

MSU ROVER TEAM

ОТКРЫТЫЕ БИБЛИОТЕКИ ДЛЯ АВТОНОМНОЙ НАВИГАЦИИ ШЕСТИКОЛЕСНОГО ПРОТОТИПА МАРСОХОДА (РОВЕРА) ПО УМЕРЕННО ПЕРЕСЕЧЁННОЙ НЕЗНАКОМОЙ МЕСТНОСТИ С ВИЗУАЛЬНЫМ РАСПОЗНАВАНИЕМ ЦЕЛИ НАВИГАЦИИ.

СОГЛАСОВАНО:

Научный эксперт проекта, к.ф.-м.н.

 В.М. Буданов

05.12.2025

УТВЕРЖДАЮ:

Руководитель проекта

 А.А. Смирнов

05.12.2025

**Открытая библиотека автономной навигации по распознанным указателям
движения.**


Руководство программиста.

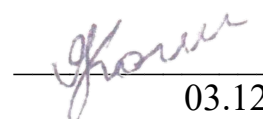
ЛИСТ УТВЕРЖДЕНИЯ

MSUROVERTEAM-SLAM-V1.0.0

(открытая библиотека в сети Интернет)

ИСПОЛНИТЕЛИ:

 В.К. Егоров
03.12.2025

 Я.А. Коломиец
03.12.2025

MSU ROVER TEAM

УТВЕРЖДЕНО

**ОТКРЫТЫЕ БИБЛИОТЕКИ ДЛЯ АВТОНОМНОЙ НАВИГАЦИИ
ШЕСТИКОЛЕСНОГО ПРОТОТИПА МАРСОХОДА (РОВЕРА) ПО УМЕРЕННО
ПЕРЕСЕЧЁННОЙ НЕЗНАКОМОЙ МЕСТНОСТИ С ВИЗУАЛЬНЫМ
РАСПОЗНАВАНИЕМ ЦЕЛИ НАВИГАЦИИ.**

**Открытая библиотека автономной навигации по распознанным указателям
движения.**

Руководство программиста.

MSUROVERTEAM-SLAM-V1.0.0

(открытая библиотека в сети Интернет)

Листов 18.

АННОТАЦИЯ.

Открытая библиотека автономной навигации по распознанным указателям движения разработана в рамках проекта «Разработки открытых библиотек для автономной навигации шестиколесного прототипа марсохода (ровера) по умеренно пересечённой незнакомой местности с визуальным распознаванием цели навигации». Проект выполнен на средства выделенные «Фондом содействия развитию малых форм предприятий в научно-технической сфере» (Фонд содействия инновациям) по договору предоставления гранта № 64ГУКодИИС13-D7/102402 от 23 декабря 2024г.

Под полностью автономным режимом навигации (движения) в данном проекте понимается режим, при котором ровер с Аккермановой геометрией поворота самостоятельно, без команд оператора (человека), передвигается по умеренно пересечённой и незнакомой местности по указателям направления движения до указателя конечной цели, может выполнить заранее запрограммированные действия у каждого указателя и самостоятельно вернуться обратно к месту старта. При этом оператор может просматривать на своем мониторе видеоизображения и телеметрию, передаваемые с ровера. «Открытая библиотека автономной навигации по распознанным указателям движения» включает в себя 2D локализацию, картирование, построение маршрута с возможностью работы без использования глобальных систем спутникового позиционирования и управление движением ровера с помощью алгоритмов SLAM (Simultaneous Localization and Mapping) в реальных условиях по умеренно пересеченной местности.

«Открытая библиотека автономной навигации по распознанным указателям движения» разработана на языке программирования C++, для платформы ROS2 Humble (Robot Operating System 2 версии Humble).

СОДЕРЖАНИЕ.

АННОТАЦИЯ.....	2
СОДЕРЖАНИЕ.	3
Общие сведения о программе.	4
Структура программы.	5
1. КЛАСС Localization.	5
2. КЛАСС Calculate_localization.....	6
3. ФУНКЦИЯ get_location().	6
4. КЛАСС Mapping.....	7
5. КЛАСС Construction_map.	7
6. ФУНКЦИЯ get_map().	8
7. КЛАСС Navigation.....	8
8. ФУНКЦИЯ move_to().	8
9. ФУНКЦИЯ setup().....	9
Интеграция с модулем управления движением робота на низком уровне.	10
Настройка ROS2 Humble для работы библиотеки.	10
Примеры использования библиотеки.	11
Пример программного кода для автономного движения робота по указателям движения (стрелкам) и по указателю конечной цели (конусу).	11
Пошаговая инструкция запуска примера.	12
Альтернативный пример программного кода для автономного движения робота до целевых точек с указанием целевой позиции.	14
Пошаговая инструкция запуска альтернативного примера.	15

Общие сведения о программе.

«Открытая библиотека автономной навигации по распознанным указателям движения» предназначена для автоматического определения текущей позиции робота (2D локализация), для построения карты местности в режиме реального времени (картирование), для построения маршрута робота с учетом рельефа местности и возможностью работы без использования глобальных систем спутникового позиционирования (навигации), а также для управления движением робота с помощью алгоритмов SLAM (Simultaneous Localization and Mapping) в реальных условиях по умеренно пересеченной местности. Метод одновременной навигации, построения карты и движения увязывает независимые процессы в непрерывный цикл последовательных вычислений, при этом результаты одного процесса участвуют в вычислениях другого процесса. Это позволяет добиться полной автономности в движении робота по незнакомой местности.

МИНИМАЛЬНЫЕ ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ:

- Операционная система: Ubuntu 22.04.6,
- Процессор: AMD Ryzen R7 6800U,
- ОЗУ: 16Гб,
- Объем диска: 16 Гб,
- Наличие камеры глубины,
- Наличие IMU WITMOTION WT901BLECL BLE5.0,
- Наличие колесной базы с энкодерами на моторах.

ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ.

Платформа: ROS2 Humble (Robot Operating System 2 версии Humble).

Язык программирования: C++.

Зависимости:

- robot_localization,
- Nav2,
- Rtabmap,
- BehaviorTree.CPP,
- BehaviorTree.ROS2,
- ROS2 (интеграция с роботом).

СПИСОК ОБЪЕКТОВ ДЛЯ ДОКУМЕНТИРОВАНИЯ.

1. Класс Localization - структура данных о положении робота.
2. Класс Calculate_localization - класс получения данных локализации.
3. Функция get_location() - получение данных о положении робота в пространстве.
4. Класс Mapping - структура данных с картой местности.
5. Класс Construction_map - класс построения карты местности.
6. Функция get_map - функция получения карты местности, построенной роботом при помощи стереокамеры.
7. Класс Navigation - класс выстраивания пайплайна действий для автономной езды робота до указанной точки.
8. Функция move_to() - функция, запускающая автономную навигацию робота до точки.
9. Функция setup() - функция для запуска навигации по стрелкам для робота.

Структура программы.

Модуль: eureka_nav_lib.hpp

Публичные объекты библиотеки:

1. Класс Localization (структура данных),
2. Класс Calculate_localization (класс получения данных локализации),
3. Функция get_location() (получение данных о положении робота в пространстве),
4. Класс Mapping (структура данных),
5. Класс Construction_map (класс построения карты местности),
6. Функция get_map (функция получения карты местности, построенной роботом при помощи стереокамеры),
7. Класс Navigation (класс выстраивания пайплайна действий для автономной езды робота до указанной точки),
8. Функция move_to() (функция для запуска автономной навигации до точки),
9. Функция setup() (функция для запуска автономной навигации по стрелкам/конусу для робота).

1. КЛАСС Localization.

Описание: структура данных для хранения данных о местоположении робота в пространстве.

Тип: dataclass.

Поля (атрибуты).

- sec: int32
Временная метка сообщения в секундах
- nanosec: int32
Временная метка сообщения в наносекундах
- position: mas[float64, float64, float64]
Позиция робота в пространстве в формате (x, y, z)

- orientation: mas[float64, float64, float64, float64]
Ориентация ровера в пространстве в формате кватерниона (x, y, z, w)
- position_v: mas[float64, float64, float64]
Линейная скорость ровера в формате (x, y, z)
- orientation_v: mas[float64, float64, float64]
Линейная скорость ровера в формате (x, y, z)

Назначение: передача информации с модуля локализации ровера, для получения данных о положении ровера в пространстве, а также скорости ровера в режиме реального времени.

2. КЛАСС `Calculate_localization`.

Описание: основной класс для получения данных о локализации ровера.

Публичные методы:

- `get_location()` -> List[Localization]
Назначение: запрос на получение всех данных с модуля локализации.
Входные параметры: нет.
Возвращаемое значение: список переменных, для получения данных положения, ориентации, линейных и угловых скоростей ровера с указанием промежутка времени.

3. ФУНКЦИЯ `get_location()`.

- Полное имя: `Calculate_localization.get_location()`
- Назначение: запрос на получение всех данных с модуля локализации.
- Входные данные: нет.
- Выходные данные:
 - список результатов локализации (List[Localization]),
 - каждый результат содержит:
 - временную метку в секундах и наносекундах;
 - позицию ровера в пространстве;
 - ориентацию ровера в пространстве;
 - линейную и угловую скорость ровера.
- Алгоритм:
 - получение данных с IMU и колесной одометрии;
 - фильтрация и фьюз данных с применение EKF (расширенного фильтра Калмана);
 - запрос данных из топики фильтрованной одометрии.
- Применение: функция для получения финальной отфильтрованной одометрии.

4. КЛАСС Mapping.

Описание: структура данных для хранения карты построенной ровером с помощью стереокамеры.

Тип: dataclass.

Поля (атрибуты).

- sec: int32
Временная метка сообщения в секундах
- nanosec: int32
Временная метка сообщения в наносекундах
- resolution: float32
Размер ячейки в метрах
- width: uint32
Размер ширины карты в ячейках
- height: uint32
Размер высоты карты в ячейках
- position: mas[float64, float64, float64]
Позиция левого нижнего угла карты в пространстве в формате (x, y, z)
- orientation: mas[float64, float64, float64, float64]
Ориентация карты в пространстве в формате кватерниона (x, y, z, w)
- data: mas[int8...]
Значения ячеек карты в диапазоне (-1 - 100), где:
 - 0 - свободное пространство;
 - 100 – препятствие;
 - -1 - неизвестное пространство;
 - 1-99 - близость к препятствию.

Назначение: хранение карты построенной ровером с препятствиями.

5. КЛАСС Construction_map.

Описание: основной класс для получения карты с препятствиями, построенной ровером.

Публичные методы:

- get_map() -> List[Mapping]
Назначение: Запрос на получение всех данных с модуля SLAM.
Входные параметры: нет.
Возвращаемое значение: список переменных, для получения карты с препятствиями в пространстве с указанием промежутка времени.

6. ФУНКЦИЯ `get_map()`.

- Полное имя: `Construction_map.get_map()`
- Назначение: запрос на получение всех данных с модуля SLAM.
- Входные данные: нет.
- Выходные данные:
 - список результатов построения карты препятствий (`List[Mapping]`);
 - каждый результат содержит:
 - временную метку в секундах и наносекундах,
 - размер ячейки карты,
 - размер карты по высоте и ширине в ячейках,
 - позицию карты в пространстве,
 - значения каждой ячейки карты, дающую информацию о препятствиях.
- Алгоритм:
 - получение данных о положении робота в пространстве;
 - получение изображения и данных глубины изображения со стереокамеры;
 - применение алгоритма RtabMap, для построения 3д карты пространства;
 - перевод карты в 2D в виде карты стоимости.
- Применение: функция для получения финальной карты, позволяющая получать информацию о ландшафте местности.

7. КЛАСС `Navigation`.

Описание: основной класс запуска навигации робота, с возможностью задачи точки или запуска движения по стрелкам.

Публичные методы:

- `move_to(coordinates: x, y, z)`
Назначение: запуск навигации для достижения роботом, указанной точки.
Входные параметры: `coordinates` (координаты целевой точки в пространстве `x`, `y` и углу по `z`).
- `setup()`
Запуск навигации по стрелкам с применением детекции стрелок и алгоритмов навигации.

8. ФУНКЦИЯ `move_to()`.

- Полное имя: `Navigation.move_to()`
- Назначение: функция запуска навигации до целевой точки с указанием целевой позиции.
- Входные данные: `coordinates` (координаты целевой точки в формате координат `x` и `y` и угла по `z`).

- Входные данные: нет.
- Алгоритм:
 - получение данных локализации;
 - получение данных о целевой позиции;
 - построение глобальной траектории до целевой точки;
 - построение карты местности;
 - генерация вектора скоростей, состоящих из двух линейных по осям X и Y и одной угловой по оси Z , контроллером для управления ровером.
- Применение: функция запускает автономную навигацию до целевой точки с указанием целевой позиции.

9. ФУНКЦИЯ `setup()`.

- Полное имя: `Navigation.setup()`
- Назначение: функция запуска навигации по стрелкам с применением дерева поведения и генерацией целевых точек.
- Входные данные: нет.
- Алгоритм:
 - запуск дерева поведения передачей переменной инициализации в топик `/init`;
 - запуск алгоритма распознавания стрелок и конуса;
 - детекция стрелки на изображении;
 - расчет параметров для расчета целевой точки ровера;
 - получение данных локализации;
 - получение данных о целевой позиции;
 - построение глобальной траектории до целевой точки;
 - построение карты местности;
 - остановка движения ровера вблизи стрелки на 10 сек.003В
 - детекция конуса и остановка движения ровера вблизи конуса.
 - генерация вектора скоростей, состоящих из двух линейных по осям X и Y и одной угловой по оси Z , контроллером для управления ровером.
- Применение: функция запускает автономную навигацию по указателям движения (стрелкам) и по указателю конечной цели (конусу).

Интеграция с модулем управления движением робота на низком уровне.

«Библиотека автономной навигации по распознанным указателям движения» интегрирована с модулем движения робота на низком уровне («Библиотека управления движением роботом на низком уровне»). При движении робота, в режиме реального времени, две данные библиотеки обмениваются следующей информацией.

- Входные данные, получаемые из модуля движения робота на низком уровне:
 - `wheel_states` - угловые скорости колёс для расчета колесной одометрии.
- Выходные данные, передаваемые в модуль движения робота на низком уровне:
 - `cmd_vel` - линейные скорости по осям X и Y и угловая скорость робота по оси Z в формате `[vel_x, vel_y, ang_z]`.

Настройка ROS2 Humble для работы библиотеки.

Для стабильной работы «Библиотеки автономной навигации по распознанным указателям движения» требуется выполнить следующие настройки пакетов навигации ROS2 Humble.

- `eureka_navigation: nav2.yaml`
- `eureka_odometry: odometry.yaml`
- `eureka_localization: ekf_el_classico.yaml`

Перед запуском автономной навигации по распознанным указателям движения необходимо выполнить глобальный launch-файл, используя команду:

```
ros2 launch eureka_navigation nav2tune.launch.py
```

Примеры использования библиотеки.

Пример программного кода для автономного движения робота по указателям движения (стрелкам) и по указателю конечной цели (конусу).

C++ код:

```
#include "eureka_nav_lib/eureka_nav_lib.hpp"
#include <rclcpp/rclcpp.hpp>

class SetupNode : public rclcpp::Node
{
public:
    SetupNode() : Node("setup_node")
    {
        /* Инициализируем общую навигационную систему
        */
        nav = std::make_shared<eureka::Navigation>(shared_from_this());

        /* Запускаем функцию движения по стрелкам
        * с остановкой в 10 секунд у каждой стрелки
        * и завершением движения возле конуса.
        */
        nav->setup();

        /* В данном примере просто выводим на консоль сообщение
        * о завершении миссии.
        */
        RCLCPP_INFO(this->get_logger(), "Navigation system setup completed");
    }

private:
    std::shared_ptr<eureka::Navigation> nav;
};

int main(int argc, char** argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<SetupNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

Пошаговая инструкция запуска примера.

1. Открыть сессию в терминале и зайти в папку ros2_ws (рабочее пространство)
2. Запустите launch-файл общего стека навигации введя команду:
`ros2 launch eureka_navigation nav2tune.launch.py`
3. Откройте новую сессию терминала и запустите launch-файл дерева поведения введя команду:
`ros2 launch eureka_bt strategy.launch.py`
4. Для запуска примера вам также требуется создать пакет, для этого введите команду:
`ros2 pkg create --build-type ament_cmake example`
5. Найдите в пакете файл CMakeLists.txt и введите следующее:

```
cmake_minimum_required(VERSION 3.16)
project(example LANGUAGES CXX)

if(CMAKE_CXX_COMPILER_ID MATCHES "(GNU|Clang)")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
set(THIS_PACKAGE_INCLUDE_DEPENDS
  eureka_nav_lib
  rclcpp
  rcpputils
)

foreach(Dependency IN ITEMS ${THIS_PACKAGE_INCLUDE_DEPENDS})
  find_package(${Dependency} REQUIRED)
endforeach()

include_directories(include/)

add_executable(${PROJECT_NAME}
  src/example.cpp
)

ament_target_dependencies(${PROJECT_NAME}
  eureka_nav_lib
  rclcpp
  rcpputils
)

# INSTALL
install(TARGETS
  ${PROJECT_NAME}
  DESTINATION lib/${PROJECT_NAME},
  DESTINATION lauch/${PROJECT_NAME})

ament_package()
```

6. Зайдите в папку src пакета для запуска примера и создайте файл example.cpp введя команду:

```
touch example.cpp
```

7. Вставьте код ниже в файл example.cpp:

```
#include "eureka_nav_lib/eureka_nav_lib.hpp"
#include <rclcpp/rclcpp.hpp>
class SetupNode : public rclcpp::Node
{
public:
    SetupNode() : Node("setup_node")
    {
        /* Инициализируем общую навигационную систему
        */
        nav = std::make_shared<eureka::Navigation>(shared_from_this());

        /* Запускаем функцию движения по стрелкам
        * с остановкой в 10 секунд у каждой стрелки
        * и завершением движения возле конуса.
        */
        nav->setup();

        /* В данном примере просто выводим на консоль сообщение
        * о завершении миссии.
        */
        RCLCPP_INFO(this->get_logger(), "Navigation system setup completed");
    }
private:
    std::shared_ptr<eureka::Navigation> nav;
};
int main(int argc, char** argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<SetupNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

8. Вернитесь в директорию ros2_ws и соберите пакет введя команду:

```
colcon build --packages-select example
```

9. Далее введите команду `source install/setup.bash` для обновления файлов вашего рабочего пространства

10. Запустите пакет с примером введя команду:

```
ros2 run example example
```

Альтернативный пример программного кода для автономного движения ровера до целевых точек с указанием целевой позиции.

C++ код:

```
#include "eureka_nav_lib/eureka_nav_lib.hpp"
#include <rclcpp/rclcpp.hpp>

/* Пример реализации автономной навигации
 * до целевой точки с указанием целевой позиции.
 * Класс SimpleNavigation создан только для данного примера.
 */
class SimpleNavigation : public rclcpp::Node
{
public:
    SimpleNavigation() : Node("simple_navigation")
    {
        /* Инициализируем значения классов библиотеки автономной навигации
         * по распознанным указателям движения.
         */
        nav = std::make_shared<eureka::Navigation>(shared_from_this());
        loc = std::make_shared<eureka::Calculate_localization>(shared_from_this());
        map = std::make_shared<eureka::Construction_map>(shared_from_this());

        /* Определяем последовательность прохода ровером целевых точек.
         */
        timer_ = this->create_wall_timer(
            std::chrono::seconds(45), // Время (в секундах),
                                     // необходимое для движения ровера
                                     // между целевыми точками.
                                     // Значение 45 сек. определено для примера!!!
                                     // В реальных условиях, данное значение может
                                     // быть рассчитано по расстоянию
                                     // и скорости движения ровера.

            [this]() {
                static int goal_num = 0;
                send_goal(goal_num);
                goal_num = (goal_num + 1) % 4;
            });
    }

private:
    std::shared_ptr<eureka::Navigation> nav;
    std::shared_ptr<eureka::Calculate_localization> loc;
    std::shared_ptr<eureka::Construction_map> map;
    rclcpp::TimerBase::SharedPtr timer_;

    /* Пример указания целевой позиции для 4-х целевых точек,
     * до которых ровер будет двигаться автономно.
     * Точек на маршруте движения ровера может быть любое кол-во.
     */
};
```

```

*/
void send_goal(int goal_id)
{
    double goals[4][3] = {
        {1.0, 0.0, 0.0},
        {2.0, 1.0, 1.57},
        {1.0, 2.0, 3.14},
        {0.0, 1.0, -1.57}
    };

    /* Вызываем функцию запуска навигации до целевой точки
    * с указанием целевой позиции.
    */
    nav->move_to(goals[goal_id][0], goals[goal_id][1], goals[goal_id][2]);
}

};

int main(int argc, char** argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<SimpleNavigation>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}

```

Пошаговая инструкция запуска альтернативного примера.

1. Открыть сессию в терминале и зайти в папку ros2_ws (рабочее пространство)
2. Запустите launch-файл общего стека навигации введя команду:
3. Для запуска примера вам также требуется создать пакет, для этого введите команду:
4. Найдите в пакете файл CMakeLists.txt и введите следующее:

```

ros2 launch eureka_navigation nav2tune.launch.py

ros2 pkg create --build-type ament_cmake example

cmake_minimum_required(VERSION 3.16)
project(example LANGUAGES CXX)

if(CMAKE_CXX_COMPILER_ID MATCHES "(GNU|Clang)")
    add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
set(THIS_PACKAGE_INCLUDE_DEPENDS
    eureka_nav_lib
    rclcpp
    rcpputils
)

foreach(Dependency IN ITEMS ${THIS_PACKAGE_INCLUDE_DEPENDS})

```



```

    find_package(${Dependency} REQUIRED)
endforeach()

include_directories(include/)

add_executable(${PROJECT_NAME}
    src/example.cpp
)

ament_target_dependencies(${PROJECT_NAME}
    eureka_nav_lib
    rclcpp
    rcpputils
)

# INSTALL
install(TARGETS
    ${PROJECT_NAME}
    DESTINATION lib/${PROJECT_NAME},
    DESTINATION launch/${PROJECT_NAME})

ament_package()

```

5. Зайдите в папку src пакета для запуска примера и создайте файл variant.cpp введя команду:
`touch variant.cpp`
6. Вставьте код ниже в файл variant.cpp:

```

#include "eureka_nav_lib/eureka_nav_lib.hpp"
#include <rclcpp/rclcpp.hpp>

/* Пример реализации автономной навигации
 * до целевой точки с указанием целевой позиции.
 * Класс SimpleNavigation создан только для данного примера.
 */
class SimpleNavigation : public rclcpp::Node
{
public:
    SimpleNavigation() : Node("simple_navigation")
    {
        /* Инициализируем значения классов библиотеки автономной навигации
         * по распознанным указателям движения.
         */
        nav = std::make_shared<eureka::Navigation>(shared_from_this());
        loc = std::make_shared<eureka::Calculate_localization>(shared_from_this());
        map = std::make_shared<eureka::Construction_map>(shared_from_this());

        /* Определяем последовательность прохода ровером целевых точек.
         */
        timer_ = this->create_wall_timer(

```

```
std::chrono::seconds(45),      // Время (в секундах),
                                // необходимое для движения ровера
                                // между целевыми точками.
                                // Значение 45 сек. определено для примера!!!
                                // В реальных условиях, данное значение может
                                // быть рассчитано по расстоянию
                                // и скорости движения ровера.

    [this]() {
        static int goal_num = 0;
        send_goal(goal_num);
        goal_num = (goal_num + 1) % 4;
    });
}

private:
    std::shared_ptr<eureka::Navigation> nav;
    std::shared_ptr<eureka::Calculate_localization> loc;
    std::shared_ptr<eureka::Construction_map> map;
    rclcpp::TimerBase::SharedPtr timer_;

    /* Пример указания целевой позиции для 4-х целевых точек,
     * до которых ровер будет двигаться автономно.
     * Точек на маршруте движения ровера может быть любое кол-во.
     */
    void send_goal(int goal_id)
    {
        double goals[4][3] = {
            {1.0, 0.0, 0.0},
            {2.0, 1.0, 1.57},
            {1.0, 2.0, 3.14},
            {0.0, 1.0, -1.57}
        };

        /* Вызываем функцию запуска навигации до целевой точки
         * с указанием целевой позиции.
         */
        nav->move_to(goals[goal_id][0], goals[goal_id][1], goals[goal_id][2]);
    }
};

int main(int argc, char** argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<SimpleNavigation>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

7. Вернитесь в директорию `ros2_ws` и соберите пакет введя команду:
`colcon build --packages-select variant`
8. Далее введите команду `source install/setup.bash` для обновления файлов вашего рабочего пространства
9. Запустите пакет с примером введя команду:
`ros2 run variant variant`