МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт искусственного интеллекта (ИИИ)

Кафедра проблем управления

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1

по дисциплине «Программное обеспечение мехатронных и робототехнических систем»

Выполнили студенты группы КРБО-03-22 Филичкина М.Д.

Рабышев Д.А.

Рольнов М.Ю.

Сафиулин Д.Р.

Проверил Морозов А.А.

Москва 2025

Пакет Nav2 фреймфорка ROS2 для задач навигации и планирования движения мобильных роботов

Цель работы: изучение пакета Nav2 фреймворка ROS 2 в рамках задач навигации и построения траекторий движения мобильных роботов в детерминированном окружении в среде виртуального моделирования Gazeebo.

Ознакомление с пакетом Nav2;

ознакомление с пакетами для моделирования робота TurtleBot 3; ознакомление с основными этапами работы с пакетом Nav2.

Задание:

1. Установить и настроить пакет Nav2 для модели мобильного робота Turtlebot 3.
2. Установить и настроить группу пакетов для моделирования мобильного робота Turtlebot 3 в среде Gazeebo.
3. Осуществить картографирование виртуального полигона.
4. Осуществить управление движением робота в произвольные координаты на виртуальном полигоне.
5. Подготовить отчет.

Дополнительное задание:

1. Разработать пакет для автоматического картографирования виртуального полигона.

Ход работы

Robot operating system 2 (ROS 2) — это набор программных библиотек и инструментов для создания программного обеспечения для роботов. Одним из инструментов виртуального моделирования является Gazeebo. Пакет Nav2 является программным стеком, потому что он состоит из многих пакетов и программ, надстроенных друг над другом и позволяющие только совместно решать комплексные задачи. Позволяет осуществлять картографирование, локальное планирование пути с учетом динамических препятствий, глобальное планирование пути, хранение и загрузку карт, локализацию. Весь описанный функционал позволяет быстро развернуть тактический уровень системы управления мобильным роботом и систему навигации без необходимости полного цикла разработки этих компонентов с нуля, что значительно ускоряет разработку системы управления. Пакет Nav2 обладает открытым исходным кодом, т.е. позволяет производить его улучшения или проводить работы по исследованию улучшенных алгоритмов навигации при использовании готовых инструментов по управлению роботом со стороны пакета Nav2 и логирования со стороны ROS 2.

ROS 2 был уже установлен. Переходим к установке пакета Nav2 и пакет с моделями робота turtlebot3 для Gazeebo, далее подготовили симулятор Gazeebo и модели робота turtlebot3 (рис.1).

```
#!/bin/bash
set -e

echo "=== 🧩 Установка ROS 2 Humble, Gazebo, Nav2 и TurtleBot3 Waffle
==="

# === Обновление системы ===
sudo apt update && sudo apt upgrade -y

# === Локаль и базовые пакеты ===
sudo apt install -y locales curl gnupg lsb-release software-properties-
common
sudo locale-gen en_US en_US.UTF-8
export LANG=en_US.UTF-8
sudo add-apt-repository universe

# === Добавление ключа и репозитория ROS 2 ===
sudo curl -sSL
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/ros-archive-keyring.gpg] \
http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
$UBUNTU_CODENAME) main" \
| sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

# === Установка ROS 2 Desktop (включает RViz2) ===
sudo apt update
sudo apt install -y ros-humble-desktop

# === Настройка автозагрузки ROS 2 окружения ===
if ! grep -Fxq "source /opt/ros/humble/setup.bash" ~/.bashrc; then
    echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
fi
source /opt/ros/humble/setup.bash

# === Установка инструментов разработчика ===
sudo apt install -y python3-colcon-common-extensions python3-rosdep
python3-vcstool build-essential
sudo rosdep init || true
rosdep update

# === Установка Gazebo Fortress (через ROS интерфейс) ===
sudo apt install -y ros-humble-ros-gz

# === Установка Navigation2 ===
sudo apt install -y ros-humble-navigation2 ros-humble-nav2-bringup

# === Установка TurtleBot3 и моделей ===
sudo apt install -y ros-humble-turtlebot3*

# === Настройка модели Waffle ===
if ! grep -Fxq "export TURTLEBOT3_MODEL=waffle" ~/.bashrc; then
    echo "export TURTLEBOT3_MODEL=waffle" >> ~/.bashrc
fi
export TURTLEBOT3_MODEL=waffle

echo "=== ✅ Установка завершена! ==="
echo "Перезапусти терминал или введи: source ~/.bashrc"
echo


echo "👉 Чтобы запустить симуляцию:"
echo "   ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py"
echo
echo "👉 Чтобы запустить планирование пути (Nav2):"
echo "   ros2 launch nav2_bringup navigation_launch.py
use_sim_time:=True"
echo
echo "🚀 После этого в RViz выбери '2D Pose Estimate' и '2D Goal Pose'
для навигации."
```

Рисунок 1. Все команды для установки ROS 2, Nav2, симулятор Gazeebo и модели робота turtlebot3

В данной практической работе используется модификация Waffle (рис.2).

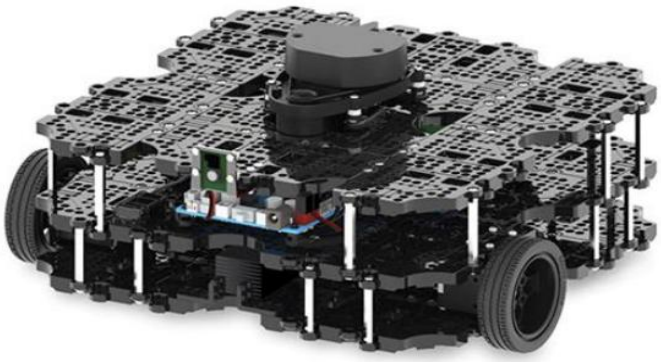

Рисунок 2. Внешний вид робота TurtleBot 3 Waffle

Тип робота задается через глобальную систему переменных:

(TURTLEBOT3_MODEL=waffle)

Вторая глобальная система переменных отвечает за устранение возможных проблем при работе с картой среды. Она переключает ROS2 с использования Fast DDS на Cyclone DDS, но дополнительно выполнили команды для установки Cyclone DDS:

sudo apt update

sudo apt install ros-humble-rmw-cyclonedds-cpp

Также внесли правки в файл с параметрами пакета turtlebot3_navigation2 и таким образом файл с параметрами принимает вид, представленный на рисунке 3.

```
amcl:
  ros__parameters:
    alpha1: 0.2
    alpha2: 0.2
    alpha3: 0.2
    alpha4: 0.2
    alpha5: 0.2
    base_frame_id: "base_footprint"
    beam_skip_distance: 0.5
    beam_skip_error_threshold: 0.9
    beam_skip_threshold: 0.3
    do_beamskip: false
    global_frame_id: "map"
    lambda_short: 0.1
    laser_likelihood_max_dist: 2.0
    laser_max_range: 100.0
    laser_min_range: -1.0
    laser_model_type: "likelihood_field"
    max_beams: 60
    max_particles: 2000
    min_particles: 500
    odom_frame_id: "odom"
    pf_err: 0.05
    pf_z: 0.99
    recovery_alpha_fast: 0.0
    recovery_alpha_slow: 0.0
    resample_interval: 1
    robot_model_type: "nav2_amcl::DifferentialMotionModel"
    save_pose_rate: 0.5
    sigma_hit: 0.2
    tf_broadcast: true
    transform_tolerance: 0.5
    update_min_a: 0.2
    update_min_d: 0.25
    z_hit: 0.5
    z_max: 0.05
    z_rand: 0.5
```

Рисунок 3. Содержание файла waffle.yaml

На этом подготовка виртуальной модели завершена и можно приступать к следующим этапам.

В данной практической работе используется виртуальная модель робота TurtleBot 3 Waffle, представляет из себя пакет для ROS 2. Она является программной моделью, что позволяет, в случае необходимости, её модифировать под конкретные текущие нужды. Рассмотрим её устройство, на рисунке 4 представлен граф топиков модели.

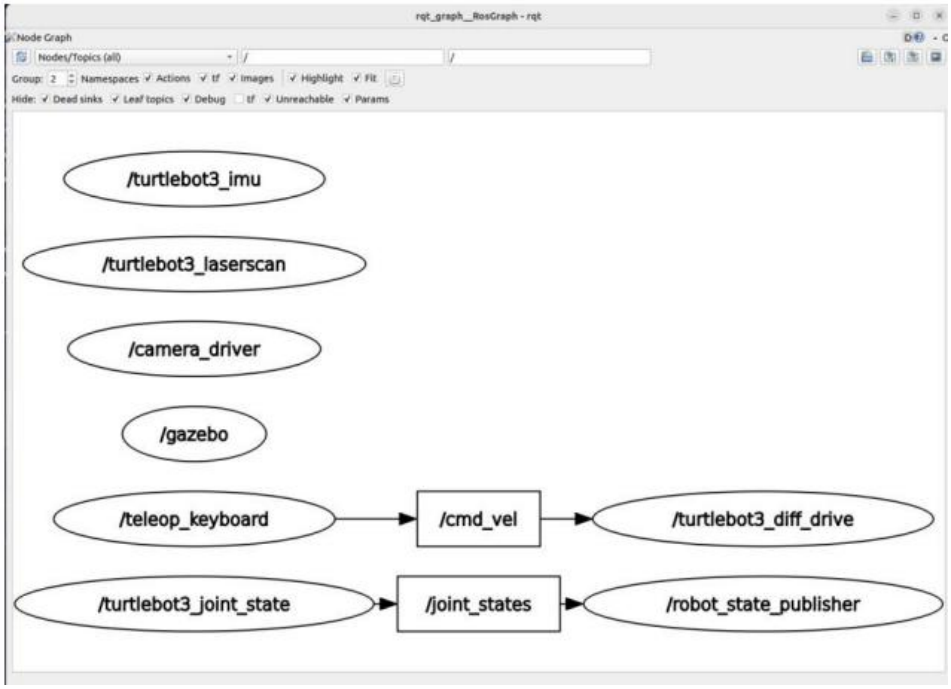

Рисунок 4. Граф узлов и топиков

Узел /turtlebot3_diff_drive управляет приводным уровнем робота, подписан на топик /cmd_vel из которого получает уставки по скорости.

Топик /cmd_vel имеет тип данных /geometry_msgs/msg/Twist, который представляет из себя структуру с полями linear и angular типа Vector3. Поле linear задает линейную скорость по трем осям OX, OY, OZ (поэтому и 8 используется тип данных Vector3, представляющий из себя массив из 3-х элементов). Другое поле - angular, задает угловую скорость относительно осей OX, OY, OZ. В рамках модели мобильного наземного робота угловые скорости относительно осей OX и OY не используются, а угловая скорость относительно оси OZ задает скорость поворот робота вокруг своей оси.

Модель с телеуправлением узел /teleop_keyboard публикует в топик /cmd_vel уставки по скорости, однако есть возможность написать собственный узел, который по некоему алгоритму управляет мобильным роботом.

Обратная связь по скорости и положению публикуется в топике /joint_states. В случае скорости симулируется работа датчиков скорости, а положение робота получается на основании одометрии. Оценка положения по одометрии будет довольно точной, поскольку используется упрощенная физическая модель и она не учитывает многие факторы, в том числе проскальзывание колес.

Карта является одним из ключевых факторов осуществления навигации робота в помещении. Запустим полигон в Gazeebo при помощи команды:

ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py (1 терминал)

В результате выполнения данной команды открывается окно Gazeebo с запущенным виртуальным полигоном и моделью робота на нём (рисунок 5).

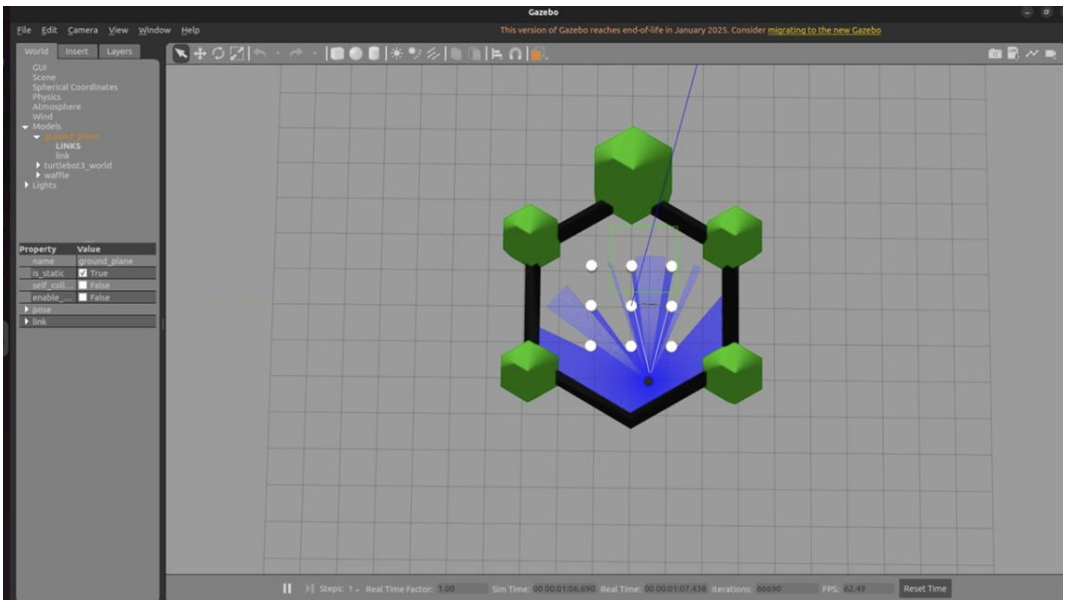

Рисунок 5. Виртуальный полигон с моделью робота TurtleBot3 Waffle

В новом окне терминала выполняем команду:

source /opt/ros/humble/setup.bash

ros2 launch turtlebot3_cartographer

cartographer.launch.py use_sim_time:=True (2 терминал)

В результате откроется Rviz 2 - инструмент отладки и визуализации данных, получаемых с датчиков роботов (рисунок 6).

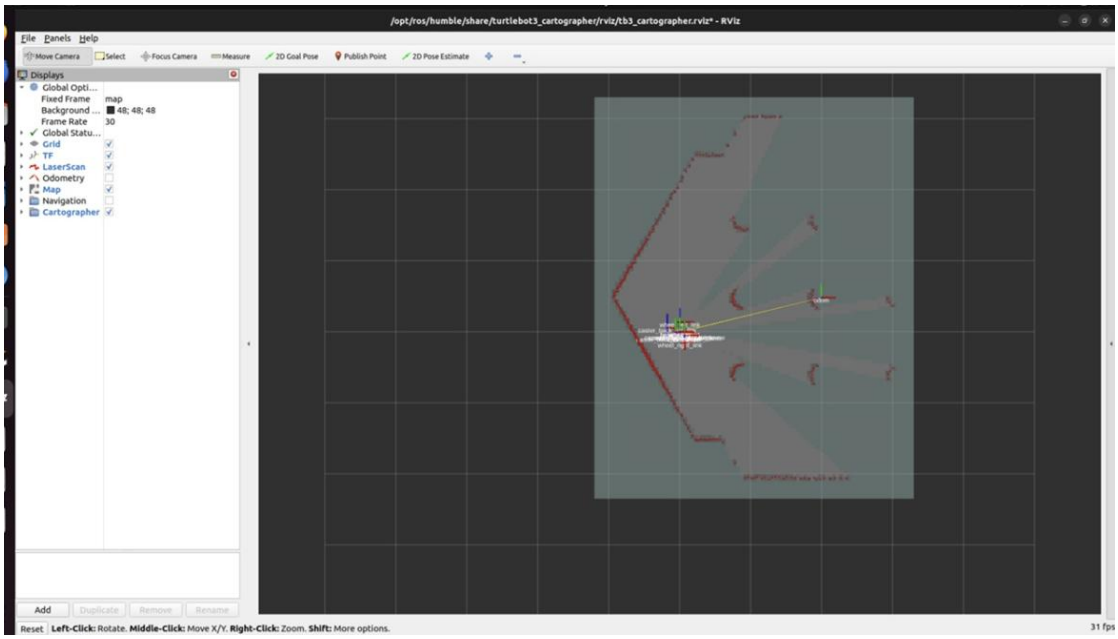

Рисунок 6. Окно Rviz2

Светло-серым цветом отмечены области, которые были внесены в карту, зелено-серым цветом отмечены незакартографированнные области, а черным цветом - препятствия.

В третьем терминале выполнили команду для запуска режима управления роботом с клавиатуры “W” (вперед), “A” (влево), “D” (вправо), “X” (назад), “S” (стоп):

source /opt/ros/humble/setup.bash

ros2 run turtlebot3_teleop teleop_keyboard (3 терминал)

Осуществляем картографирование виртуального полигона вручную с помощью режима управления роботом с клавиатуры. Робот не должен набирать слишком большую скорость, чтобы не происходили столкновения с элементами полигона. В окне RViz 2 получается картина,

представленная на рисунке 8. На рисунке 7 не до конца сформированная карта.

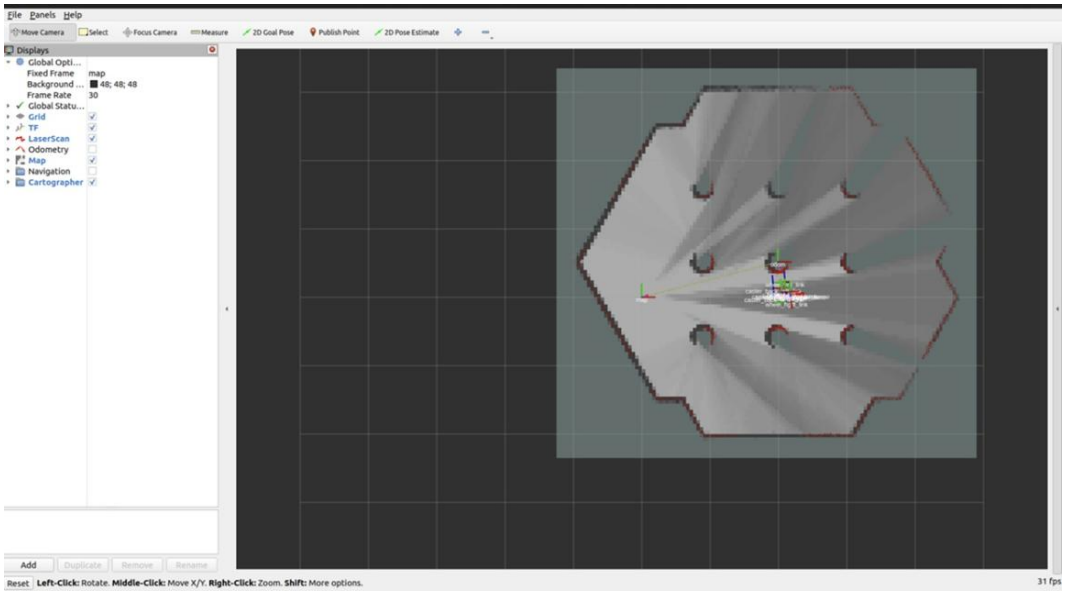

Рисунок 7. Карта сформирована не до конца

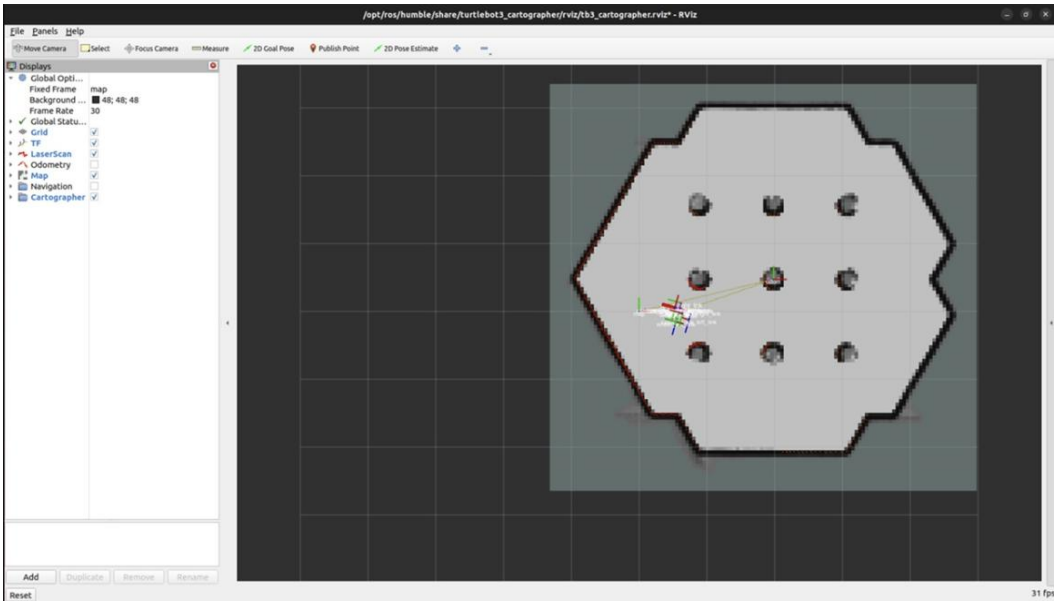

Рисунок 8. Карта сформирована полностью

Далее полученные карты сохраняем:

```
cd ~ && mkdir -p maps
ros2 run nav2_map_server map_saver_cli -f
 ~/maps/my_map
```

Чтобы открыть карту и посмотреть её используется команда:

```
eog ~/maps/my_map.pgm &
```

Таким образом в каталоге maps было создано 2 файла карты: my_map.yaml и my_map.pgm. Они будут использованы пакетом Nav2 для осуществления навигации и построения маршрута движения робота.

Переходим к этапу, где робот будет двигаться в обозначенные пользователем точки на виртуальным полигоне осуществляя глобальное и локальное планирование траектории, а также объезд препятствий.

Запустить модель робота и виртуальный полигон в Gazeebo с помощью команды:

ros2 launch turtlebot3_gazebo

turtlebot3_world.launch.py (1 терминал)

Во втором терминале находясь в домашнем каталоге пользователя запустить пакет для робота, использующий Nav2:

source /opt/ros/humble/setup.bash (2 терминал)

ros2 launch turtlebot3_navigation2

navigation2.launch.py use_sim_time:=True

map:=maps/my_map.yaml

Зададим текущее положение робота на карте. Окно Rviz 2 примет вид, показанный на рисунке 9, что говорит о том, начальная позиция была задана и произошло позиционирование робота на карте.

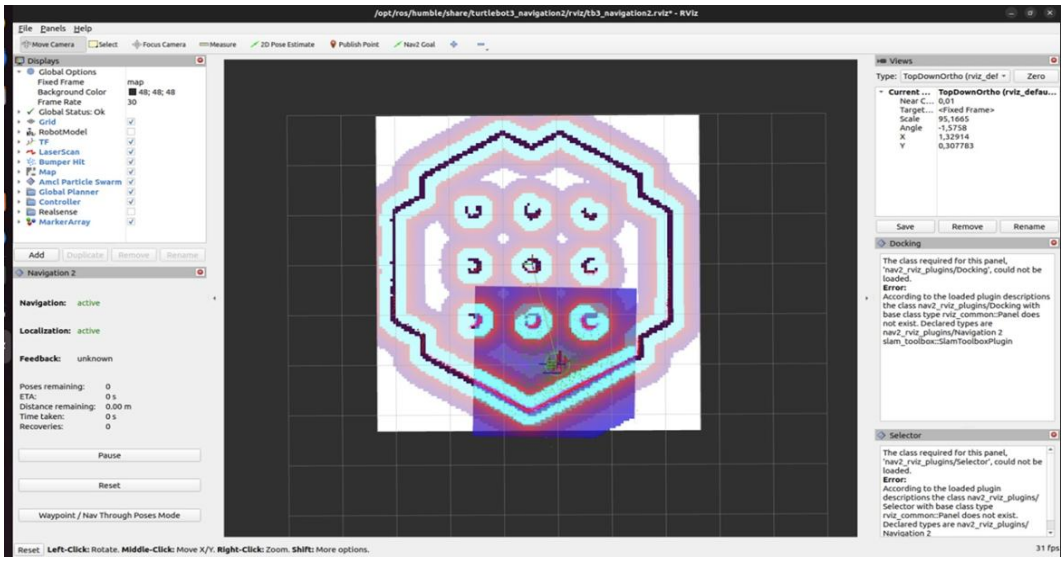

Рисунок 9. Внешний вид окна Rviz2 после указания начальной позиции робота на карте

На рисунке 10 представлено изображение пути в rviz2 и видно, что робот проехал.

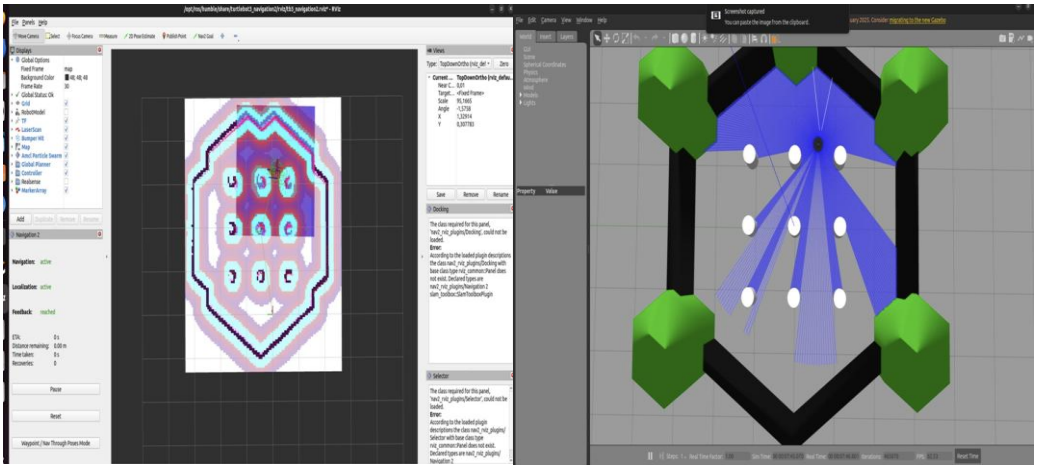

Рисунок 10. Слева путь в rviz2, справа движение робота

Дополнительное задание:

1. Разработать пакет для автоматического картографирования виртуального полигона.

Код для пакета автоматического картографирования виртуального полигона представлен в приложении А. Для запуска модели робота и виртуальный полигон в Gazeebo используются команды:

ros2 launch turtlebot3_gazebo
turtlebot3_world.launch.py (1 терминал)

Во втором терминале выполняем команду:

source /opt/ros/humble/setup.bash
ros2 launch turtlebot3_cartographer
cartographer.launch.py use_sim_time:=True (2 терминал)

В третьем терминале находясь в домашнем каталоге пользователя запустить пакет для робота, использующий Nav2:

source /opt/ros/humble/setup.bash
ros2 launch turtlebot3_navigation2
navigation2.launch.py use_sim_time:=True
map:=maps/my_map.yaml (3 терминал)

В четвертом терминале команда запускает программу на Python, где показывает карту и перемещение робота в реальном времени:

python3 ~/slam_explorer.py (4 терминал)

Результат дополнительного задания изображен на рисунке 11.

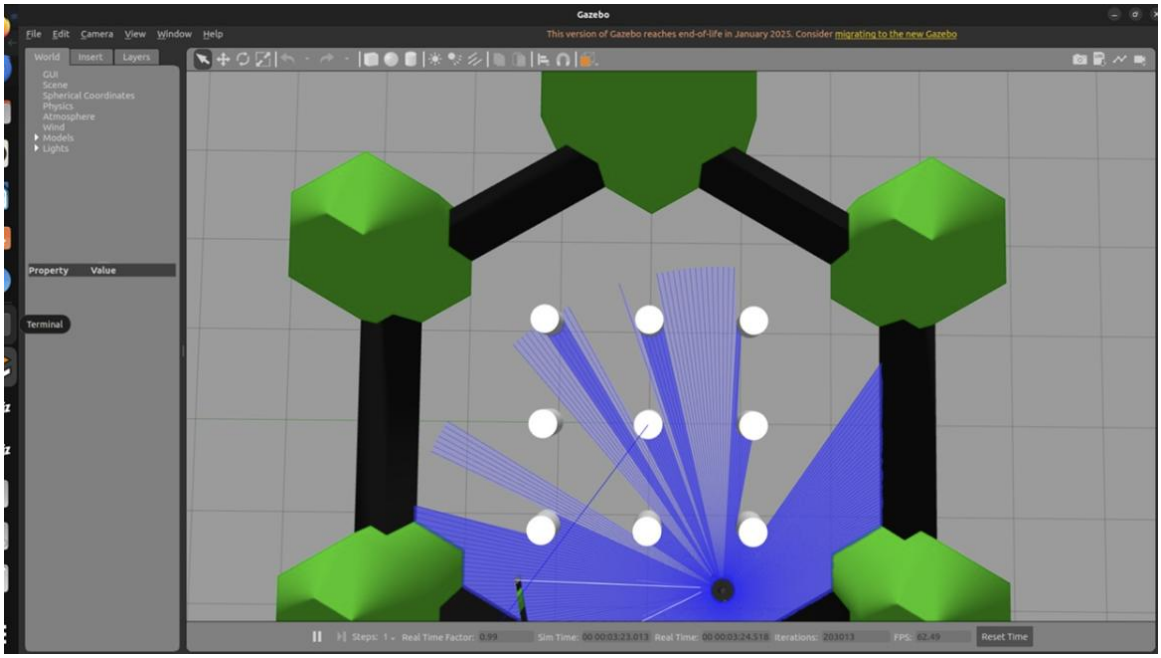

Рисунок 11. Реализация дополнительного задания

Вывод: в ходе выполнения практической работы был успешно изучен пакет Nav2 фреймворка ROS 2, предназначенный для решения задач навигации и планирования движения мобильных роботов. Установлены и настроены необходимые пакеты для работы с моделью робота TurtleBot 3 в среде виртуального моделирования Gazebo. Также освоены навыки работы с пакетом Nav2, включая его структуру и взаимодействие с другими компонентами ROS 2, проведено картографирования виртуального полигона и сохранение полученной карты. Реализовано управление движением робота по заданным координатам с использованием возможностей глобального и локального планирования траектории, а также обхода препятствий. Выполнено дополнительное задание: разработан пакет для автоматического исследования и картографирования полигона.

Приложение А

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import math
class SmartSafeMapper(Node):
    def __init__(self):
        super().__init__('smart_safe_mapper')
        self.cmd_vel_pub = self.create_publisher(Twist, '/cmd_vel', 10)
        self.lidar_sub = self.create_subscription(
            LaserScan, '/scan', self.lidar_callback, 10)
        self.timer = self.create_timer(0.1, self.control_loop)
        self.lidar_data = None
        # БЕЗОПАСНЫЕ ПАРАМЕТРЫ
        self.safe_distance = 0.7  # метр до стены - начинаем поворачивать
        self.min_distance = 0.3   # абсолютный минимум - стоп и задний ход
        self.forward_speed = 0.12 # медленная скорость
        self.turn_speed = 0.8     # скорость поворота
        self.backup_speed = -0.1  # скорость заднего хода
        self.backup_time = 0.0    # время движения
        self.state = "EXPLORING"  # EXPLORING, BACKING_UP, TURNING
    def lidar_callback(self, msg):
        self.lidar_data = msg
    def get_safe_directions(self):
        if self.lidar_data is None:
            return None
        ranges = list(self.lidar_data.ranges)
        num_ranges = len(ranges)
        # ДЛЯ ЛИДАРА:
        # 0° - спереди, 90° - слева, 180° - сзади, 270° - справа
        # Спереди: -30° до +30° (330°-30°)
        front_indices = list(range(0, 30)) + list(range(330, 360))
```

```
        # Слева: 60° до 120° (левый бок)
        left_indices = list(range(60, 120))
        # Справа: 240° до 300° (правый бок)
        right_indices = list(range(240, 300))
        # Корректируем индексы для длины массива
        front_indices = [i % num_ranges for i in front_indices]
        left_indices = [i % num_ranges for i in left_indices]
        right_indices = [i % num_ranges for i in right_indices]
        def get_sector_min(indices):
            valid_ranges = []
            for idx in indices:
                if idx < len(ranges):
                    r = ranges[idx]
                    if not math.isnan(r) and r > 0.1 and r < self.lidar_data.range_max:
                        valid_ranges.append(r)
            return min(valid_ranges) if valid_ranges else self.lidar_data.range_max
        front_min = get_sector_min(front_indices)
        left_min = get_sector_min(left_indices)
        right_min = get_sector_min(right_indices)
        # Если разница меньше 0.3м - считаем что пространство одинаковое
        left_better = (left_min - right_min) > 0.3
        right_better = (right_min - left_min) > 0.3
        return {
            'front': front_min,
            'left': left_min,
            'right': right_min,
            'left_better': left_better,
            'right_better': right_better,
            'equal': not left_better and not right_better
        }
    def control_loop(self):
        if self.lidar_data is None:
            return
        sectors = self.get_safe_directions()
        if sectors is None:
```

```
            return
        cmd_vel = Twist()
        front_distance = sectors['front']
        left_distance = sectors['left']
        right_distance = sectors['right']
        self.get_logger().info(f'State: {self.state} | Front: {front_distance:.2f}, Left:
{left_distance:.2f}, Right: {right_distance:.2f}')
        # АВТОМАТ КОНЕЧНЫХ СОСТОЯНИЙ
        if self.state == "EXPLORING":
            # ОПАСНО БЛИЗКО - начинаем задний ход
            if front_distance < self.min_distance:
                self.state = "BACKING_UP"
                self.backup_time = 0.0
                cmd_vel.linear.x = self.backup_speed
                cmd_vel.angular.z = 0.0
                self.get_logger().warning('TOO CLOSE! BACKING UP')
            elif front_distance < self.safe_distance:
    cmd_vel.linear.x = self.forward_speed * 0.3
                if sectors['left_better']:
                    cmd_vel.angular.z = self.turn_speed  # поворот НАЛЕВО
                    self.get_logger().info('Turning LEFT - more space on left')
                elif sectors['right_better']:
                    cmd_vel.angular.z = -self.turn_speed  # поворот НАПРАВО
                    self.get_logger().info('Turning RIGHT - more space on right')
                else:
                    # Если пространство одинаковое - поворачиваем в случайную сторону
                    import random
                    cmd_vel.angular.z = self.turn_speed if random.random() > 0.5 else -
self.turn_speed
                    self.get_logger().info('Turning RANDOM - equal space')
            # СВОБОДНО - едем прямо
            else:
                cmd_vel.linear.x = self.forward_speed
                cmd_vel.angular.z = 0.0
```

```
            elif self.state == "BACKING_UP":
                # Двигаемся назад 1.5 секунды если все поломалось
                self.backup_time += 0.1
                cmd_vel.linear.x = self.backup_speed
                cmd_vel.angular.z = 0.0
                if self.backup_time >= 1.5:  # 1.5 секунды назад
                    self.state = "TURNING"
                    self.get_logger().info('Backup complete, starting turn')
            elif self.state == "TURNING":
                # Поворачиваемся на месте 2 секунды
                self.backup_time += 0.1
                cmd_vel.linear.x = 0.0
                if sectors['left_better']:
                    cmd_vel.angular.z = self.turn_speed
                elif sectors['right_better']:
                    cmd_vel.angular.z = -self.turn_speed
                else:
                    import random
                    cmd_vel.angular.z = self.turn_speed if random.random() > 0.5 else -
self.turn_speed
                if self.backup_time >= 3.5:  # 2 секунды поворота (1.5 + 2.0)
                    self.state = "EXPLORING"
                    self.get_logger().info('Turn complete, resuming exploration')
            self.cmd_vel_pub.publish(cmd_vel)
        def main(args=None):
            rclpy.init(args=args)
            node = SmartSafeMapper()
            try:
                rclpy.spin(node)
            except KeyboardInterrupt:
                # Останавливаем робота
                stop_msg = Twist()
                node.cmd_vel_pub.publish(stop_msg)
                node.get_logger().info('Mapper stopped safely')
            finally:
```

```
        node.destroy_node()
        rclpy.shutdown()
if __name__ == '__main__':
    main()
```