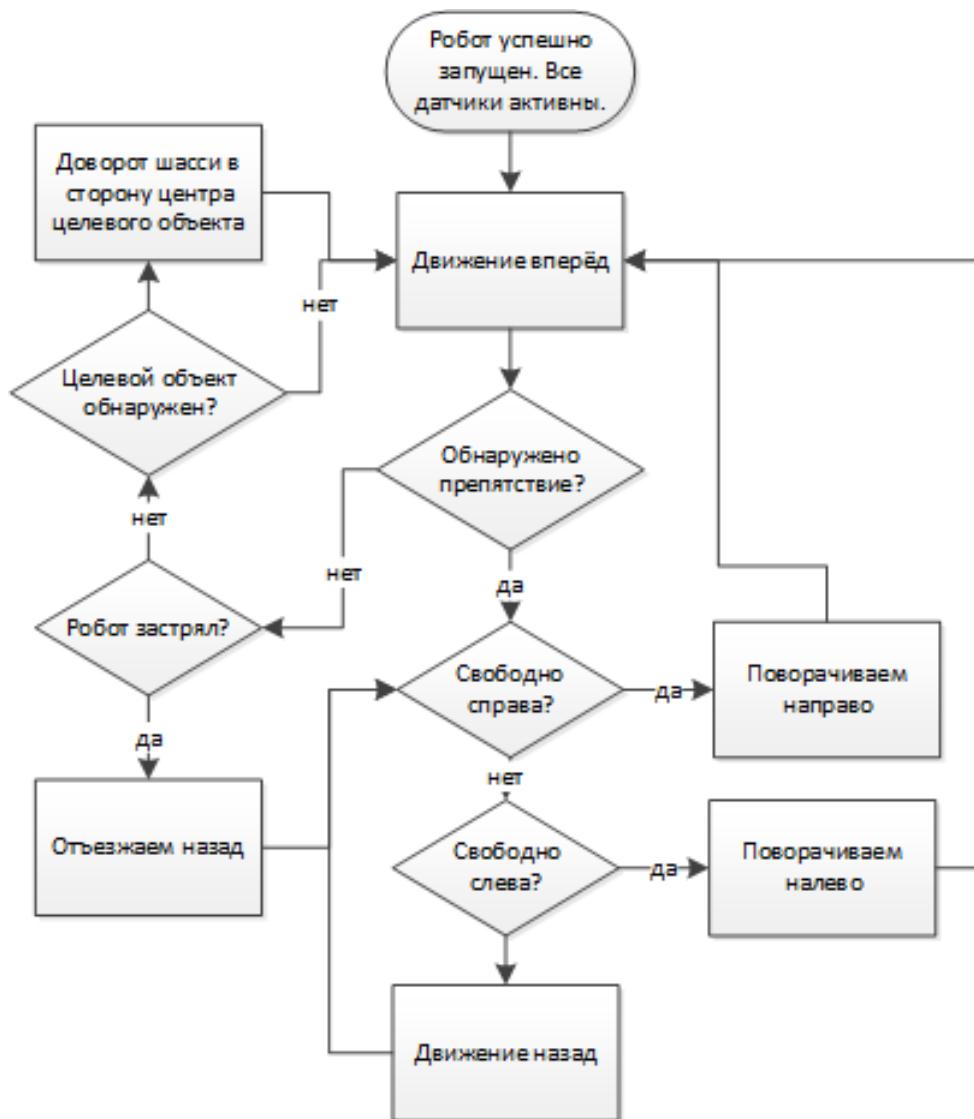


Рисунок 3.19 – Общая схема работы алгоритма движения робота.

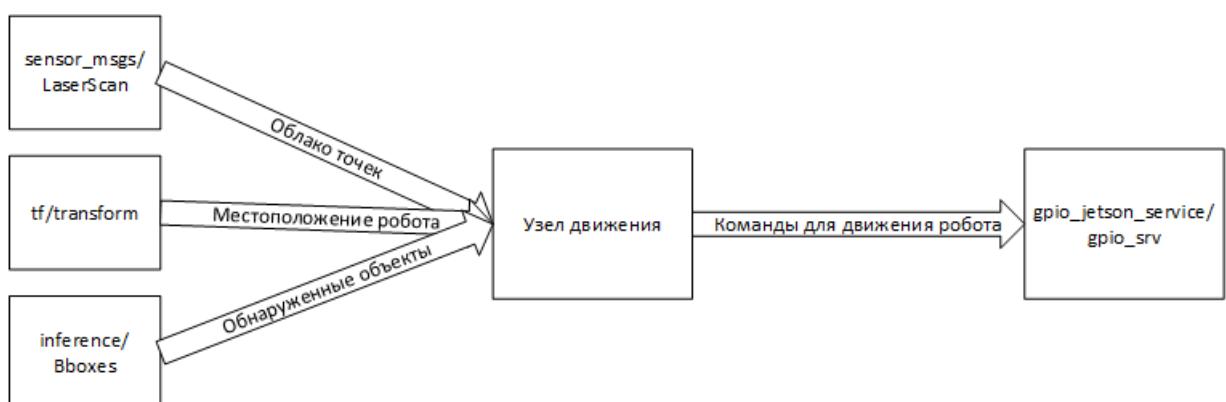


Рисунок 3.20 – Общая схема работы узла движения.

Заключение

Основные результаты работы заключаются в следующем.

1. Цели и задачи поставленные в данной ВКР были успешно выполнены;
2. На основе анализа предметной области был реализован и построен целый программно-аппаратный комплекс, выполняющий свою задачу;
3. Тестирования показали, что робот, в большинстве случаев справляется со своей задачей нахождения целевых объектов;
4. Моделирование различных ситуаций показало, что робот справится далеко не с каждым случаем, в котором он может оказаться (например, на широкой улице). Поэтому для конкретных случаев скорее всего потребуется дополнительное совершенствование и корректировка текущей реализации.

Таким образом была разработана система управления и формирования поведенческой стратегии автономного мобильного робота на основе визуального анализа окружающего пространства. Изображения готового робота можно увидеть на Рисунке 3.21.



Рисунок 3.21 — Готовый робот, выполняющий поиск целевых объектов в доме.

Список литературы

1. *Сергеев, Е.* Стратегия новой индустриализации России: автоматизация, роботизация, нанотехнологии [Текст] / Е. Сергеев. — ЛитРес, 2018. — 200 с. — Текст: непосредственный.
2. *Лорье, Ж.-Л.* Системы искусственного интеллекта: Пер. с франц. [Текст] / Ж.-Л. Лорье. — Мир, 1991. — 568 с. — Текст: непосредственный.
3. *Aycock, T.* A Simultaneous Localization and Mapping Implementation Using Inexpensive Hardware / T. Aycock, U. of Alabama. Department of Electrical, C. Engineering. — Текст: электронный. — 2010. — URL: https://ir.ua.edu/bitstream/handle/123456789/885/file_1.pdf?sequence=1&isAllowed=y (дата обр. 20.05.2020).
4. *Xiao, J.* Proceedings of the 2018 International Symposium on Experimental Robotics [Текст] / J. Xiao, O. Khatib, T. Kroger. — Springer International Publishing, 2020. — 804 с. — Текст: непосредственный. — (Springer Proceedings in Advanced Robotics Series).
5. *Hutter, M.* Field and Service Robotics: Results of the 11th International Conference [Текст] / M. Hutter, R. Siegwart. — Springer International Publishing, 2017. — 715 с. — Текст: непосредственный. — (Springer Proceedings in Advanced Robotics).
6. *Jia, Y.* Proceedings of 2019 Chinese Intelligent Systems Conference: Volume I [Текст] / Y. Jia, J. Du, W. Zhang. — Springer Singapore, 2019. — 771 с. — Текст: непосредственный. — (Lecture Notes in Electrical Engineering).
7. *Mather, P.* Computer Processing of Remotely-Sensed Images: An Introduction [Текст] / P. Mather. — Wiley, 2005. — 442 с. — Текст: непосредственный.
8. *Савинов, В.* Kinect — продвинутый датчик для роботов / В. Савинов. — Текст: электронный // Хабр : [сайт]. — 2010. — 27 нояб. — URL: <https://habr.com/ru/post/108927/> (дата обр. 20.05.2020).

9. Технические спецификации NVIDIA Jetson Nano. — Текст: электронный // NVIDIA : [сайт]. — 2020. — URL: <https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-nano/> (дата обр. 20.05.2020).
10. NVIDIA Kaya. — Текст: электронный // NVIDIA : [сайт]. — 2020. — URL: https://docs.nvidia.com/isaac/isaac/doc/tutorials/assemble_kaya.html (дата обр. 20.05.2020).
11. Running Isaac SDK on Kaya. — Текст: электронный // NVIDIA : [сайт]. — 2020. — URL: https://docs.nvidia.com/isaac/isaac/doc/tutorials/kaya_software.html (дата обр. 20.05.2020).
12. JetBot AI Kit, AI Robot Based on Jetson Nano. — Текст: электронный // Waveshare : [сайт]. — 2020. — URL: <https://www.waveshare.com/product/jetbot-ai-kit.htm> (дата обр. 20.05.2020).
13. JetBot examples. — Текст: электронный // GitHub : [сайт]. — 2020. — URL: <https://github.com/NVIDIA-AI-IOT/jetbot/wiki/examples> (дата обр. 20.05.2020).
14. Types of Robot Sensors. — Текст: электронный // robot platform : [сайт]. — 2020. — URL: http://robotplatform.com/knowledge/sensors/types_of_robot_sensors.html (дата обр. 20.05.2020).
15. Real-Time Loop Closure in 2D LIDAR SLAM / W. Hess [и др.] // 2016 IEEE International Conference on Robotics and Automation (ICRA). — Текст: электронный. — 2016. — С. 1271–1278. — URL: <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45466.pdf> (дата обр. 20.05.2020).
16. Joseph, L. Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System, 2nd Edition [Текст] / L. Joseph, J. Cacace. — Packt Publishing, 2018. — 580 с. — Текст: непосредственный.
17. ROS Index Statistics. — Текст: электронный // ROS Index : [сайт]. — 2020. — URL: <https://index.ros.org/stats/> (дата обр. 20.05.2020).
18. Cartographer. — Текст: электронный // Read the Docs : [сайт]. — 2020. — URL: <https://google-cartographer.readthedocs.io/en/latest/> (дата обр. 20.05.2020).

19. *kangalow.* Jetson Nano GPIO / kangalow. — Текст: электронный // JetsonHacks : [сайт]. — 2020. — URL: <https://www.jetsonhacks.com/2019/06/07/jetson-nano-gpio/> (дата обр. 20.05.2020).

Список рисунков

| | |
|--|-------------------------------------|
| 1.1 Пример картинки, генерируемой 3D лидаром, представленном на выставке CEATEC 2017 компанией Panasonic в Японии. | 8 |
| 1.2 Пример картинки, генерируемой 2D лидаром YDLIDAR X4. | 9 |
| | |
| 2.1 Игровой сенсор Xbox Kinect, представленный в 2009 году в рамках выставки электронных развлечений E3. | 11 |
| 2.2 NVIDIA Jetson Nano - компактный и мощный одноплатный компьютер, представленный в 2019 году. | 13 |
| 2.3 Внешний вид робота NVIDIA Kaya. | 14 |
| 2.4 Набор инструментов NVIDIA JetBot от Waveshare. | 15 |
| | |
| 3.1 Шасси TS100 для самодельного робота. | 18 |
| 3.2 Шасси робота без установленного на него оборудования. | 19 |
| 3.3 Общая структура желаемой модели контроллера. | 19 |
| 3.4 Готовый экземпляр контроллера двигателей. | 20 |
| 3.6 Общая схема алгоритма обезьяда препятствий. | 22 |
| 3.7 Пример карты, сгенерированной Google Cartographer. | 24 |
| 3.8 Общая структура Robot Operating System. | 26 |
| 3.9 Глобальный репозиторий ROS Index и версии ROS. | 27 |
| 3.10 Общая схема взаимодействия всех узлов и сервисов робота, сгенерированная ROS Graph. | 29 |
| 3.11 Пример получаемого изображения с CSI камеры Sony IMX219. . | 30 |
| 3.12 Общая схема работы узла видеокамеры. | 30 |
| 3.13 Общая схема работы узла YDLIDAR. | 30 |
| 3.14 Общая схема работы узла YDLIDAR. | 31 |
| 3.15 Общая структура разъёма J41 на компьютере Nvidia Jetson NANO. . | 32 |
| 3.16 Общая схема работы сервиса GPIO. | 34 |
| 3.17 Общая схема работы узла записи видео. | 34 |
| 3.18 Общая схема работы узла распознавания объектов. | 36 |
| 3.19 Общая схема работы алгоритма движения робота. | 39 |
| 3.20 Общая схема работы узла движения. | 39 |
| 3.21 Готовый робот, выполняющий поиск целевых объектов в доме. . | 40 |

Приложение А

Форматы сообщений ROS, используемые в ВКР

Листинг А.1: sensors_msg/Image.msg

```
std_msgs/Header header
5      uint32 height
      uint32 width
      string encoding
      uint8 is_bigendian
      uint32 step
      uint8[] data
```

Листинг А.2: sensors_msg/LaserScan.msg

```
std_msgs/Header header
5      float32 angle_min
      float32 angle_max
      float32 angle_increment
      float32 time_increment
      float32 scan_time
      float32 range_min
      float32 range_max
      float32[] ranges
10     float32[] intensities
```

Листинг А.3: sensors_msg/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

```

10

Листинг А.4: cartographer_ros_msgs/SubmapList.msg

```

std_msgs/Header header
cartographer_ros_msgs/SubmapEntry[] submap

```

Листинг А.5: nav_msgs/OccupancyGrid.msg

```

std_msgs/Header header
nav_msgs/MapMetaData info
int8[] data

```

Листинг А.6: gpio_jetson_service/gpio_srv.srv

```

uint8 command
---
bool success

```

Листинг А.7: inference/Bboxes.msg

```
Bbox[] bboxes
```

Листинг A.8: inference/Bbox.msg

```
5   float32 x_min
    float32 y_min
    float32 x_max
    float32 y_max
    float32 score
    string label
```

Приложение Б

Реализованный исходный код

Листинг Б.1: Исходный код узла движения робота

```

#include "ros/ros.h"
#include "../../gpio_jetson_service/include/
    gpio_jetson_service/commands.hpp"
#include "pcl_ros/point_cloud.h"
#include "gpio_jetson_service/gpio_srv.h"
5 #include <sensor_msgs/LaserScan.h>
#include <vector>
#include "tf/transform_listener.h"
#include "inference/Bboxes.h"

10 #define IMAGE_WIDTH 300
#define IMAGE_HEIGHT 300

bool backward, left, forward, right;
float backward_m = 0, left_m = 0, forward_m = 0, right_m = 0;
15 double x = 0, y = 0, r = 0;
double transform_time_sec;
ros::ServiceClient gpio_client;
tf::TransformListener* transformListener;
tf::StampedTransform transform_bot;
20 float image_middle_x, image_middle_y;

void gpio_command(const uint8_t command) {
    gpio_jetson_service::gpio_srv service;
    service.request.command = MoveCommands::FULL_STOP;
25    gpio_client.call(service);
    gpio_jetson_service::gpio_srv service2;
    service2.request.command = command;
    gpio_client.call(service2);
}

30 void inferenceCallback(const inference::BboxesConstPtr &bboxes
) {
    if(bboxes->bboxes.empty()) {
        return;
    }
}

```

```

35     ROS_WARN("Bboxes got! Size: %lu", bboxes->bboxes.size());
inference::Bbox bbox;
40     if (bboxes->bboxes.size() > 1) {
        float max_score = 0;
        unsigned long max_score_index = 0;
        for (unsigned long i = 0; i < bboxes->bboxes.size(); i
++) {
            ROS_WARN("Object %s with score %f.", bboxes->
bboxes[i].label.c_str(), bboxes->bboxes[i].score);
            if (max_score < bboxes->bboxes[i].score) {
                if (bboxes->bboxes[i].label != "tvmonitor")
continue;
45            max_score = bboxes->bboxes[i].score;
            max_score_index = i;
        }
        bbox = bboxes->bboxes[max_score_index];
50    } else bbox = bboxes->bboxes[0];
    if (bbox.label != "tvmonitor") return;
55    ROS_WARN("Selected object %s with score %f and (%f,%f,%f,%
f).", bbox.label.c_str(), bbox.score, bbox.x_min, bbox.
y_min, bbox.x_max, bbox.y_max);
    float x1 = bbox.x_min;
    float y1 = bbox.y_min;
    float x2 = bbox.x_max;
    float y2 = bbox.y_max;
60    float object_center_x = (x1 + x2) / 2;
    float object_center_y = (y1 + y2) / 2;
    ROS_WARN("Object center (%f,%f).", object_center_x,
object_center_y);
65    if (object_center_x < image_middle_x - 10) {
        ROS_WARN("Follow left to the object... ");
        gpio_command(MoveCommands::RIGHT_FORWARD_LOW);
        usleep(100000);
    }
70    if (object_center_x > image_middle_x + 10) {

```

```

        ROS_WARN("Follow right to the object...");
        gpio_command(MoveCommands::LEFT_FORWARD_LOW);
        usleep(100000);
75    }
    gpio_command(MoveCommands::FULL_STOP);
}

void ydLidarPointsCallback(const sensor_msgs::
    LaserScanConstPtr& message) {
    float backward_lm = 0, left_lm = 0, forward_lm = 0,
right_lm = 0;
    for (int i = 0; i < 719; ++i) {
        /*if (message->ranges[i] > 1) {
            ROS_WARN("Range on %d > 1 !!! and equals %f", i,
message->ranges[i]);
        }*/
85    if (i > 270 && i < 450) {
        backward_lm += message->ranges[i] > 0 ? message->
ranges[i] : 1;
    } else
        if (i > 90 && i < 270) {
            left_lm += message->ranges[i] > 0 ? message->
ranges[i] : 1;
        } else
            if (i > 630 || i < 90) {
                forward_lm += message->ranges[i] > 0 ? message->
ranges[i] : 1;
            } else
                if (i > 450 && i < 630) {
                    right_lm += message->ranges[i] > 0 ? message->
ranges[i] : 1;
                }
95    }
    backward_m = backward_lm / 180;
    left_m = left_lm / 180;
    forward_m = forward_lm / 180;
    right_m = right_lm / 180;
    /*if (message->ranges[360] > 0 && message->ranges[360] <
0.2f) {
100   }*/
    for (int i = 0; i < 720; i++) {
        left = right = backward = forward = false;
105
}

```

```

    if (message->ranges[i] > 0 && message->ranges[i] < 0.3
f) {
    if (i > 270 && i < 450) {
        ROS_WARN("Backward obstacle");
        backward = true;
        return;
    } else
    if (i > 90 && i < 270) {
        ROS_WARN("Left obstacle");
        left = true;
        return;
    } else
    if (i > 630 || i < 90) {
        ROS_WARN("Forward obstacle");
        forward = true;
        return;
    } else
    if (i > 450 && i < 630) {
        ROS_WARN("Right obstacle");
        right = true;
        return;
    }
}
}

void movement() {
    if (forward) {
        int min = left_m >= right_m ? 0 : 1;
        switch (min) {
            case 0:
                ROS_WARN("Going to the left side");
                gpio_command(MoveCommands::
RIGHT_FORWARD_MIDDLE);
                sleep(1);
                gpio_command(MoveCommands::FORWARD_LOW);
                break;
            case 1:
                ROS_WARN("Going to the right side");
                gpio_command(MoveCommands::LEFT_FORWARD_MIDDLE
);
                sleep(1);
                gpio_command(MoveCommands::FORWARD_LOW);
}
}
}

```

```

        break;
    default:
        ROS_ERROR("Case doesn't exist!");
    }
150 } else {
    gpio_command(MoveCommands::FORWARD_LOW);
}
}

155 void stuck_detect() {
    ros::Time now = ros::Time::now();
    if (now.toSec() - transform_time_sec < 1) {
        return;
    }
160 try {
    transformListener->waitForTransform("base_link", "map",
, ros::Time(0), ros::Duration(1.0));
    transformListener->lookupTransform("base_link", "map",
ros::Time(0), transform_bot);
} catch (tf2::LookupException exception) {
    ROS_ERROR("error: %s", exception.what());
    return;
}

165 double bot_x = transform_bot.getOrigin().x();
double bot_y = transform_bot.getOrigin().y();
170 double roll, pitch, yaw;
transform_bot.getBasis().getRPY(roll, pitch, yaw);

double bot_dir = yaw * 180.0 / M_PI;
175 double dX = std::abs(bot_x - x);
double dY = std::abs(bot_y - y);
double dR = std::abs(bot_dir - r);

180 if (dX < 0.03 && dY < 0.03 && dR < 3.0) {
    ROS_WARN("Stuck detected!!!!");
    gpio_command(MoveCommands::BACKWARD_FAST);
    sleep(1);
}
185

```

```

    ROS_WARN("dX = %f dY = %f dR = %f", bot_x - x, bot_y - y,
bot_dir - r);

    x = bot_x;
    y = bot_y;
190   r = bot_dir;
    transform_time_sec = ros::Time::now().toSec();
}

int main(int argc, char **argv) {
195   ros::init(argc, argv, "movement");
    image_middle_x = IMAGE_WIDTH / 2.0;
    image_middle_y = IMAGE_HEIGHT / 2.0;
    ros::NodeHandle nodeHandle;
    sleep(5);
200   transform_time_sec = ros::Time::now().toSec();
    transformListener = new tf::TransformListener(nodeHandle);
    ros::Subscriber ydLidarPointsSub =
        nodeHandle.subscribe<sensor_msgs::LaserScan>("/scan",
205   1000, ydLidarPointsCallback);
    ros::Subscriber inferenceSub =
        nodeHandle.subscribe<inference::Bboxes>("/bboxes",
1, inferenceCallback);
    gpio_client = nodeHandle.serviceClient<gpio_jetson_service
:gpio_srv>("gpio_jetson_service");
    gpio_jetson_service::gpio_srv service;
    service.request.command = MoveCommands::FULL_STOP;
    gpio_client.call(service);
210   while (ros::ok()) {
        movement();
        stuck_detect();
        ROS_INFO("Forward: %f, Left: %f, Right: %f, Backward:
215   %f", forward_m, left_m, right_m, backward_m);
        ros::spinOnce();
    }
    gpio_command(MoveCommands::FULL_STOP);
    return EXIT_SUCCESS;
}

```