МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество «Казахский национальный исследовательский технический университет имени К.И.Сатпаева»

Институт Автоматики и информационных технологий

Кафедра Робототехники и технических средств автоматики

Сабитов Темирлан Рустамович

Разработка системы сканирования пространства для мобильного робота

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

6В07111 – Робототехника и мехатроника

Алматы 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество «Казахский национальный исследовательский технический университет имени К.И.Сатпаева»

SATBAYEV UNIVERSITY

Институт Автоматики и информационных технологий

Кафедра Робототехники и технических средств автоматики

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой РТиТСА
кандидат технических наук,
ассоциированный профессор
_____Ожикенов К. А.
«_7_» _июня_ 2024 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

На тему: «Разработка системы сканирования пространства для мобильного робота»
6B07111 – Робототехника и мехатроника

Выполнил Сабитов Темирлан Рустамович

Рецензент
Проректор по НиС ALT Университет имени Мухамеджана Тынышпаева, PhD, ассоциированный профессор
_____ Сергазин Г. К.
«_22_» май 2024 г.

Научный руководитель
Кандидат физико-математических наук, ассоциированный профессор
_____ Бактыбаев М. К.
«_30_» май 2024 г.

Алматы 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество «Казахский национальный исследовательский технический университет имени К.И.Сатпаева»

SATBAYEV UNIVERSITY

Институт Автоматики и информационных технологий

Кафедра Робототехники и технических средств автоматики

6В07111 – Робототехника и мехатроника

УТВЕРЖДАЮ
Заведующий кафедрой РТиТСА
кандидат технических наук,
ассоциированный профессор
_____ Ожикенов К. А.
«4» июня 2024 г.

ЗАДАНИЕ
на выполнение дипломного проекта

Обучающегося Сабитова Темирлана Рустамовича
Тема: Разработка системы сканирования пространства для мобильного робота

Утверждена приказом Ректора Университета №548 от "04" января 2024 г.Срок сдачи законченной работы «3 06 2024 г.

Исходные данные к дипломному проекту: Visual Studio Code, Перечень подлежащих разработке в дипломном проекте вопросов: а) построили алгоритм работы программного обеспечения;
б) разработали принципиальную схему «Мобильного робота с системой сканирования пространства»;
в)отстроили 3Д-модель «Мобильного робота с системой сканирования пространства» в программном

обеспечении SolidWorks.

Перечень графического материала (с точным указанием обязательныхчертежей):

Представлены 20 *слайдов презентации работы*

Рекомендуемая основная литература: *из* 47 *наименований* 47

ГРАФИК

подготовки дипломной работы (проекта)

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечания
3D модель	16.01 - 12.02.2024г	Выполнено
Печать корпуса	20.03 - 17.04.2024г	Выполнено
Сборка	17.04 - 25.05.2024г	Выполнено.
Програмная часть	27.05 - 29.05.2024г	Выполнено.

Подписи

консультантов и норм контролера на законченную дипломную работу (проект)с указанием относящихся к ним разделов работы (проекта)

Наименование разделов	Консультанты, И.О.Ф. (уч. степень, звание)	Дата подписания	Подпись
Норм контролер	Ингибай Е. А	04.06.2024	

Научный руководитель: _____ Бактыбаев М. К.

Задание принял к исполнению обучающийся: _____ Сабитов Т. Р.

Дата «4» июня 2024

АҢДАТПА

Бұл дипломдық жұмыстың мақсаты мобильді робот үшін кеңістікті сканерлеу жүйесін әзірлеу болып табылады. Жоба жұмыс кеңістігінің картасын құру, рельефті бағдарлау, роботты қашықтан басқару және автономды қозғалыс сияқты робототехниканың қолданбалы мәселелерін шешеді.

Теориялық бөлімде ROS жүйесі қарастырылған. Жүйенің негізгі ұғымдары мен негізгі элементтері келтірілген. Жоғарыда аталған мәселелерді шешуге арналған ROS құралдары, сондай-ақ оларды қолданудың өзектілігі қарастырылады. Gazebo ортасында осы құралдарды қолданатын роботты модельдеу көрсетіледі.

Практикалық бөлімде кеңістікті сканерлеу жүйесі бар мобильді роботтың прототипі сипатталған. Тапсырмаларды орындау үшін қажетті бағдарламалық кодтардың үзінділері, пайдаланылған компоненттердің электр тізбектері, сондай-ақ жүргізілген эксперименттердің нәтижелері келтірілген.

Қорытынды бөлім қаржылық және математикалық есептеулерге арналған. Онда қолданылған құрылғыларды калибрлеу және конфигурациялау үшін қолданылатын математикалық модельдер сипатталған.

Жобаның негізгі идеясы-ROS жүйесінің негізгі функционалдығын қолдайтын бюджеттік мобильді робот құру. Бұл жүйенің пайдалы ерекшелігі-барлық бағдарламалық код физикалық сипаттамаларына қарамастан кез-келген басқа мобильді платформаға бейімделуі мүмкін. Сондықтан, бұл жұмыстың нәтижелері білім беру және практикалық мақсаттарда пайдалы болады деп үміттенемін.

ABSTRACT

The goal of this diploma project is to develop a space scanning system for a mobile robot. The project addresses practical robotics tasks such as workspace mapping, localization, remote robot control, and autonomous movement.

In the theoretical part, the ROS (Robot Operating System) is reviewed. Key concepts and fundamental elements of the system are presented. The ROS tools for solving the problems are discussed, along with their practical relevance. A robot simulation using these tools in the Gazebo environment is demonstrated.

The practical section describes a prototype of a mobile robot equipped with a space scanning system. It includes snippets of code necessary to accomplish the specified tasks, electrical schematics of the components used, and the results of conducted experiments.

The final part focuses on calculations, both financial and mathematical. Mathematical models used for calibrating and tuning the employed devices are described.

The main idea of the project is to create an affordable mobile robot that supports the core functionality of ROS. A valuable feature of this system is that all the software code can be adapted to any other mobile platform, regardless of its physical characteristics. Therefore, I hope that the results of this work will be useful for educational and practical purposes.

АННОТАЦИЯ

Целью настоящей дипломной работы является разработка системы сканирования пространства для мобильного робота. Проект решает прикладные задачи робототехники, такие как построение карты рабочего пространства, ориентация в местности, удаленное управление роботом и автономное движение.

В теоретической части обозревается система ROS. Приводятся ключевые понятия и основные элементы системы. Рассматривается инструменты ROS для решения вышеупомянутых проблем, а также актуальность их применения. Демонстрируется симуляция робота, использующего данные инструменты, в среде Gazebo.

В практической части описывается прототип мобильного робота с системой сканирования пространства. Приводятся фрагменты программных кодов, необходимых для выполнения поставленных задач, электрические схемы использованных компонентов, а также результаты проведенных экспериментов.

Завершительная часть посвящена расчетам, как финансовым, так и математическим. В ней описаны математические модели, используемые для калибровки и настройки использованных устройств.

Основная идея проекта заключается в создании бюджетного мобильного робота, поддерживающего основной функционал системы ROS. Полезной особенностью данной системы является то, что весь программный код может быть адаптирован под любую другую мобильную платформу, независимо от ее физических характеристик. Поэтому я надеюсь, что результаты данной работы окажутся полезными как в образовательных, так и в практических целях.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Система сканирования пространства мобильного робота необходима для выполнения прикладных задач робототехники, таких как построение карты, навигация в местности, построение пути и автономное движение.

Целью данной дипломной работы является разработка системы сканирования пространства мобильного робота, базирующейся на таких датчиках, как LiDAR и камеры. В качестве платформы, связывающей сенсоры и мобильную платформу, выступает ROS.

ROS (операционная система роботов) – среда с открытым исходным кодом [1], совокупность программ для решения широкого круга задач робототехники, начиная с получения данных сенсоров, заканчивая управлением группой роботов.

ROS активно используется в различных областях, таких как научные исследования и образование [2], промышленная, обслуживающая робототехника [3]. Он является мощным инструментом, так как поддерживает все типы роботов, от мобильных платформ до антропоморфных систем.

На данный момент существуют несколько версий ROS - ROS и ROS2. Новых подверсий ROS выпускаться больше не будет, хотя последняя из них на момент 2024 года продолжает поддерживаться сообществом. Финальным дистрибутивом ROS является ROS Noetic, официально выпущенный 23 Мая 2020 года.

ROS объединяет сообщество робототехников со всей планеты, так как код доступен в любой точке мира, где проведен интернет, и каждый может внести свой вклад, поделившись разработками в GitHub.

Официальная документация, обучающие материалы и ознакомительные статьи по данной системе ведутся на соответствующих веб страницах, а программные коды ежегодно обновляются, что объясняет скромное наличие литературы на эту тематику.

Несмотря на частые обновления, концепт системы ROS и его основные элементы остаются постоянными, они и будут рассмотрены в следующем разделе.

1. Теоретическая часть

1.1. Основные элементы ROS

Любая программа, которая запускается с ROS, именуется узлом [4]. Узлы обычно пишутся на c++ или python. Они могут быть связаны между собой различными методами, такими как темы (topic) и сервисы (service). Информация, которую программы передают друг другу, называется сообщением (message)

Тема [5] – канал связи, по которому один узел может принять или передать сообщение. При этом узел, посылающий сообщения называется издатель (publisher), принимающий – подписчик (subscriber). Через эти каналы можно передать данные датчиков, команды для приводов, информацию о строении робота и т. д.

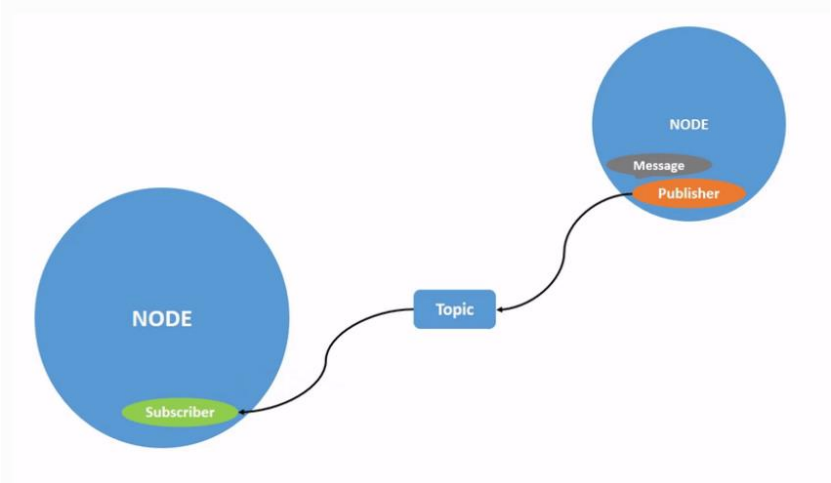


Рисунок 1.1 – Схема взаимодействия узлов через топики.

Сервис [6] – механизм взаимодействия двух узлов, при котором один из них посылает сообщение (request), делая запрос, а другой отправляет ответ (response). При этом первый называется клиентом, второй- сервером. Сервисы используются для разовых операций, например калибровки датчиков, в свою очередь топики предназначены для непрерывной передачи данных.

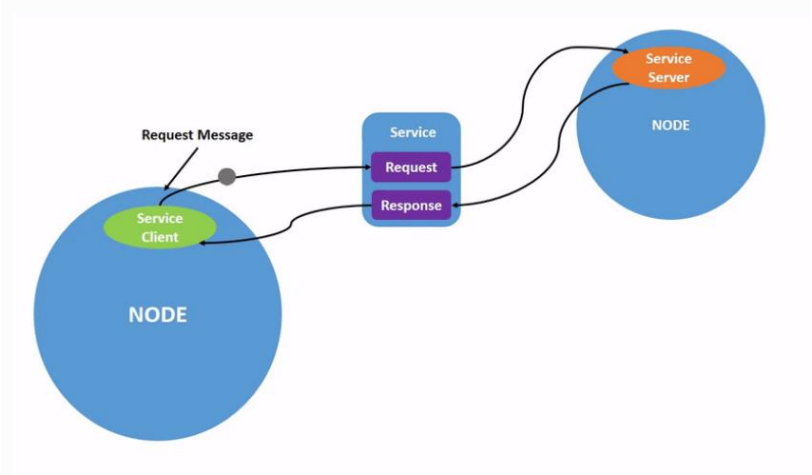


Рисунок 1.2 – Схема взаимодействия узлов через сервисы.

Сообщение [7] – информация, которую программы передают друг другу. Существует множество видов сообщений для различных типов данных, как простейших, вроде string, int, float, так и составных, как cmd_vel или imu. В случае необходимости пользователь может создать собственный тип сообщений.

Параметры [8] – переменные узлов, хранящие определенное значение, влияющие на результат работы программ. Например, параметр включения графического интерфейса – логическая переменная.

Узлы хранятся в пакетах [9]. Пакет – папка, содержащая исходный код, необходимые материалы для его компиляции, дополнительные файлы, содержащие информацию о сообщениях, симуляциях, инструкциях запуска и др.

Пакеты хранятся в рабочем пространстве [10]. Рабочее пространство – папка, содержащая папки логически связанных пакетов. Компиляция кода происходит при вводе соответствующей команды в терминале, внутри директории рабочего пространства.

Совместно с ROS в данной работе используются программы Rviz (Robot Visualization) и Gazebo. RViz [11] – программа для визуализации модели робота и данных датчиков. Gazebo [12] – среда для создания симуляций роботов.

Рисунок 1.3 – Логотипы Rviz и Gazebo

За управление исполнительными устройствами отвечает ROS_Control – концепт [13], разделяющий контроль на высший и низший уровни (high и low level control соответственно). Низшим уровнем является микроконтроллер, подсоединенный к двигателям и сенсорам, высшим – микрокомпьютер с ROS, отправляющий команды и принимающий данные с микроконтроллера. Уровни контроля связаны между собой интерфейсами, как показано на диаграмме.

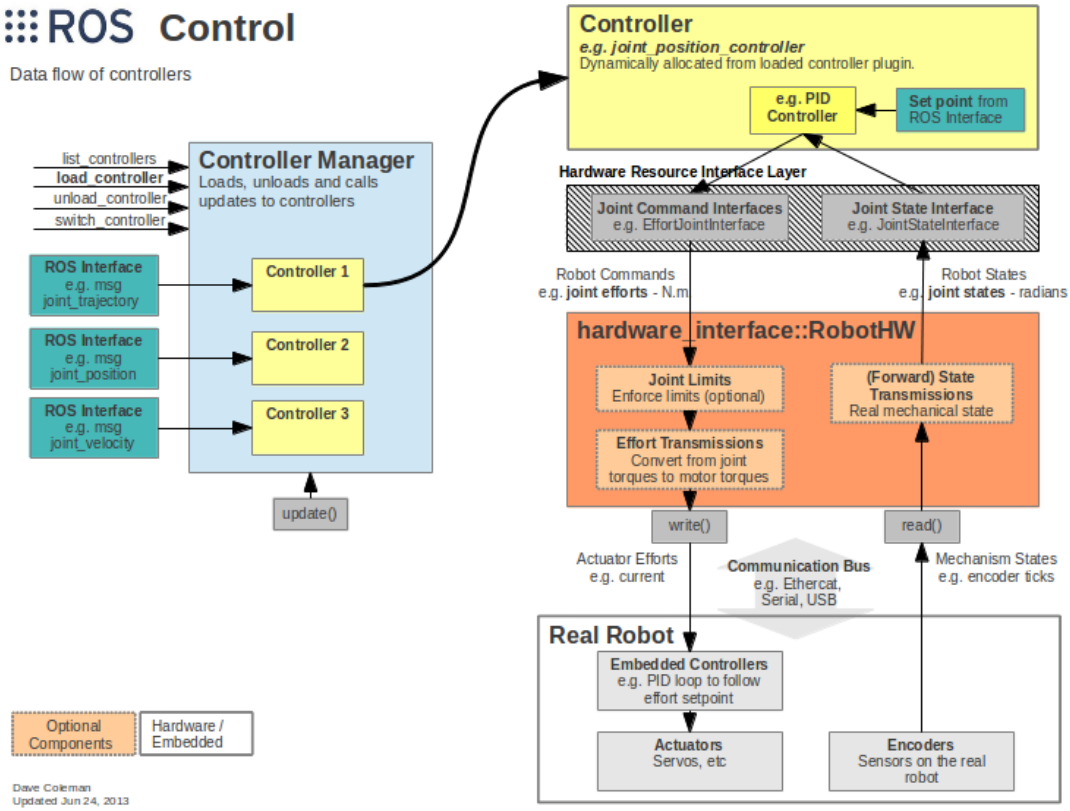


Рисунок 1.4 – Схема работы ROS_Control

1.2. Пакеты ROS для сканирования пространства.

ROS предлагает ряд пакетов для элементов системы сканирования пространства. В нашем случае интерес представляют только некоторые из них, предназначенные для работы с камерами и LiDAR.

1) usb_cam [14] – для получения изображения камер, подключенных к usb порту, и вывода его на экран RViz.
2) rplidar [15] – для работы с лидарами марки “rplidar”.
3) Freenect_stack [16] – для работы с камерой глубины Kinect azure.
4) Hector_slam [17] – для построения карты местности по данным

лидара.

5) Gmapping [18] – для построения карты по данным лидара и одометрии.

6) Rtabmap_ros [19] – для построения 3D карты с помощью камеры глубины по облаку точек.

Помимо упомянутых для автономной работы робота необходимы следующие пакеты:

1) Rosserial [20] – для использования основных элементов ROS с Arduino
2) Ros_control [21] – для отслеживания состояния робота, получения команд и отправления сигналов на приводы.
3) Move_base [22] – для получения целевой точки на карте, куда должен поехать робот, построения пути, обхода препятствий и отправления команд на контроллер.
4) Imu_tools [23] – для работы с датчиками imu.
5) teleop_twist_keyboard [24] – для управления роботом через клавиатуру (телеоперации).
6) Robot_localization [25] – для объединения данных датчиков одометрии и инерциальной навигации.
7) Robot_state_publisher [26] – для вычисления положения звеньев друг относительно друга.
8) Joint_state_publisher [27] – для отправки данных о состоянии каждого из звена.
9) Amcl [28] - для локализации робота по данным одометрии и лидара в известной карте.
10) Map_server [29] – для работы с картой местности, представленной в виде прямоугольной сетки.

1.3. Симуляция робота с системой сканирования пространства.

Алгоритм создания и запуска симуляции мобильного робота с системой сканирования пространства следующий:

1) Установить требуемые версии ROS [30] и Gazebo [31].
2) Создать рабочее пространство.
3) Создать пакет для симуляции.
4) Создать URDF файл модели.
5) Добавить к файлу настройки Gazebo
6) Добавить к файлу настройки ROS control.
7) Создать launch.py файл.
8) Скомпилировать рабочее пространство.
9) Запустить файл.

После установки ROS и Gazebo создается папка рабочего пространства space_scanner_ws, а в ней пакет для симуляции.

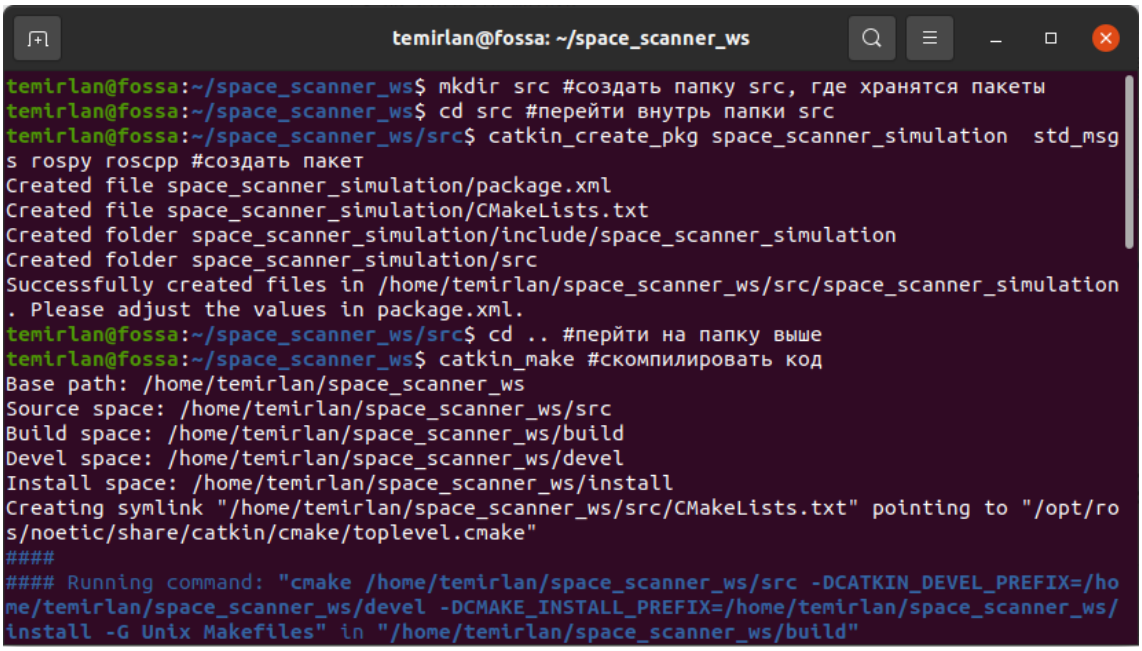


Рисунок 1.5 – Создание пакета в терминале

Выполнение данных команд добавляет подпапки в рабочем пространстве.

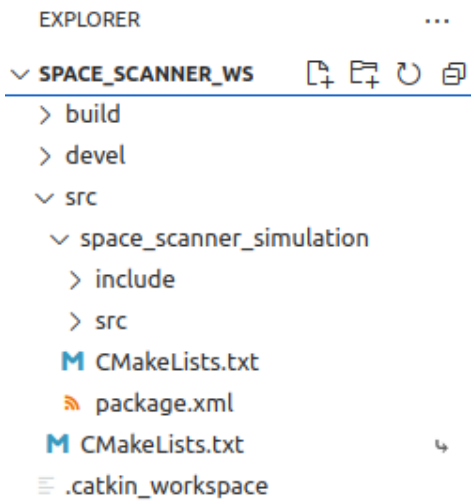


Рисунок 1.6 – Структура созданного рабочего пространства

Внутри пакета space_scanner_simulation создается папка urdf, в которой будут храниться urdf файлы для описания физических и геометрических параметров робота [32]. Формат URDF (United Robot Description Format) основан на XML, поэтому написание файлов данного формата состоит из использования тегов и добавления к ним значений. Чтобы определить модель

робота, следуют прописать теги звеньев “link” и теги кинематических пар “joint” [33].

materials.gazebo | chassi_bot.urdf.xacro | materials.xacro

src > space_scanner_simulation > urdf > chassi_bot.urdf.xacro > robot > xacro:include

```
<?xml version="1.0"?>
<robot name="test_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:include filename="$(find space_scanner_simulation)/urdf/include/materials.xacro" />
  <xacro:include filename="$(find space_scanner_simulation)/urdf/include/materials.gazebo" />

  <xacro:property name="rear_wheelX" value="-0.06"/>
  <xacro:property name="rear_wheelY" value="0.065"/>
  <xacro:property name="front_wheelX" value="0.06"/>
  <xacro:property name="front_wheelY" value="0.065"/>
  <xacro:property name="wheelZ" value="-0.03"/>
  <!-- **************************links****************************** -->
  <xacro:rectangular_cuboid_link suffix="base_link" box_w="0.01" box_d="0.01" box_h="0.01" colour="green" mass="10"/>
  <xacro:rectangular_cuboid_link suffix="upper_link" box_w="0.01" box_d="0.01" box_h="0.01" colour="green" mass="10"/>
  <xacro:rectangular_cuboid_link suffix="base_footprint" box_w="0.01" box_d="0.01" box_h="0.01" colour="green" mass="10" visual="false"/>
  <xacro:cylinder_link suffix="left_front_wheel" radius="0.03" height="0.025" colour="red" mass="1" rear ="false" />
  <xacro:cylinder_link suffix="right_front_wheel" radius="0.03" height="0.025" colour="red" mass="1" rear ="false" />
  <xacro:cylinder_link suffix="left_rear_wheel" radius="0.03" height="0.025" colour="blue" mass="1" />
  <xacro:cylinder_link suffix="right_rear_wheel" radius="0.03" height="0.025" colour="blue" mass="1"/>
  <xacro:cylinder_link suffix="laser_link" radius="0.035" height="0.025" colour="black" mass="1" rear ="false" visual="false"/>
  <xacro:rectangular_cuboid_link suffix="imu_link" box_w="0.01" box_d="0.01" box_h="0.01" colour="blue" mass="0.01" visual="false"/>

  <!-- ***********************Joints****************************** -->
  <xacro:fixed_joint suffix="base_part"  z="0.03" parent="base_link" child="upper_link" />
  <xacro:fixed_joint suffix="laser_joint" x="0.10" z="0.02" parent="upper_link" child="laser_link" />
  <xacro:fixed_joint suffix="basetofp"  z="${-2*0.03}" parent="base_link" child="base_footprint" />
  <xacro:fixed_joint suffix="base2imu"  y="-0.07" parent="upper_link" child="imu_link" />

  <joint name="wheel_0_joint" type="revolute">…
  </joint>
  <joint name="wheel_1_joint" type="revolute">…
  </joint>
  <joint name="wheel_3_joint" type="revolute">…
  </joint>
  <joint name="wheel_2_joint" type="revolute">…
  </joint>

</robot>
```

Рисунок 1.7 – URDF - файл робота

Для читабельности и краткости файла были использованы xacro шаблоны – фрагменты повторяющегося кода [34]. Шаблоны описаны в отдельном вложенном файле.

```
<!-- ************macros****************** -->
<xacro:macro name="rectangular_cuboid_link" params="suffix box_w box_d box_h colour mass collision:=true visual:=true">
    <link name="${suffix}">

        <visual>
            <material name="${colour}"/>
            <geometry>
                <xacro:if value="${visual}">
                    <mesh filename="package://space_scanner_simulation/meshes/base.dae" scale="1 1 1" />
                </xacro:if>
                <xacro:unless value="${visual}">
                    <box size="${box_w} ${box_d} ${box_h}"/>
                </xacro:unless>
            </geometry>
        </visual>

        <xacro:if value="${collision}">
            <collision>
                <geometry>
                    <box size="${box_w} ${box_d} ${box_h}"/>
                </geometry>
            </collision>
        </xacro:if>
        <inertial>
            <mass value="${mass}"/>
            <inertia ixx="${1/12*mass*(box_d*box_d+box_h*box_h)}" ixy="0.0" ixz="0.0" iyy="${1/12*mass*(box_w*box_w+box_h*box_
        </inertial>

        </link>
</xacro:macro>
```

Рисунок 1.8 – использование xacro шаблонов

После того, как urdf файл написан, можно будет визуализировать модель в Rviz.

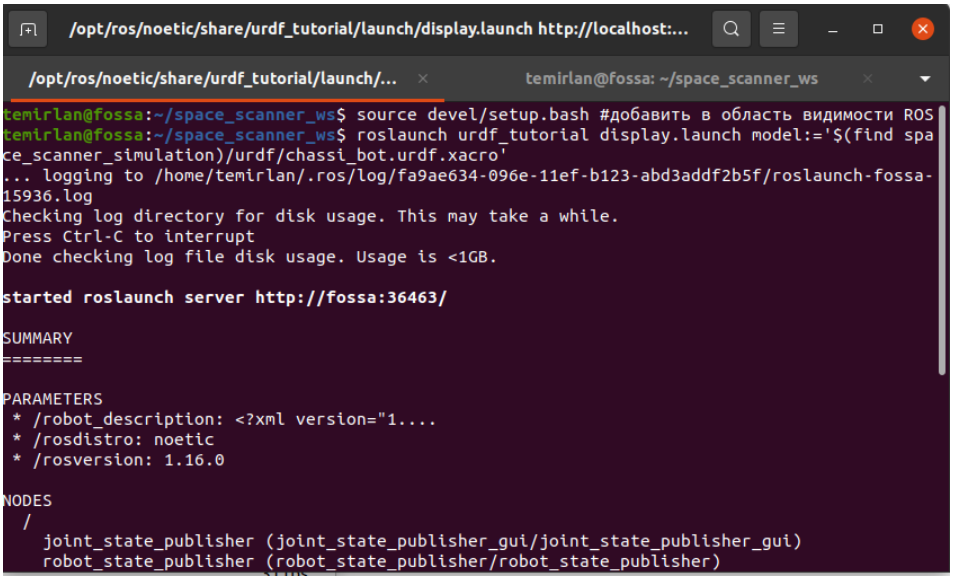


Рисунок 1.9 – Запуск узла визуализации.

После выполнения команды на экране выйдет графическое окно RViz. Пользователю необходимо нажать на кнопку “Add” и выбрать, что из

предложенного будет отображено на экране. Настроенную конфигурацию можно сохранить в отдельный “.rviz” файл.

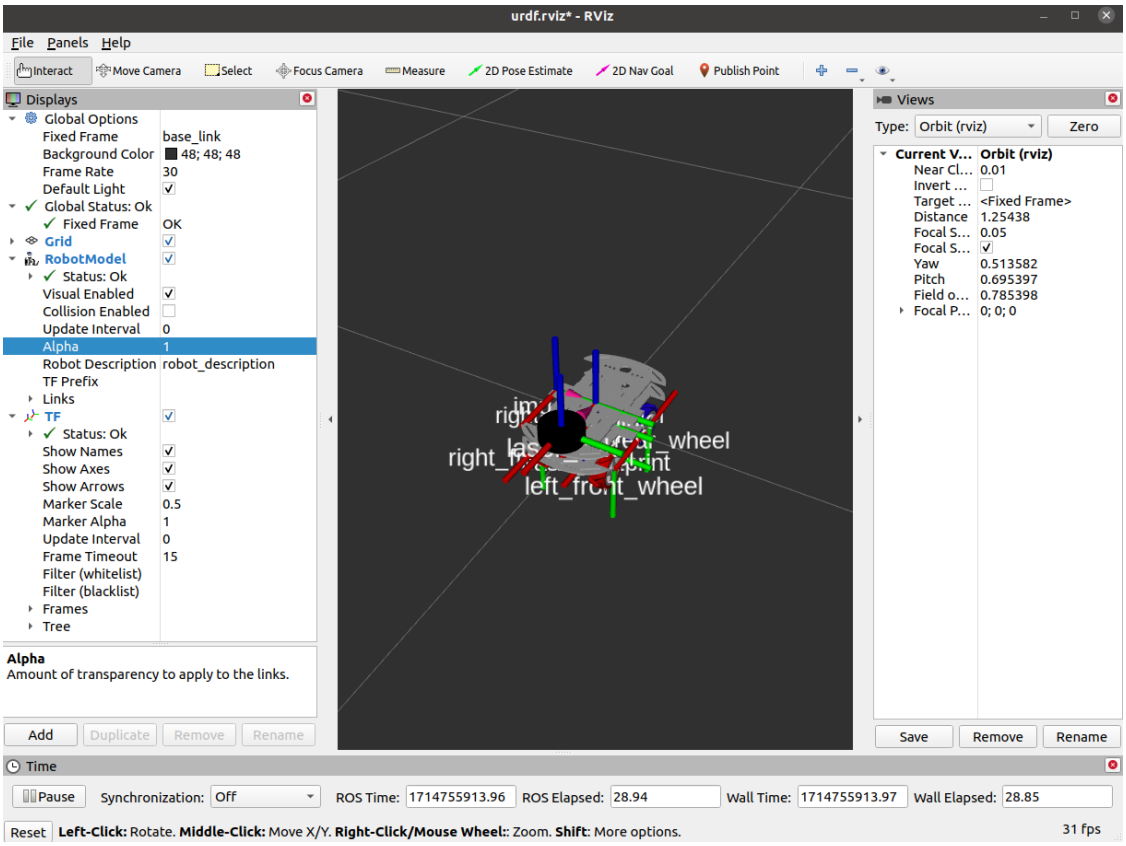


Рисунок 1.10 – Модель робота в Rviz

Чтобы использовать модель робота в Gazebo нужно добавить плагины датчиков и контроля. Для этого создается отдельный файл, в котором определяются данные элементы [35].

```
<?xml version="1.0"?>
<robot name="test_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:macro name="gz_ref" params="link colour:=0 mu1:=0 mu2:=0 kd:=1e10 kp:=1e10 minDepth:=0.01">…
    </xacro:macro>
    <xacro:property name="update_rate" value="50"/>
    <!-- ************links****************** -->
    <xacro:gz_ref link="laser_link" colour="Black" />
    <xacro:gz_ref link="base_link" colour="White"/>
    <xacro:gz_ref link="upper_link" colour="White"/>
    <xacro:gz_ref link="left_front_wheel" colour="Red" mu1="2" mu2="2" />
    <xacro:gz_ref link="left_rear_wheel" colour="Blue" mu1="2" mu2="2" />
    <xacro:gz_ref link="right_front_wheel" colour="Red" mu1="2" mu2="2" />
    <xacro:gz_ref link="right_rear_wheel" colour="Blue" mu1="2" mu2="2"/>
    <!-- plugins-->
    <gazebo reference="laser_link">…
    </gazebo>
    <gazebo reference="camera_link">…
      </gazebo>
    <gazebo>…
    </gazebo>
    <gazebo>…
    </gazebo>


</robot>
```

Рисунок 1.11 – Добавление элементов Gazebo

Затем пишется launch файл, в котором указываются все запускаемые узлы [36]. Для симуляции применяются узлы следующих пакетов: gazebo_ros, urdf_spawner, joint_state_publisher, robot_state_publisher, rviz, teleop_twist_keyboard.

```
mu1="2e-2" mu2="
<launch>
  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="model" default="$(find space_scanner_simulation)/urdf/chassi_bot.urdf.xacro"/>

  <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro $(arg model)" />

  <!-- push robot_description to factory and spawn robot in gazebo -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
        args="-z 0.05 -unpause -urdf -model robot -param robot_description" respawn="false" output="screen" />

  <node pkg="joint_state_publisher" type="joint_state_publisher"  name="joint_state_publisher"/>

  <node pkg="robot_state_publisher" type="robot_state_publisher"  name="robot_state_publisher">
    <!-- чтобы были tf данные о связи с base footprint -->
    <param name="use_tf_static" value="false"/>
  </node>
  <node type="rviz" name="rviz" pkg="rviz" args="-d $(find space_scanner_simulation)/rviz/config.rviz" />
  <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py" name="teleop"/>
</launch>
```

Рисунок 1.12 – launch файл для запуска симуляции и визуализации

Файл запускается через терминал командой roslaunch. Как только команда выполнится, на экране появятся окна Gazebo и RViz с роботом и данными датчиков сканирования пространства. Для телеоперации используются клавиши “u”, “i”, “o”, “j”, “k”, “l”, “m”, “,”, “.”.

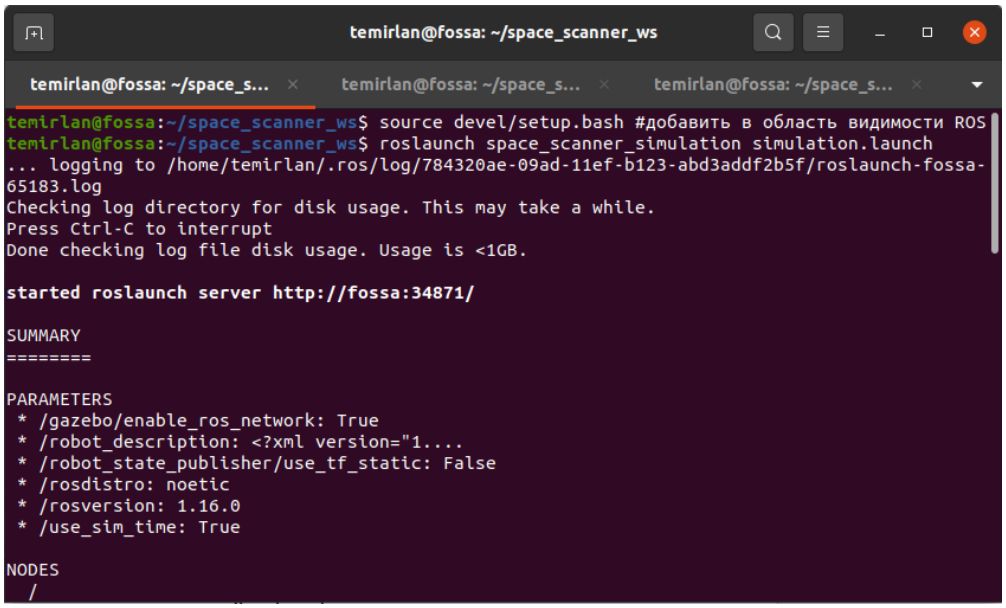


Рисунок 1.13 – Запуск launch файла

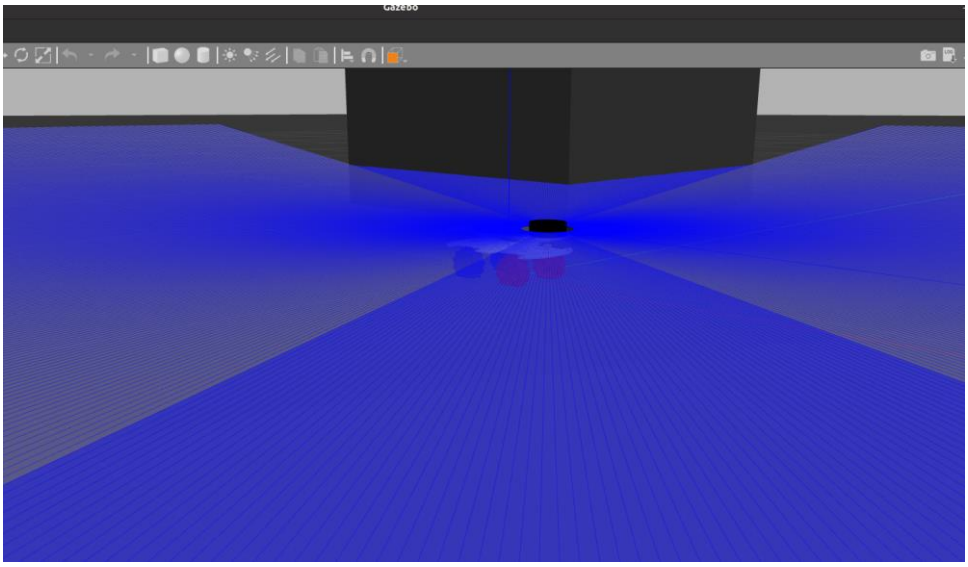


Рисунок 1.14 – Симуляция робота с системой сканирования пространства в Gazebo

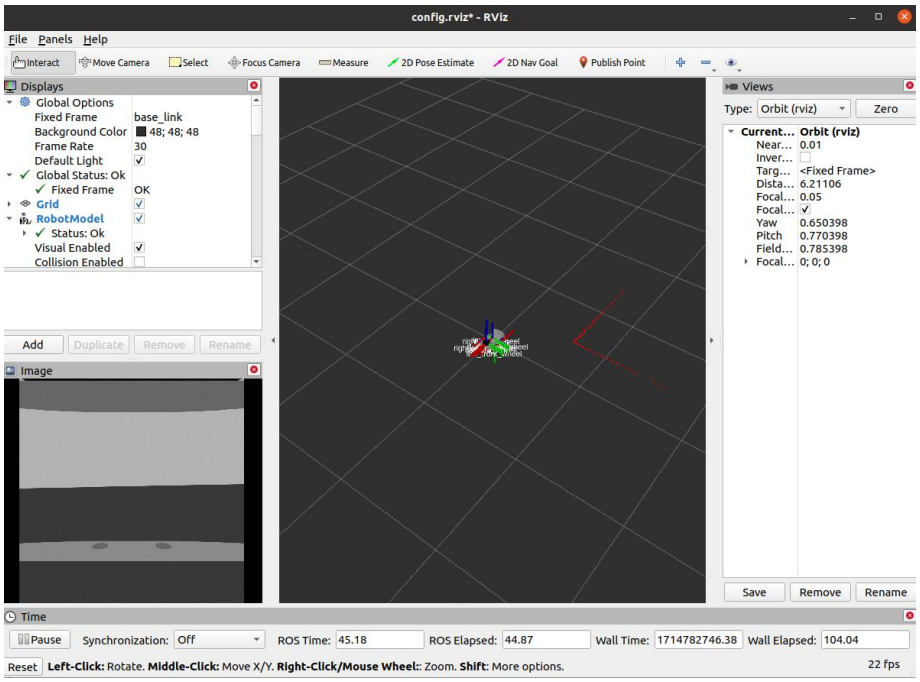


Рисунок 1.15 – Визуализация робота с системой сканирования пространства в Rviz

2. Практическая часть

2.1. Характеристика робота

Робот представляет собой сборочную 4-х колесную платформу, размерами 260х150х100мм.

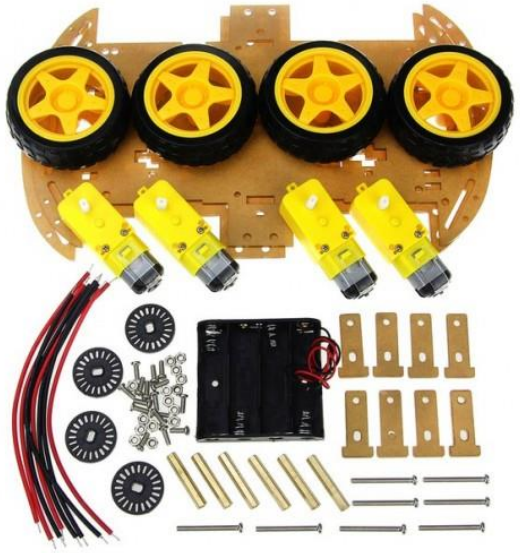


Рисунок 2.1 – Сборочное шасси и компоненты.

Платформа состоит из трех этажей, на первом из них – колеса, драйвер и энкодеры, на втором – блок батареек 3.7В на 4 штуки, на третьем – Raspberry Pi, повербанк, Arduino Mega, mpu6050, лидар и камера.

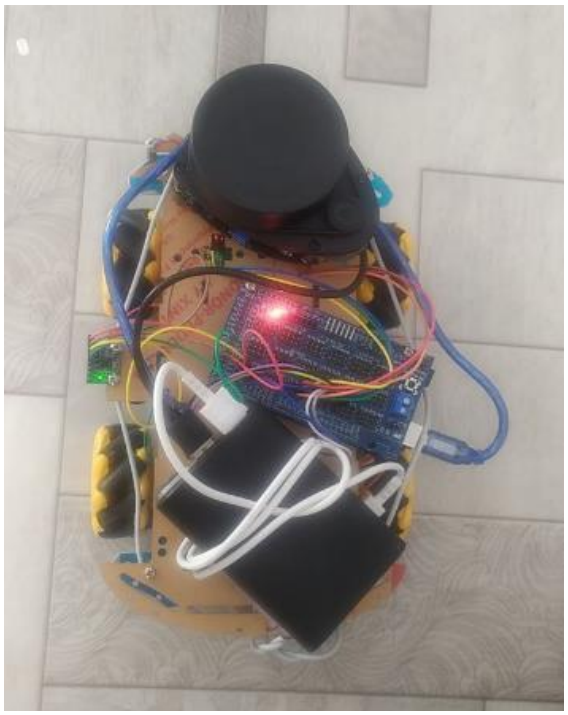


Рисунок 2.2 – Собранный робот

2.2. Электронные компоненты робота

В качестве микроконтроллера выступает Arduino Mega, который ответственен за передачу сигналов на драйвер L298N, соединенный с 4-мя колесами на 6-ти вольтовых моторах с редукторами.

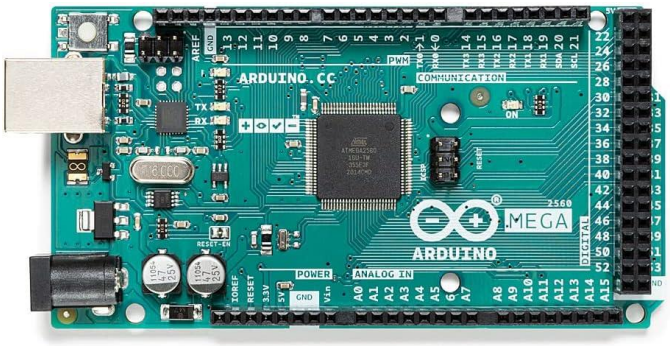


Рисунок 2.3 – Arduino Mega

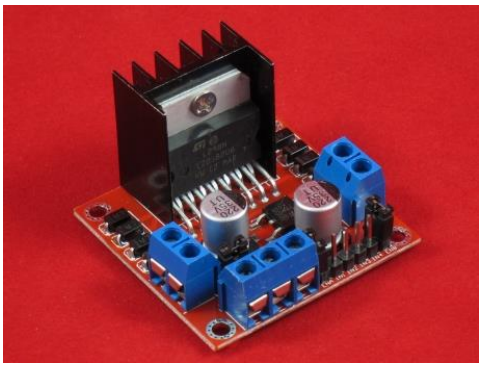


Рисунок 2.4 – L298N

Помимо драйвера к микроконтроллеру подключены два энкодера LM393 и модуль mpu6050 (гироскоп и акселерометр).

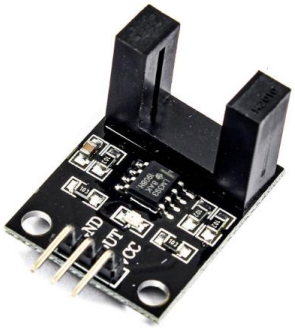


Рисунок 2.5 – LM393

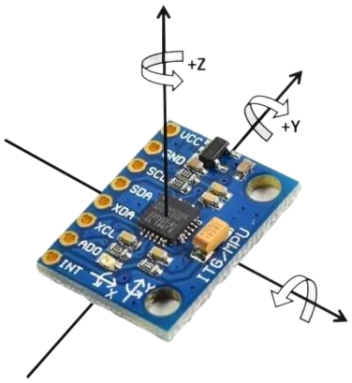


Рисунок 2.6 – mpu6050

В роли лидара выбран относительно дешевый RPLidar A1, камеры – raspberry camera v1.

Рисунок 2.7 – RPLidar A1

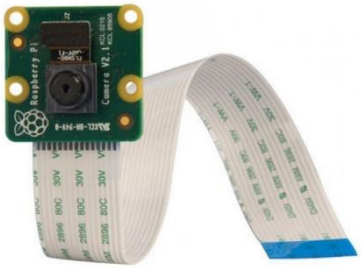


Рисунок 2.8 – Raspberry Camera

Вся система управляется микрокомпьютером Raspberry Pi 4b с установленной Ubuntu 20.04.

Рисунок 2.9 – Raspberry Pi

2.3. Электронная схема подключения.

Колеса, расположенные с одной стороны, попарно подключаются к соответствующим входам драйвера. К входам питания драйвера подключен блок батарей на 12в, земля с микроконтроллером общая. Контакты управления подключены к 4 по 9 пинам Arduino.

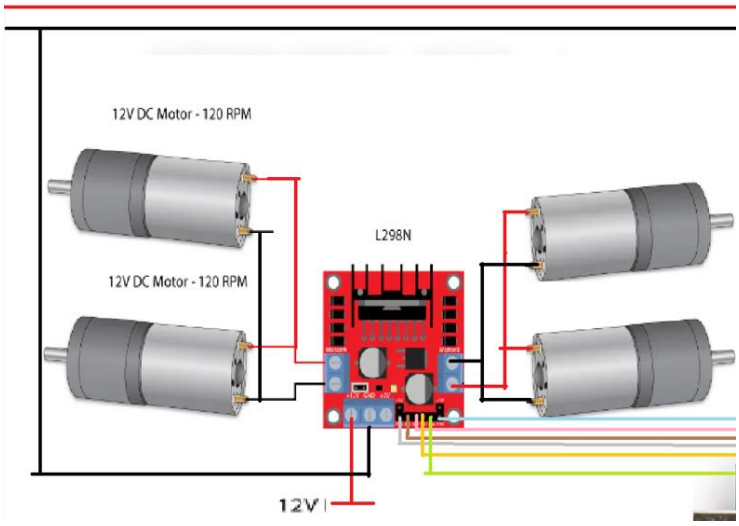


Рисунок 2.10 – Подключение моторов

Энкодеры имеют 3 выхода, два на питание, один сигнальный, подключенные ко 2 и 3 пину Arduino.

MPU6050 подключается к 4 входам микроконтроллера, 2 на питания, 2 к SCL/SDA пинам.

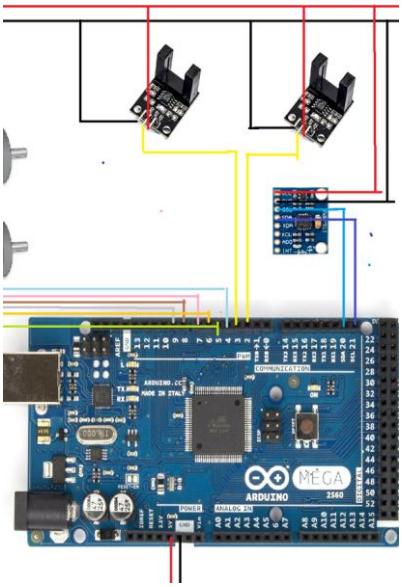


Рисунок 2.11 – Подключение датчиков

Полная схема подключения представлена ниже.

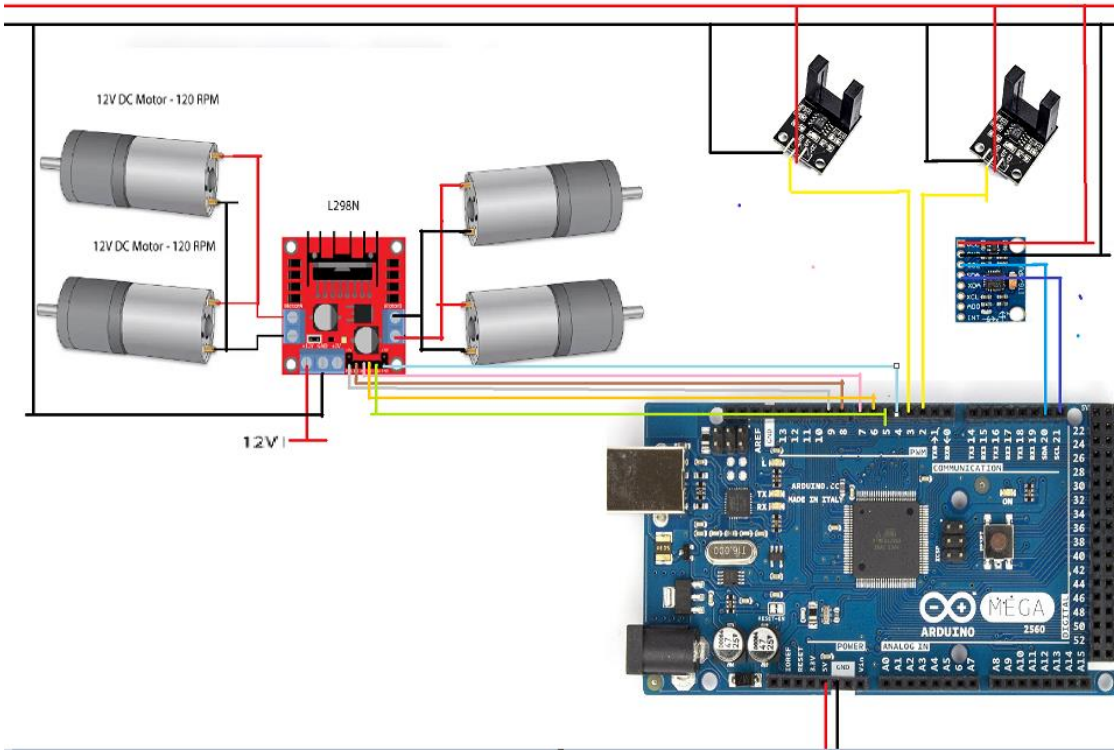


Рисунок 2.12 – Полная схема

Лидар и Arduino подключаются к USB портам микрокомпьютера, а шлейф камеры – в предназначенный разъем.

2.4. Контроль низшего уровня.

Все используемые программы написаны на c++. Ниже представлены только фрагменты. Полный листинг хранится в репозитории GitHub [37].

Скетч Arduino должен включать в себя основные элементы ROS для связи с системой. Ниже приведен фрагмент кода, создающий узлы подписчиков и издателей [38]. Посредством топиков они будут передавать и принимать данные датчиков и команды на двигатели.

```
//ros
ros::NodeHandle nh;
sensor_msgs::JointState cmd_msg;
sensor_msgs::Imu imu_msg;
std_msgs::Float64 lm, rm;
void leftWheelCB(const std_msgs::Int32 &lCb);
void rightWheelCB(const std_msgs::Int32 &rCb);
ros::Publisher leftWheelPublisher("motor_left_vel", &lm);
ros::Publisher rightWheelPublisher("motor_right_vel", &rm);
ros::Publisher imuPublisher("imu/data_raw", &imu_msg);

ros::Subscriber<std_msgs::Int32> leftWheelSubscriber ("motor_left_cmd", &leftWheelCB);
ros::Subscriber<std_msgs::Int32> rightWheelSubscriber ("motor_right_cmd", &rightWheelCB);
```

Рисунок 2.13 – Элементы ROS в Arduino

Необходимо определить функции обратной связи, которые будут выполняться при получении сообщения через топик. Как только оно будет принято, его значение будет присвоено переменной, которая отправляется на PID контроллер.

```
void leftWheelCB(const std_msgs::Int32 &lCb) {
  setupVelL = float(lCb.data);
}
void rightWheelCB(const std_msgs::Int32 &rCb) {
  setupVelR = float(rCb.data);
}
```

Рисунок 2.14 – Функции обратной связи

В функции setup задается скорость передачи данных по серийному порту и запускаются узлы.

```
void setup() {
  Serial.begin(57600);
  attachInterrupt(encL.pin, encL_update, CHANGE);
  attachInterrupt(encR.pin, encR_update, CHANGE);
  nh.initNode();
  nh.advertise(leftWheelPublisher);
  nh.advertise(rightWheelPublisher);
  nh.advertise(imuPublisher);
  nh.subscribe(leftWheelSubscriber);
  nh.subscribe(rightWheelSubscriber);
  setup_imu(mpu6050);


}
```

Рисунок 2.15 – Функция setup()

В основном цикле происходит получение актуальных данных датчиков и передача их системе, а также принятие команд от контроллера и управление двигателями.

```
void loop() {
  encL.checkVel();
  encR.checkVel();
  lm.data = encL.getVel();
  rm.data = encR.getVel();
  leftWheelPublisher.publish(&lm);
  rightWheelPublisher.publish(&rm);
  //if (abs(setupVelL)<10 && (setupVelL)==-(setupVelR) && setupVelL!=0) setupVelL=10; setupVelR=10;
  leftPID.control(setupVelL, encL, L298n, (millis()-cTime)*0.001);
  rightPID.control(setupVelR, encR, L298n, (millis()-cTime)*0.001);
  //Serial.println(encL.getVel());
  nh.spinOnce();
  cTime = millis();
  if (cTime - imuTime > 100) {
    imu_data(mpu6050, imu_msg, nh);
    imuPublisher.publish(&imu_msg);
    imuTime=cTime;
//    nh.spinOnce();
//    cTime = millis();
  }
  delay(1);
}
```

Рисунок 2.16 – Основной цикл

2.5. Контроль высшего уровня

Для связи низшего уровня с ROS нужно создать класс, унаследованный от RobotHW, и прописать методы read (для чтения данных датчиков, переданных от низшего контроля) и write (для передачи команд на микроконтроллер) [39].

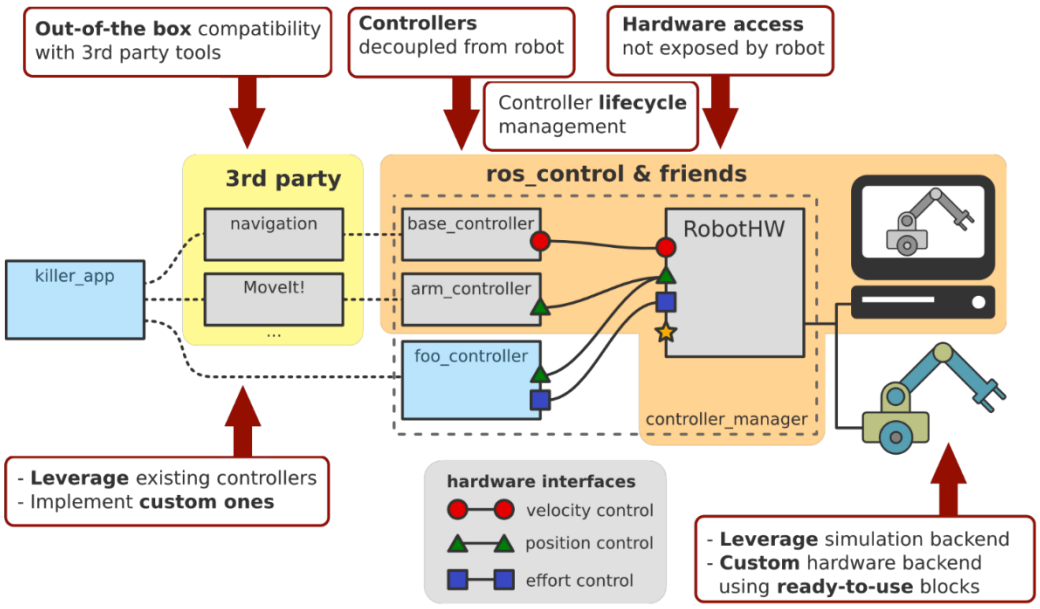


Рисунок 2.17 – Схема ROS Control

В нашем случае в качестве контроллера выступает diff_drive_controller [40]. Он будет принимать данные о текущих скоростях вращения колес робота, высчитывать, с какой скоростью они должны крутиться, чтобы соответствовать командам телеоперации (которые публикуются в топик cmd_vel), а также отправлять команды на моторы.

Для того, чтобы связать класс RobotHW и контроллер, нужно прописать интерфейсы – абстрактную репрезентацию робота [41]. Стандартные интерфейсы представляют собой интерфейсы состояния (Joint State Interface) и команд (Joint Command Interface). В программах они выражены классами, хранящими название кинематической пары, а также данные о положении (угол, расстояние), скорости и усилии ее элементов [42]. Если это командный интерфейс, то он хранит значение команды, передаваемое контроллером, и в зависимости от вида команд может управлять скоростью, позицией, усилием.

Рассматриваемый мобильный робот описывается 4 интерфейсами состояния и команд, по одному на каждое колесо. В качестве командного интерфейса выступает интерфейс скорости Velocity Joint Interface [43].

Связь между RobotHW и контроллером осуществляется менеджером контроллеров, который считывает значения интерфейсов состояния и меняет переменные интерфейсов команд [44].

Для программной реализации контроля высшего уровня создаются два файла, один заголовочный (.h), второй основной (.cpp). В первом определяется

класс RobotHW, во втором – происходит его связь с контролером и реализуются методы read() и write().

В заголовочном файле в первую очередь прописывается конструктор нового класса, в котором создаются и регистрируются интерфейсы.

```
class Diffbot : public hardware_interface::RobotHW
{
public:
  Diffbot()
  : running_(true)
  , start_srv_(nh_.advertiseService("start", &Diffbot::start_callback, this))
  , stop_srv_(nh_.advertiseService("stop", &Diffbot::stop_callback, this))
  {
    // Intialize raw data
    std::fill_n(pos_, NUM_JOINTS, 0.0);
    std::fill_n(vel_, NUM_JOINTS, 0.0);
    std::fill_n(eff_, NUM_JOINTS, 0.0);
    std::fill_n(cmd_, NUM_JOINTS, 0.0);

    // Connect and register the joint state and velocity interface
    for (unsigned int i = 0; i < NUM_JOINTS; ++i)
    {
      std::ostringstream os;
      os << "wheel_" << i << "_joint";

      hardware_interface::JointStateHandle state_handle(os.str(), &pos_[i], &vel_[i], &eff_[i]);
      jnt_state_interface_.registerHandle(state_handle);

      hardware_interface::JointHandle vel_handle(jnt_state_interface_.getHandle(os.str()), &cmd_[i]);
      jnt_vel_interface_.registerHandle(vel_handle);
    }

    registerInterface(&jnt_state_interface_);
    registerInterface(&jnt_vel_interface_);
```

Рисунок 2.18 – Создание и регистрация интерфейсов в классе.

Затем происходит запуск узлов подписчиков/издателей и определение функций обратной связи.

```
void  leftMotorCallback(const std_msgs::Float64::ConstPtr& msg){
  vel_[0]=msg->data;
  vel_[2]=vel_[0];
  //ROS_WARN_STREAM("Commands for joints: " <<vel_[0]);
}
void  rightMotorCallback(const std_msgs::Float64::ConstPtr& msg){
  vel_[1]=msg->data;
  vel_[3]=vel_[1];
  //ROS_WARN_STREAM("Commands for joints: " <<vel_[1]);

}
void init(ros::NodeHandle &nh){
  nh_=nh;
  pub_left_motor_value_=nh_.advertise<std_msgs::Int32>("motor_left_cmd", 10);
  pub_right_motor_value_=nh_.advertise<std_msgs::Int32>("motor_right_cmd",10);
  sub_left_motor_vel_=nh_.subscribe("motor_left_vel", 10, &Diffbot::leftMotorCallback, this);
  sub_right_motor_vel_=nh_.subscribe("motor_right_vel", 10, &Diffbot::rightMotorCallback, this);
  ROS_INFO_STREAM("init completed");
}
```

Рисунок 2.19 – Настройка узлов подписчиков и издателей

После этого определяется метод read() предназначенный для получения данных датчиков о состоянии кинематических пар робота.

```
void read()
{
  // Read the joint state of the robot into the hardware interface
  if (running_)
  {
    for (unsigned int i = 0; i < NUM_JOINTS; ++i)
    {
      // Note that pos_[i] will be NaN for one more cycle after we start(),
      // but that is consistent with the knowledge we have about the state
      // of the robot.
      pos_[i] += vel_[i]*getPeriod().toSec(); // update position
      //vel_[i] = cmd_[i]; // might add smoothing here later
    }
  }
  else
  {
    std::fill_n(pos_, NUM_JOINTS, std::numeric_limits<double>::quiet_NaN());
    std::fill_n(vel_, NUM_JOINTS, std::numeric_limits<double>::quiet_NaN());
  }
}
```

Рисунок 2.20 – Функция read()

Наконец, прописывается метод write() для отправки команд на микроконтроллер.

```
  void write()
{
  // // Write the commands to the joints
  // std::ostringstream os;
  // for (unsigned int i = 0; i < NUM_JOINTS - 1; ++i)
  // {
  //   os << vel_[i] << ", ";
  // }
  // os << vel_[NUM_JOINTS - 1];

  //ROS_WARN_STREAM("Commands for vel: " << os.str());
  if (cmd_[0]!=lcmd.data || cmd_[1]!=rcmd.data){
    lcmd.data=cmd_[0];
    rcmd.data=cmd_[1];
    pub_left_motor_value_.publish(lcmd);
    pub_right_motor_value_.publish(rcmd);
  }

}
```

Рисунок 2.21 – Функция write()

Все переменные класса находятся в приватной секции.

```
private:
  hardware_interface::JointStateInterface    jnt_state_interface_;
  hardware_interface::VelocityJointInterface jnt_vel_interface_;
  double cmd_[NUM_JOINTS];
  double pos_[NUM_JOINTS];
  double vel_[NUM_JOINTS];
  double eff_[NUM_JOINTS];
  bool running_;

  ros::NodeHandle nh_;
  ros::ServiceServer start_srv_;
  ros::ServiceServer stop_srv_;

  ros::Publisher pub_left_motor_value_;
  ros::Publisher pub_right_motor_value_;
  ros::Subscriber sub_left_motor_vel_;
  ros::Subscriber sub_right_motor_vel_;
  std_msgs::Int32 rcmd;
  std_msgs::Int32 lcmd;
```

Рисунок 2.22 – Переменные класса

Во втором файле .cpp менеджер контроллеров связывает контроллер с роботом и запускается основной цикл.

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

  // This should be set in launch files as well
  nh.setParam("/use_sim_time", true);

  Diffbot<4> robot;
  ROS_WARN_STREAM("period: " << robot.getPeriod().toSec());
  controller_manager::ControllerManager cm(&robot, nh);
```

Рисунок 2.23 – Связь робота с контроллером

Основной цикл включает в себя три функции read(), update() и write().

```
while(ros::ok())
{
  begin = std::chrono::system_clock::now();

  robot.read();
  cm.update(internal_time, dt);

  robot.write();

  end = std::chrono::system_clock::now();

  elapsed_secs = std::chrono::duration_cast<std::chrono::duration<double> >((end - begin)).count();

  if (dt.toSec() - elapsed_secs < 0.0)
  {
    ROS_WARN_STREAM_THROTTLE(
          0.1, "Control cycle is taking to much time, elapsed: " << elapsed_secs);
  }
  else
  {
    ROS_DEBUG_STREAM_THROTTLE(1.0, "Control cycle is, elapsed: " << elapsed_secs);
    std::this_thread::sleep_for(std::chrono::duration<double>(dt.toSec() - elapsed_secs));
  }

  rosgraph_msgs::Clock clock;
  clock.clock = ros::Time(internal_time);
  clock_publisher.publish(clock);
  internal_time += dt;
}
```

Рисунок 2.24 – Основной цикл контроллера

При создании рассмотренных файлов был использован шаблон для мобильных роботов с дифференциальным приводом [45].

2.6. Конфигурация ROS для системы сканирования пространства.

Все программы, необходимые для запуска системы сканирования пространства хранятся в соответствующих пакетах, и дополнительных узлов писать не требуется.

В первом launch файле запускаются узлы, предназначенные для связи мобильного робота с ROS и настройки управления.

```
<?xml version="1.0"?>
<launch>
  <!-- Use simulation time -->
  <rosparam param="use_sim_time">false</rosparam>
  <arg name="use_tf_static" default="false"/>
  <!-- Load diffbot model -->
  <param name="robot_description" command="$(find xacro)/xacro '$(find diff_drive_controller)/urdf/chassi_bot.urdf.xacro'" />
  <!-- Start diffbot -->
  <node name="diffbot" pkg="diff_drive_controller" type="diffbot"/>
  <!-- Load controller config -->
  <rosparam command="load" file="$(find diff_drive_controller)/test/diffbot_controllers.yaml" />
  <rosparam command="load" file="$(find diff_drive_controller)/test/diffbot_limits.yaml" />
  <!-- Spawn controller -->
  <node name="controller_spawner"
        pkg="controller_manager" type="spawner" output="screen"
        args="diffbot_controller joint_state_controller" />
   <node pkg="robot_state_publisher" type="robot_state_publisher" name="rob_st_pub" >
      <!-- remap from="robot_description" to="my_robot_description" /-->
      <!--remap from="diffbot_controller/odom" to="odom" /-->
      <!--param name="use_tf_static" value="$(arg use_tf_static)"/-->
   </node>
     <!-- send joint values -->
  <!-- <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher"/> -->
   <include file="$(find rplidar_ros)/launch/rplidar.launch" />
   <include file="$(find imu_filter_madgwick)/launch/imu_filter_madgwick.launch" />
  <node pkg="robot_localization" type="ekf_localization_node" name="ekf_loc_node" clear_params="true">
  <rosparam command="load" file="$(find diff_drive_controller)/config/ekf.yaml"/>
  <remap from="odometry/filtered" to="odom"/>
  </node>
   <node name="serial_node" pkg="rosserial_python" type="serial_node.py">
        <param name="port" type="string" value="/dev/ttyACM0" />
        <param name="baud" type="int" value="57600" />
   </node>
</launch>
```

Рисунок 2.25 – Launch файл контроля

Второй launch файл запускает gmapping для создания карты местности системой сканирования пространства. В нем указываются такие параметры, как интервал обновления, дальность сканирования, используемая дальность сканирования, максимальные движения (линейные, вращательные) после которых происходит обработка данных. Построенная карта сохраняется в отдельный файл и используется при навигации.

```
<?xml version="1.0"?>
<launch>
<include file="$(find diff_drive_controller)/launch/robot_config.launch" />
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
   <param name="map_update_interval" value="5.0"/>
  <param name="maxUrange" value="6.0"/>
   <param name="maxRange" value="8.0"/>
   <param name="linearUpdate" value="0.05"/>
  <param name="angularUpdate" value="0.5"/>
 </node>
    <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py" name="teleop" output="screen">
   <remap from="/cmd_vel" to ="/diffbot_controller/cmd_vel" />
   </node>

</launch>
```

Рисунок 2.26 – Launch файл создания карты

Третий launch файл запускает узлы для навигации и автономного

движения. В нем загружается построенная карта, amcl – метод монте карло для локализации по данным лидара и одометрии и параметры для планировщика путей.

```
<?xml version="1.0"?>
<launch>
        <include file="$(find diff_drive_controller)/launch/robot_config.launch" />
         <arg name="map_file" default="$(find diff_drive_controller)/maps/mymap.yaml"/>

        <!-- Run the map server -->
        <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

 <!--- Run AMCL -->
    <!--include file="$(find amcl)/examples/amcl_diff.launch" /-->
     <include file="$(find diff_drive_controller)/launch/amcl.launch" />
    <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find diff_drive_controller)/config/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find diff_drive_controller)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find diff_drive_controller)/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find diff_drive_controller)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find diff_drive_controller)/config/base_local_planner_params.yaml" command="load" />
    <param name="controller_frequency" value="10" />
    <remap from="/cmd_vel" to="/diffbot_controller/cmd_vel" />

 </node>
</launch>
```

Рисунок 2.27 – launch файл автономного движения

В планировщике путей уточняются минимальные/максимальные значения линейных, угловых скоростей, а также ускорения.

```
TrajectoryPlannerROS:
  max_vel_x: 0.6
  min_vel_x: 0.35 #0.2
  max_vel_theta: 0.05
  min_vel_theta: -0.05
  min_in_place_vel_theta: 0.9 #10

  acc_lim_theta: 5
  acc_lim_x: 1
  acc_lim_y: 1

  holonomic_robot: false
  yaw_goal_tolerance: 0.15
  xy_goal_tolerance: 0.2
```

Рисунок 2.28 – Параметры планировщика путей

В параметрах карты указываются названия систем отсчета робота и карты, а также геометрические параметры робота и препятствий.

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[-0.08,-0.12], [-0.08,0.12], [0.08,0.12], [0.08,-0.12]]



#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor

laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic: scan, marking: true, clearing: true}

plugins:
  - {name: static,                 type: "costmap_2d::StaticLayer"}
  - {name: inflation_g,              type: "costmap_2d::InflationLayer"}
```

Рисунок 2.29 – Геометрические параметры робота и препятствий

```
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 2
  static_map: true
```

Рисунок 2.30 – Параметры карты

2.7. Результат работы

После запуска launch файлов робот при движении начал строить карту комнаты, которая отображалась в Rviz.

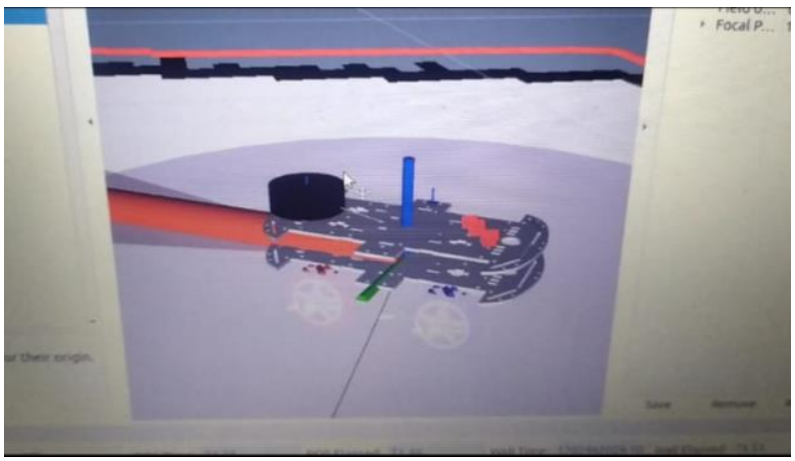


Рисунок 2.31 – Визуалиция робота в Rviz

Во время остановок система сканирования, учитывая движение робота, сравнивает предыдущие и текущие данные, исправляя ошибки и неточности.

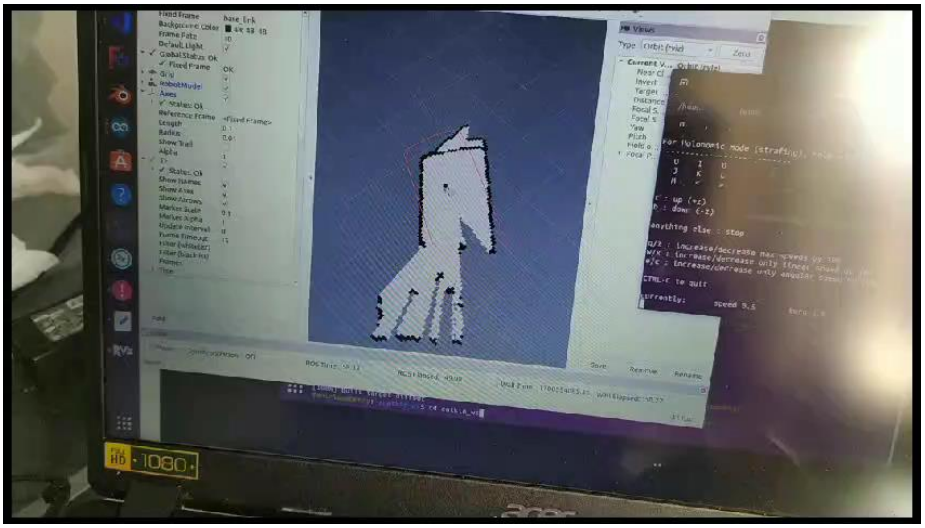


Рисунок 2.32 – Начало построения карты

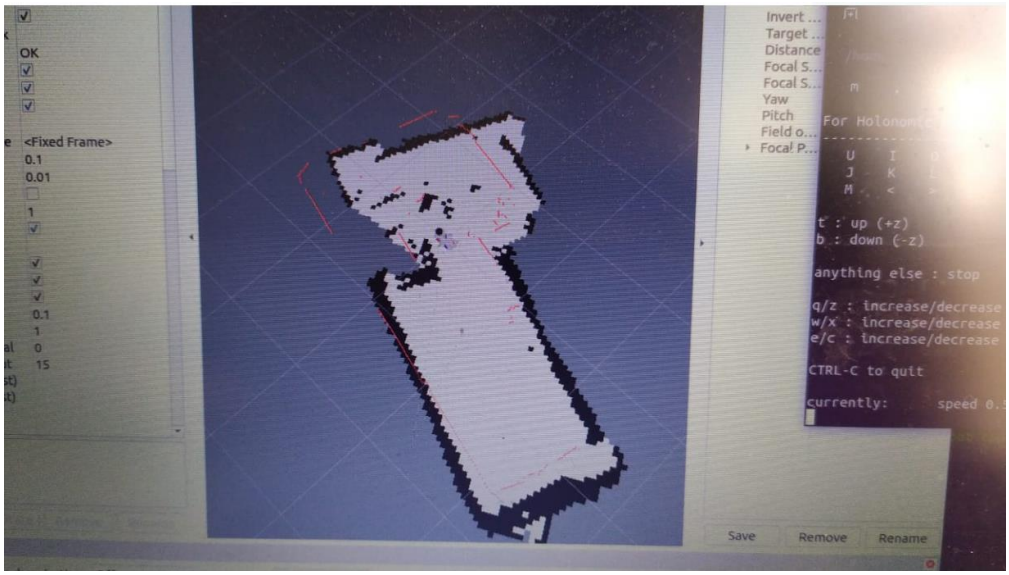


Рисунок 2.33 – Исправление неточностей

Как только карта полностью построится, ее описание в виде сетки занятости можно будет сохранить в отдельный файл.

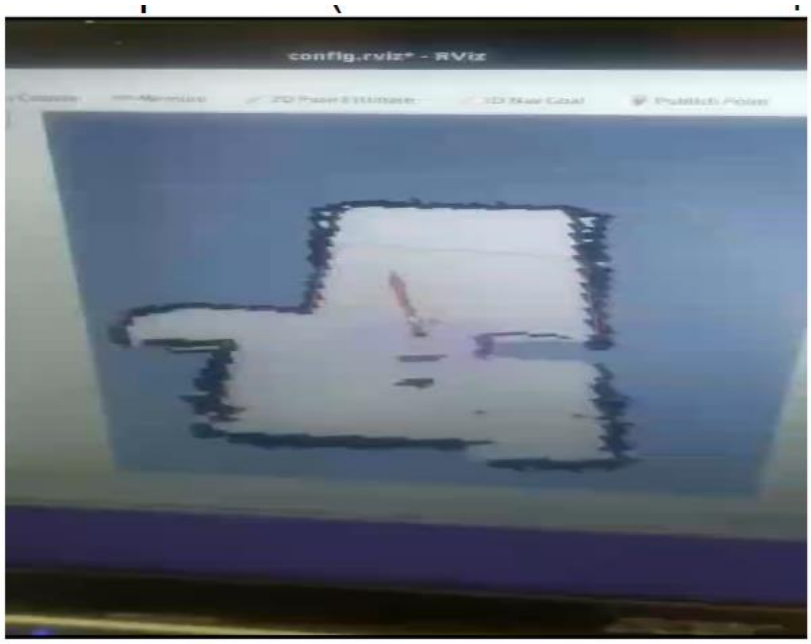


Рисунок 2.34 – Построенная карта комнаты.

3. Модели и расчеты

3.1. LiDar.

Основной датчик – LiDar, работает по принципу фазового дальномера. Если расстояние, пройденное светом за время t

$$l = ct \tag{1}$$

То изменение фазы определяется следующим выражением

$$\varphi = 2\pi f t \tag{2}$$

Рисунок 3.1 – Испущенная и отраженная волны

3.2. IMU

Модель измерений акселерометра, включающая аддитивную и мультипликативную погрешности следующая [46]

$$M \approx Am + b \tag{3}$$

Истинное значение находится преобразованием исходного выражения.

$$A^{-1}(M - b) = m \tag{4}$$

Чтобы найти b и A используется метод вписанного эллипсоида.

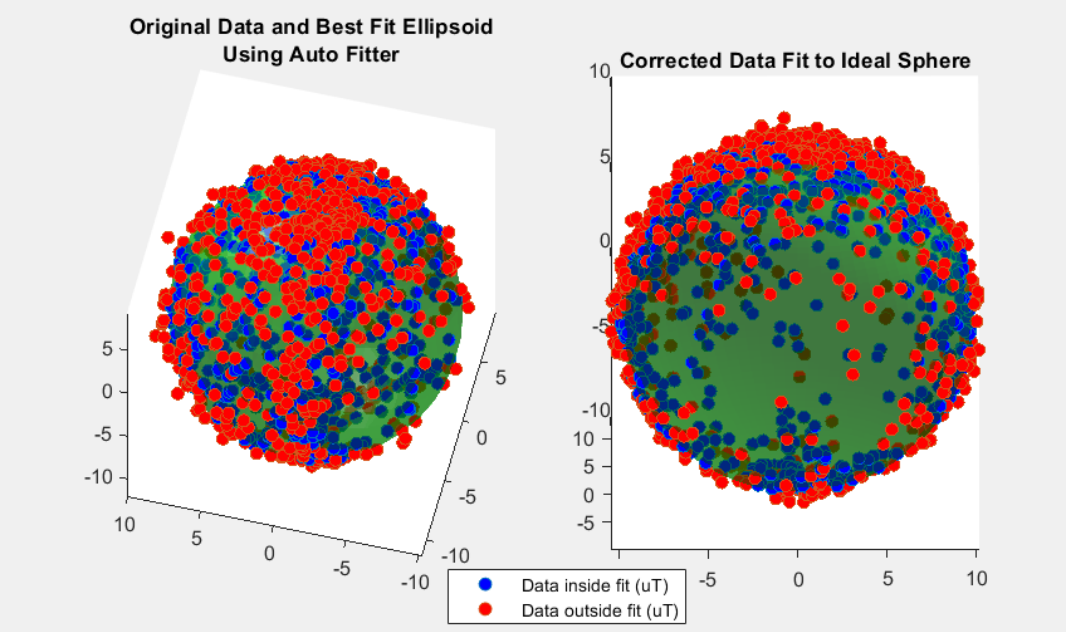


Рисунок 3.2 – Сырые и откалиброванные значения

Модель измерения гироскопа схожа с рассмотренной, только в ней мультипликативной погрешностью можно пренебречь. Чтобы убрать аддитивную погрешность следует собрать массив данных в состоянии покоя и отнять его среднее число.

3.3. Оптические энкодеры

Выходной сигнал цифрового энкодера – логический. В идеале его график представляет собой периодическую пилообразную волну, если колеса крутятся с постоянной скоростью.

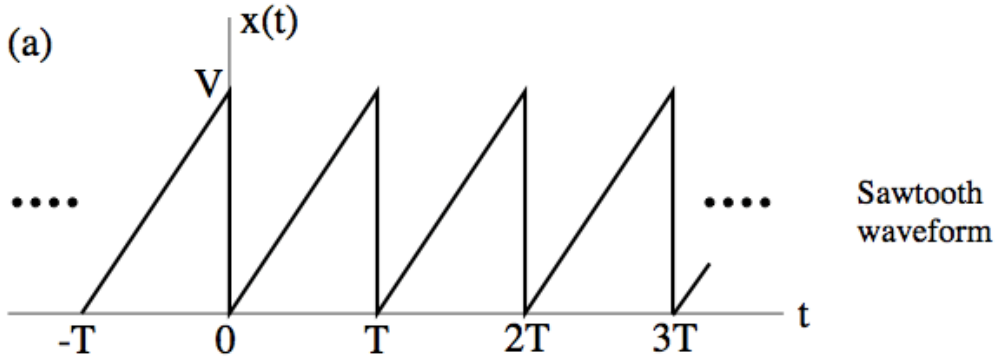


Рисунок 3.3 – Пилообразная волна

Однако данный датчик подвержен шумам, дребезгу и прочим негативным воздействиям, что приводит к “подрезанию” углов – один и тот же сигнал зачитывается два раза, либо происходит его пропуск, и местами образуются ступени.

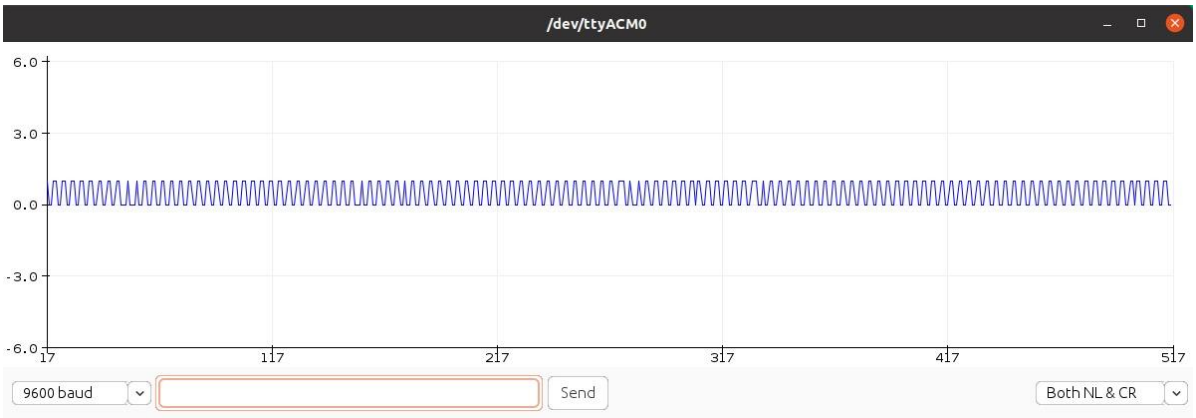


Рисунок 3.4 – График вывода энкодера

Рисунок 3.5 – Вывод энкодера

Чтобы исправить эту погрешность, следует программно пропускать значения, которые повторяются с предыдущими.

3.4. PID контроль

Значения для PID контроля были выбраны эмпирическим способом [47]. В зависимости от результата работы, меняя порядки чисел, сначала был выбран пропорциональный коэффициент P, затем интегрирования I, и наконец D. Формула, связывающая выходной сигнал с ошибкой, следующая.

$$u(t) = K_p e(t) + K_i \int_0^t e(z)dz + K_d \dot{e}(t) \quad (5)$$

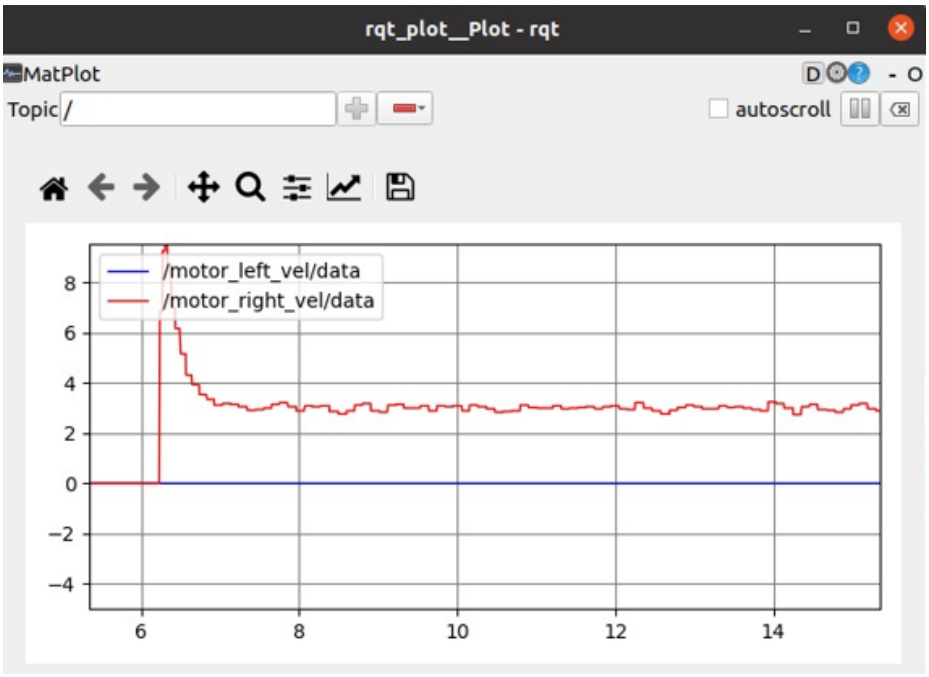


Рисунок 3.6 – График скорости вращения колеса

3.5. Модель движения

Контроль дифференциального привода описывается двумя уравнениями, определяющими угловые скорости каждого из колес, в зависимости от необходимой линейной скорости, с которой должен двигаться робот и расстояния между колесами.

$$\omega_L = \frac{V - \omega * \frac{b}{2}}{r} \tag{6}$$

$$\omega_R = \frac{V + \omega * \frac{b}{2}}{r} \tag{7}$$

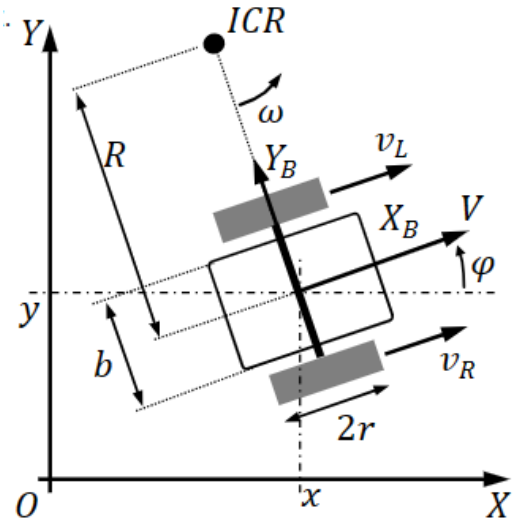


Рисунок 3.7 – Схема движения робота

3.6. Инерциальные параметры

Для расчетов моментов инерции были использованы соответствующие формулы для цилиндра и параллелепипеда

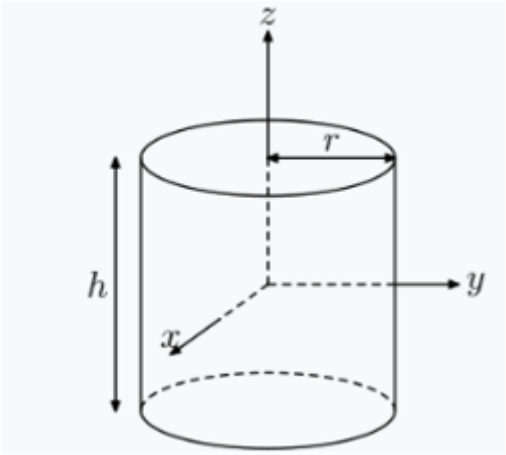


Рисунок 3.8 – Цилиндр и привязанная система координат

$$I_z = \frac{1}{2}mr^2 \tag{8}$$

$$I_x = I_y = \frac{1}{12}m(3r^2 + h^2) \tag{9}$$

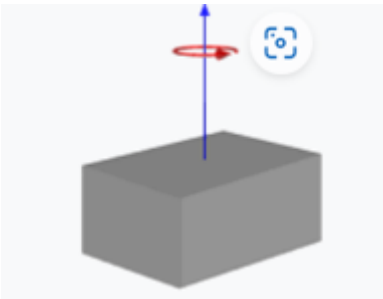


Рисунок 3.9 – Параллелепипед

$$I_x = \frac{1}{12} m(y^2 + z^2) \quad (10)$$

$$I_y = \frac{1}{12} m(x^2 + z^2) \quad (11)$$

$$I_z = \frac{1}{12} m(x^2 + y^2) \quad (12)$$

3.7. Экономический расчет и время работы.

В таблице представлены цены компонентов в Казахстане и Китае.

Компонент	**Цена в Казахстане(тг)**	**Цена в Китае (тг)**	**Потребляемый ток мА**
L393	1000 x 2	700	10x2
Arduino Mega	7690	6000	100
MPU6050	1000	500	10
L298N	1200	510	50
Raspberry pi 4b	68000	40000	1500
RPLidar A1	-	40000	250
Шасси	7000	4000	125 x 4
Аккумулятор 3,7В (2000мАч)	2000 x 4	1500	-

Raspberry Camera	2500	1200	250
Провода	1000	500	-
Sensor Shield	1750	760	10
Повербанк (10АЧ)	9000	8600	-
Итог	109140 + 50000	104270	2140+550

Для оценки времени работы воспользуемся формулой, содержащей емкость и потребляемый ток.

$$T = \frac{Q}{I} \tag{13}$$

T (драйвер + моторы) =2000/550=3,63Ч

T (RPi+Lidar+Arduino) =10000/2140=4.67Ч

ЗАКЛЮЧЕНИЕ

Благодаря использованию ROS удалось разработать систему сканирования пространства для мобильного робота и решить поставленные задачи, включающие построение карты, автономные навигацию и движение по местности.

Написанный код может быть использованным в любой системе с дифференциальным приводом, как только в элементы контроля низшего уровня будут внесены необходимые поправки для совместимости с новой конфигурацией.

Код, ответственный за контроль высшего уровня, изменений не требует, все, что нужно для интеграции с другими машинами – настройка параметров в .yaml файлах.

Результаты работы могут быть использованы как в образовательных, так и практических целях. Система сканирования пространства может быть доработана, например, подключена к технологии искусственного интеллекта и машинного зрения.

Исходный код написан на c++ и хранится в репозитории GitHub.

Список литературы

[1] Newman, W. S. (2017). *A Systematic Approach to Learning Robot Programming with ROS.* CRC Press.

[2] Ramkumar Gandhinathan, L. J. (2019). *ROS Robotics Projects.* Packt Publishing.

[3] Joseph, L. (2021). *Mastering ROS for Robotics Programming.* Packt Publishing.

[4] *Nodes*. (2018). Получено из Ros.org: https://wiki.ros.org/Nodes

[5] *Understanding topics*. (2024). Получено из Ros.org: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

[6] *Understanding services*. (2024). Получено из Ros.org: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

[7] *Messages*. (2016). Получено из Ros.org: https://wiki.ros.org/Messages

[8] *Parameter Server*. (2018). Получено из Ros.org: https://wiki.ros.org/Parameter%20Server

[9] *Packages*. (2019). Получено из Ros.org: https://wiki.ros.org/Packages

[10] *Catkin Workspaces*. (2017). Получено из Ros.org: https://wiki.ros.org/catkin/workspaces

[11] *rviz*. (2018). Получено из Ros.org: http://wiki.ros.org/rviz

[12] *Gazebo*. (2014). Получено из Ros.org: http://wiki.ros.org/gazebo

[13] Sachin Chitta, E. M.-E. (2017). ros_control: A generic and simple control framework for. *The Journal of Open Source Software*. Получено из http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf

[14] *usb_cam*. (2023). Получено из Ros.org: http://wiki.ros.org/usb_cam

[15] *rplidar*. (2019). Получено из Ros.org: http://wiki.ros.org/rplidar

[16] *freenect_stack*. (2013). Получено из Ros.org: https://wiki.ros.org/freenect_stack

[17] *hector_slam*. (2014). Получено из Ros.org: http://wiki.ros.org/hector_slam

[18] *gmapping*. (2019). Получено из Ros.org: http://wiki.ros.org/gmapping

[19] *rtabmap_ros*. (2023). Получено из Ros.org: http://wiki.ros.org/rtabmap_ros

[20] *rosserial*. (2018). Получено из Ros.org: https://wiki.ros.org/rosserial

[21] *ros_control*. (2023). Получено из Ros.org: http://wiki.ros.org/ros_control

[22] *move_base*. (2020). Получено из Ros.org: https://wiki.ros.org/move_base

[23] *imu_tools*. (2022). Получено из Ros.org: https://wiki.ros.org/imu_tools

[24] *teleop_twist_keyboard*. (2015). Получено из Ros.org: https://wiki.ros.org/teleop_twist_keyboard

[25] *robot_localization*. (2020). Получено из Ros.org: https://wiki.ros.org/robot_localization

[26] *robot_state_publisher*. (2020). Получено из Ros.org: https://wiki.ros.org/robot_state_publisher

[27] *joint_state_publisher*. (2022). Получено из Ros.org:

https://wiki.ros.org/joint_state_publisher
[28] *amcl*. (2020). Получено из Ros.org: https://wiki.ros.org/amcl
[29] *map_server*. (2020). Получено из Ros.org: https://wiki.ros.org/map_server
[30] *Installation*. (2023). Получено из Ros.org: https://wiki.ros.org/ROS/Installation
[31] *Installing gazebo_ros_pkgs (ROS 1)*. (2014). Получено из gazebosim.org: https://classic.gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros
[32] *urdf*. (2023). Получено из Ros.org: https://wiki.ros.org/urdf
[33] *urdf/Tutorials*. (2016). Получено из Ros.org: https://wiki.ros.org/urdf/Tutorials
[34] *xacro*. (2022). Получено из Ros.org: https://wiki.ros.org/xacro
[35] *Gazebo plugins in ROS*. (2014). Получено из gazebosim.org: https://classic.gazebosim.org/tutorials?tut=ros_gzplugins
[36] *roslaunch/XML*. (2017). Получено из ros.org: https://wiki.ros.org/roslaunch/XML
[37] *ChassiBot*. (2024). Получено из github.com: https://github.com/TopTyan/Chassi_BOT/tree/master
[38] *rosserial_arduino/Tutorials*. (2014). Получено из Ros.org: https://wiki.ros.org/rosserial_arduino/Tutorials
[39] Tsouroukdissian, A. R. (2014). *ROS control, an overview.* Получено из Ros.org: https://roscon.ros.org/2014/wp-content/uploads/2014/07/ros_control_an_overview.pdf
[40] *diff_drive_controller*. (2023). Получено из Ros.org: https://wiki.ros.org/diff_drive_controller
[41] *Create your own hardware interface*. (2020). Получено из Ros.org: https://wiki.ros.org/ros_control/Tutorials/Create%20your%20own%20hardware%20interface
[42] *joint_state_interface.h*. (2022). Получено из Ros.org: https://docs.ros.org/en/melodic/api/hardware_interface/html/c++/joint__state__interface_8h_source.html
[43] *hardware_interface::VelocityJointInterface Class Reference*. (2022). Получено из Ros.org: https://docs.ros.org/en/melodic/api/hardware_interface/html/c++/classhardware__interface_1_1VelocityJointInterface.html#details
[44] *controller_manager*. (2019). Получено из Ros.org: https://wiki.ros.org/controller_manager
[45] *diff_drive_controller/test*. (2020). Получено из github.com: https://github.com/ros-controls/ros_controllers/tree/noetic-devel/diff_drive_controller/test
[46] Сабитов Т., Б. А. (2023). Алгоритм построения путей для группы роботов, движущихся в двух измерениях в среде с неподвижными препятствиями. *Вестник КазАТК*, 268-276.
[47] Kevin M. Lynch, N. M. (2015). *Embedded computing and mechatronics*

with the PIC32 microcontroller. Newnes is an imprint of Elsevier .

НЕКОММЕРЧЕСКОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО
«КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ К.А. САТПАЕВА»

РЕЦЕНЗИЯ

дипломного проекта Сабитова Темирлана Рустамовича
по специальности 6B07111 – «Робототехника и мехатроника»
Satbayev University

Тема дипломной работы: «Разработка системы сканирования пространства для мобильного робота»

Выполнено:
а) В графическом разделе номер рисунков___
б) Объяснительная записка номер страниц

Актуальность темы исследования. Дипломная работа посвящена разработке системы сканирования пространства для мобильного робота, что является актуальной задачей в сфере робототехники и мехатроники. Проект включает создание карты рабочего пространства, ориентацию в местности, удаленное управление роботом и автономное движение

Структура работы. Дипломная работа имеет четкую структуру, включающую теоретическую и практическую части, а также раздел с расчетами. Теоретическая часть охватывает основные элементы ROS, в то время как практическая часть демонстрирует применение этих знаний при создании прототипа робота.

Оценка работы. Работа представляется качественной и содержательной. Она включает в себя подробное описание процесса разработки и тестирования системы сканирования пространства, а также экономический расчет макета проекта. Результаты работы могут быть полезны как в образовательных, так и в практических целях. Особенно ценным является универсальность программного кода, который может быть адаптирован под любую мобильную платформу.

Резензент
Проректор по НиС ALT Университет имени Мухамеджана Тынышпаева,
PhD, ассоциированный профессор
_____ Сергазин Г.К.
05 2024 г.

HR департамент
HR департамент

ПОДПИСЬ ЗАВЕРЯЮ

Ф ҚазҰТЗУ 704-22. Рецензия

НЕКОММЕРЧЕСКОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО «КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени К.И. САТПАЕВА»

ОТЗЫВ

дипломного проекта (работы)

студента специальности 6B07111 – «Робототехника и мехатроника»

Сабитова Темирлана Рустамовича

На тему: **«Разработка системы сканирования пространства для мобильного робота»**

Дипломный проект студента бакалавриата Сабитова Темирлана Рустамовича посвящен «Разработке системы сканирования пространства для мобильного робота».

Целью работы является создание системы сканирования пространства с использованием лазерных датчиков и камер, предназначенной для решения практических задач робототехники, таких как построение карты местности, локализация и навигация. В ходе выполнения дипломного проекта Сабитов Темирлан Рустамович тщательно изучил программные пакеты ROS и создал на их основе собственную конфигурацию.

Практическая часть работы включает разработку конструкции робота, выбор компонентов, сборку и программирование системы управления сканирования пространства. В результате проведенных экспериментальных исследований была построена карта местности и произведено автономное движение.

Дипломный проект выполнен на высоком уровне, соответствует общим требованиям к содержанию, оформлению и изложению материала. Работа содержит экономические расчеты и математические модели, а также иллюстративный графический материал, включающий электронные схемы, 3D-модели и программный код.

Считаю, что дипломный проект студента Сабитова Темирлана Рустамовича полностью соответствует предъявляемым требованиям, заслуживает оценки "отлично" и академической степени бакалавра.

Научный руководитель
Кандидат физико-математических наук, ассоциированный профессор
Бактыбаев М.К.
«30» 05 2024 г.

StrikePlagiarism.com SATBAYEV UNIVERSITY

Дата отчета 5/29/2024
Дата редактирования 5/30/2024

Отчет не был оценен.

Метаданные

Название
Разработка системы сканирования пространства для мобильного робота.

Автор
Сабитов Темирлан Рустамович

Научный руководитель / Эксперт
Мурат Бактыбаев

Подразделение
ИАиИТ

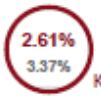
Тревога

В этом разделе вы найдете информацию, касающуюся текстовых искажений. Эти искажения в тексте могут говорить о ВОЗМОЖНЫХ манипуляциях в тексте. Искажения в тексте могут носить преднамеренный характер, но чаще, характер технических ошибок при конвертации документа и его сохранении, поэтому мы рекомендуем вам подходить к анализу этого модуля со всей долей ответственности. В случае возникновения вопросов, просим обращаться в нашу службу поддержки.

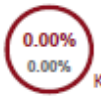
Замена букв		0
Интервалы		0
Микропробелы		0
Белые знаки		0
Парафразы (SmartMarks)		13

Объем найденных подобий

КП-ия определяют, какой процент текста по отношению к общему объему текста был найден в различных источниках.. Обратите внимание!Высокие значения коэффициентов не означают плагиат. Отчет должен быть проанализирован экспертом.



КП1



КП2

КЦ

25
Длина фразы для коэффициента подобия 2

3947
Количество слов

30759
Количество символов

Поиск контента ИИ

Интегрированный модуль поиска контента AI. Нажмите «Подробнее», чтобы узнать больше о результатах и алгоритме поиска.

Коэффициент вероятности ИИ

Подобия по списку источников

Ниже представлен список источников. В этом списке представлены источники из различных баз данных. Цвет текста означает в каком источнике он был найден. Эти источники и значения Коэффициента Подобия не отражают прямого плагиата. Необходимо открыть каждый источник и проанализировать сожержание и правильность оформления источника.

10 самых длинных фраз

Цвет текста

ПОРЯДКОВЫЙ НОМЕР	НАЗВАНИЕ И АДРЕС ИСТОЧНИКА URL (НАЗВАНИЕ БАЗЫ)	КОЛИЧЕСТВО ИДЕНТИЧНЫХ СЛОВ (ФРАГМЕНТОВ)	
1	Team Mountaineers Space Robotic Challenge Phase-2 Qualification Round Preparation Report Christopher A. Tatsch,Derek W. Ross,Jared J. Beard,Cagri Kilic,Jason N. Gross,Bernardo Martinez R. Jr;	18	0.46 %
2	http://kutenk2000.blogspot.com/2016/05/847-new-computer-books.html	12	0.30 %
3	https://nardar.ru/articles/chto-yavlyaetsya-tselyu-strukturnyh-metodov-proektirovaniya-ps	11	0.28 %
4	https://ela.kpi.ua/bitstream/123456789/43624/1/Saiapin_bakalavr.pdf	11	0.28 %
5	https://dspace.www1.vlsu.ru/bitstream/123456789/6911/1/00746.docx	7	0.18 %
6	http://web.tpu.ru/webcenter/content/conn/WebCenterSpaces-ucm/path/PersonalSpaces/FWC%20folders/year%202021/month%206/day%2001%2000-02-31/56956476.DOCX	6	0.15 %
7	https://skachatvs.com/4220320/primenenie-neyrosetey-dlya-raspoznavaniya-tekstov-i-obucheniya	6	0.15 %
8	https://nardar.ru/articles/chto-yavlyaetsya-tselyu-strukturnyh-metodov-proektirovaniya-ps	6	0.15 %
9	https://www.enu.kz/downloads/iyun-2020/instr-tvorch-examen.docx	6	0.15 %
10	https://dspace.www1.vlsu.ru/bitstream/123456789/6911/1/00746.docx	5	0.13 %

из базы данных RefBooks (0.46 %)

ПОРЯДКОВЫЙ НОМЕР	НАЗВАНИЕ	КОЛИЧЕСТВО ИДЕНТИЧНЫХ СЛОВ (ФРАГМЕНТОВ)	
Источник: https://arxiv.org/			
1	Team Mountaineers Space Robotic Challenge Phase-2 Qualification Round Preparation Report Christopher A. Tatsch,Derek W. Ross,Jared J. Beard,Cagri Kilic,Jason N. Gross,Bernardo Martinez R. Jr;	18 (1)	0.46 %

из домашней базы данных (0.00 %)

ПОРЯДКОВЫЙ НОМЕР	НАЗВАНИЕ	КОЛИЧЕСТВО ИДЕНТИЧНЫХ СЛОВ (ФРАГМЕНТОВ)

из программы обмена базами данных (0.00 %)

ПОРЯДКОВЫЙ НОМЕР	НАЗВАНИЕ	КОЛИЧЕСТВО ИДЕНТИЧНЫХ СЛОВ (ФРАГМЕНТОВ)

из интернета (2.91 %)

ПОРЯДКОВЫЙ НОМЕР	ИСТОЧНИК URL	КОЛИЧЕСТВО ИДЕНТИЧНЫХ СЛОВ (ФРАГМЕНТОВ)	
1	https://dspace.www1.vlsu.ru/bitstream/123456789/6911/1/00746.docx	27 (5)	0.68 %
2	http://web.tpu.ru/webcenter/content/conn/WebCenterSpaces-ucm/path/PersonalSpaces/FWC%20folders/year%202021/month%206/day%2001%2000-02-31/56956476.DOCX	21 (4)	0.53 %
3	https://nardar.ru/articles/chto-yavlyaetsya-tselyu-strukturnyh-metodov-proektirovaniya-ps	17 (2)	0.43 %
4	https://www.enu.kz/downloads/iyun-2020/instr-tvorch-examen.docx	16 (3)	0.41 %
5	http://kutenk2000.blogspot.com/2016/05/847-new-computer-books.html	12 (1)	0.30 %
6	https://skachatvs.com/4220320/primenenie-neyrosetey-dlya-raspoznavaniya-tekstov-i-obucheniya	11 (2)	0.28 %
7	https://ela.kpi.ua/bitstream/123456789/43624/1/Saiapin_bakalavr.pdf	11 (1)	0.28 %

Список принятых фрагментов

ПОРЯДКОВЫЙ НОМЕР	СОДЕРЖАНИЕ	КОЛИЧЕСТВО ИДЕНТИЧНЫХ СЛОВ (ФРАГМЕНТОВ)
Team Mountaineers Space Robotic Challenge Phase-...		**18 (0.46%)**
http://kutenk2000.blogspot.com/2016/05/847-new-c...		**12 (0.30%)**