Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Направление подготовки: 09.04.04 – Программная инженерия

Магистерская программа: Робототехника

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА И ИМПЛЕМЕНТАЦИЯ ПРАВИЛ ДВИЖЕНИЯ ГЕТЕРОГЕННЫХ НАЗЕМНЫХ РОБОТОВ В СРЕДЕ УМНОГО ГОСПИТАЛЯ

Обучающийся 2 курса

группы 11-931 Сафин Р.Р.

Научный руководитель

PhD (техничекие науки),

профессор кафедры Магид Е.А.

интеллектуальной

робототехники

Директор ИТИС КФУ

канд. техн. наук Абрамский М.М.

Казань – 2021

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

В сфере робототехники, помимо промышленной области, заметно развитие использования роботов в роли помощников, функционирующих в одной среде с людьми и другими роботами [1]. Развитие этого направления особенно заметно в отношении мест, где наблюдается необходимость выполнять постоянные рутинные задачи, к примеру – перемещение тех или иных предметов из одного помещения в другое, или требуется наличие социального компаньона, работающего с клиентами и выполняющего те или иные задачи рабочего персонала - например, отели, больницы и офисы. Использование роботов для выполнения транспортных задач значительно снизит стоимость ресурсов и рабочего времени людей для ряда рутинных задач [2], однако такой подход требует решения таких проблем, как надежная навигация, безопасность движения роботов, а также решения логистических задач, связанных с оптимальным маршрутом роботов. К тому же, в отличие от промышленных роботов, сервисные роботы работают в одном же пространстве с другими роботами и людьми, и им необходимо работать в соответствии с набором правил и требований, чтобы поведение роботов было предсказуемым, облегчало работу персонала и повышало его продуктивность [3][4]. Решение этих проблем, в свою очередь, требует тестирования робототехнической системы для самых различных сценариев.

Данная работа была разработана с использованием фреймворка ROS [5] и протестирована с помощью виртуального моделирования в среде Gazebo [6]. Тестирование проводилось в виртуальной модели этажа медицинского учреждения, где роботы выполняют транспортировочные задачи, перемещаясь в одной среде. Виртуальное моделирование позволяет использовать модели роботов, соответствующие их реальным физическим характеристикам и оснащенных датчиками. Для тестирования системы использовались роботы TIAGo Base (PAL-Robotics) [7][8] и Ridgeback (Clearpath) [9] [10]. Задачи отправляются роботам посредством графического

интерфейса, после чего роботы были начинают их выполнение в соответствии с приоритетами задач, избегая при этом столкновения с другими участниками движения, при этом уступая дорогу роботам, выполняющим задачи с более высоким приоритетом.

Целью данной работы является создание робототехнической системы, включающей в себя имплементацию модели правил движения для группы гетерогенных роботов. Для достижения поставленной цели был составлен следующий ряд задач:

- Постановка правил движения группы гетерогенных наземных роботов в госпитале.
- Настройка и отладка конфигурации системы навигации для всех используемых роботов.
- Проектирование и реализация конечного автомата для описания возможных состояний роботов и логики их поведения в этих состояниях.
- Реализация менеджера и графического интерфейса для назначения роботам навигационных задач.

Данная работа состоит из нескольких этапов:

1) *Обзор литературы*. Изучение и анализ научных работ, рассматривающих проблемы навигации и логистики роботов в больничных условиях и закрытых помещениях в целом.
2) *Разработка менеджера задач*. В данном разделе описана работа менеджера задач и его графического интерфейса.
3) *Система навигации*. Описание принципов работы модуля навигации в системе ROS, обзор использованных элементов и их настройка.
4) *Разработка автомата состояний*. Применение библиотеки SMACH для разработки автомата состояний, определяющего поведение роботов во время навигации и выполнения задач.

5) *Разработка модели правил движения*. Описание правил движения для роботов во время навигации и соответствующих состояний.
6) *Выводы и заключение*. Проведенные эксперименты и их результаты, выводы о проделанной работе и возможные улучшения в дальнейшем.

2. ОБЗОР ЛИТЕРАТУРЫ

Одна из наиболее важных задач для работы с роботом в среде, состоящей из множества коридоров и помещений, в которых также передвигаются другие объекты (например, люди или другие роботы), - это эффективная и безопасная навигация, которая должна не только гарантировать, что робот достигнет своего следующего пункта назначения, но также избежит столкновений с другими движущимися объектами.

В работе [11] авторы предложили метод обхода движущихся и статических препятствий для всенаправленного мобильного робота с платформой для грузов на примере робота MKR (Muratec Keio Robot), изображенного на рисунке 2.1.

Рисунок 2.1. Робот MKR-003

Их подход основан на планировании пути с использованием виртуальных потенциальных полей [12]. Однако этот подход не подходит для использования с помощью роботов с дифференциальным приводом, алгоритмы навигации для которых обычно демонстрируют эффективность и стабильность в среде со статическими препятствиями, однако не всегда обеспечивают возможность решения проблемы динамического избегания от препятствий [13].

В другом подходе авторы предлагают систему мобильного робота, предназначенную для решения задач по доставке различных медицинских предметов с одной станции на другую в условиях больницы [14]. В качестве мобильной платформы был использован робот Nomadic XR4000. Потолки больницы оснащены люминесцентными лампами, которые используются роботом как естественные ориентиры для определения своего положения и ориентации. Недостатком данного подхода является использование искусственных меток локализации (люминесцентные лампы), что требует предварительной модификации пространства, в котором работают роботы, и зачастую влечет значительные дополнительные расходы на оснащение окружения подобными метками.

В следующей работе рассматривается проблема согласованного выполнения отдельных задач нескольких роботов, где роботы с течением времени получают задачи от разных источников [15]. Авторы описали подход координации роботов с низкой частотой, что требует от роботов отслеживания своих состояний и возможности запрашивать состояния друг друга только когда это необходимо. Подход основан на представлении задач в виде графов инструкций и разреженной координации с использованием метода Sparse Coordination Instruction Graphs [16].

В [17] рассматривается проблема управления процессом назначения задач для мобильных роботов, работающего в одном пространстве с людьми. Изначально роботы получают задачи через централизованный источник (веб-

интерфейс) и выполняют их без возможности изменить процесс выполнения задач во время их выполнения. Авторы представили решение, позволяющее прерывать (в том числе временно) выполнение текущей задачи – как явно отменять задачи, так и запрашивать текущий статус задачи робота, назначать новую задачу, которая должна быть завершена либо сразу, либо может быть отложена. Взаимодействие с роботом осуществляется при помощи голосового помощника на борту робота. Результаты были протестированы на основе различных сценариев прерывания текущих задач робота.

3. ИНСТРУМЕНТАЛЬНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

3.1. Robotic Operating System

Robotic Operating System (сокр. ROS) - это популярная платформа для разработки приложений для роботов, которая предоставляет различные функции, такие как передача сообщений, распределенные вычисления, повторное использование кода и так далее. Благодаря системе пакетов и публичному репозиторию пакетов, который позволяет публиковать и использовать пользовательские пакеты из единой системы, ROS позволяет максимизировать повторное использование кода и значительно ускорить процесс разработки.

Экосистема фреймворка ROS состоит из ряда элементов, и далее будут подробнее рассмотрены основные из них.

Пакет: Пакеты (англ. package) ROS представляют из себя базовую единицу программного обеспечения ROS. Пакеты содержат узлы (англ. node) ROS, каждая из которых работает как отдельный процесс, библиотеки, файлы конфигурации и т. д., которые организованы вместе как единое целое. Пакеты - это элементарный и необходимый элемент сборки в программном обеспечении ROS. Структура типичного пакета изображена на рисунке 3.1.1.

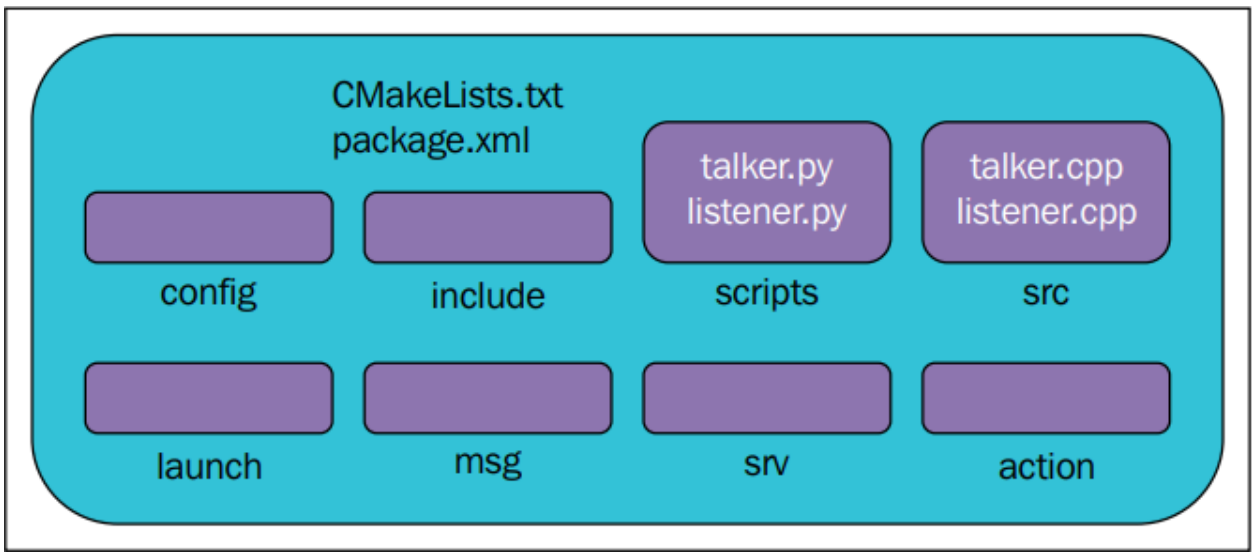

Рис. 3.1.1. Структура пакета в системе ROS

Структура пакета в системе ROS обычно состоит из следующих директорий:

- **config**: в этой папке хранятся все файлы конфигурации, которые используются в данном пакете ROS. Эта папка создается пользователем и ее принято называть config, чтобы в ней хранились файлы конфигурации.
- **include/package_name**: в данной папке хранятся заголовки и библиотеки, которые будут использоваться внутри пакета.
- **scripts**: в этой папке хранятся исполняемые скрипты Python.
- **src**: в этой папке хранятся исходные коды C++.
- **launch**: в этой папке хранятся файлы запуска (. launch), которые используются для запуска одного или нескольких узлов ROS.
- **msg**: в этой папке содержатся пользовательские типы сообщений (.msg).
- **srv**: в этой папке содержатся пользовательские сервисы.
- **action**: в данное папке содержатся пользовательские. action файлы. Они используются при работе с ROS модулем actionlib и нужны для создания пользовательских “действий” (англ. action).
- **package.xml**: Файл манифеста пакета.
- **CMakeLists.txt**: Файл сборки CMake пакета.

Манифест пакета - это файл манифеста пакета находится внутри пакета, который содержит информацию о пакете, его авторе, лицензии, зависимостях, флагах компиляции и т. д. Файл package.xml внутри пакета ROS является файлом манифеста этого пакета.

Мета-пакеты. термин “метапакет” используется для обозначения группы пакетов специального назначения. Одним из примеров метапакета является стек навигации ROS.

Манифест метапакетов. Манифест метапакета похож на манифест пакета, но различия заключаются в том, что он может включать пакеты внутри себя в качестве зависимостей времени выполнения и объявлять тег экспорта.

Сообщения в системе ROS - это структура данных, которая может быть отправлена от одного процесса ROS к другому посредством топиков. Расширение файла сообщений - .msg.

Сервисы. Сервис ROS предоставляет способ взаимодействия между процессами типа запрос/ответ. Типы данных ответа и запроса могут быть определены внутри папки srv внутри пакета с расширением srv.

3.2 Gazebo

Gazebo - это симулятор роботов и физического окружения для комплексного моделирования робототехники в помещениях и на открытом пространстве. Gazebo позволяет моделировать и тестировать в виртуальной симуляции сложных роботов, их сенсоры, а также различные 3D-объекты. В репозитории Gazebo предоставлены готовые модели популярных роботов, датчиков и различных 3D-объектов. Помимо этого, Gazebo предоставляет возможность использовать плагины, описывая логику работы сенсоров робота или создавать пользовательскую логику поведения физики в симуляции.

Gazebo состоит из следующих основных компонентов:

- **Файлы мира** (англ. world files) - содержат все элементы моделирования, включая роботов, источники света, датчики и статические объекты. На рисунке 2 представлен пример мира Gazebo и его графический интерфейс.

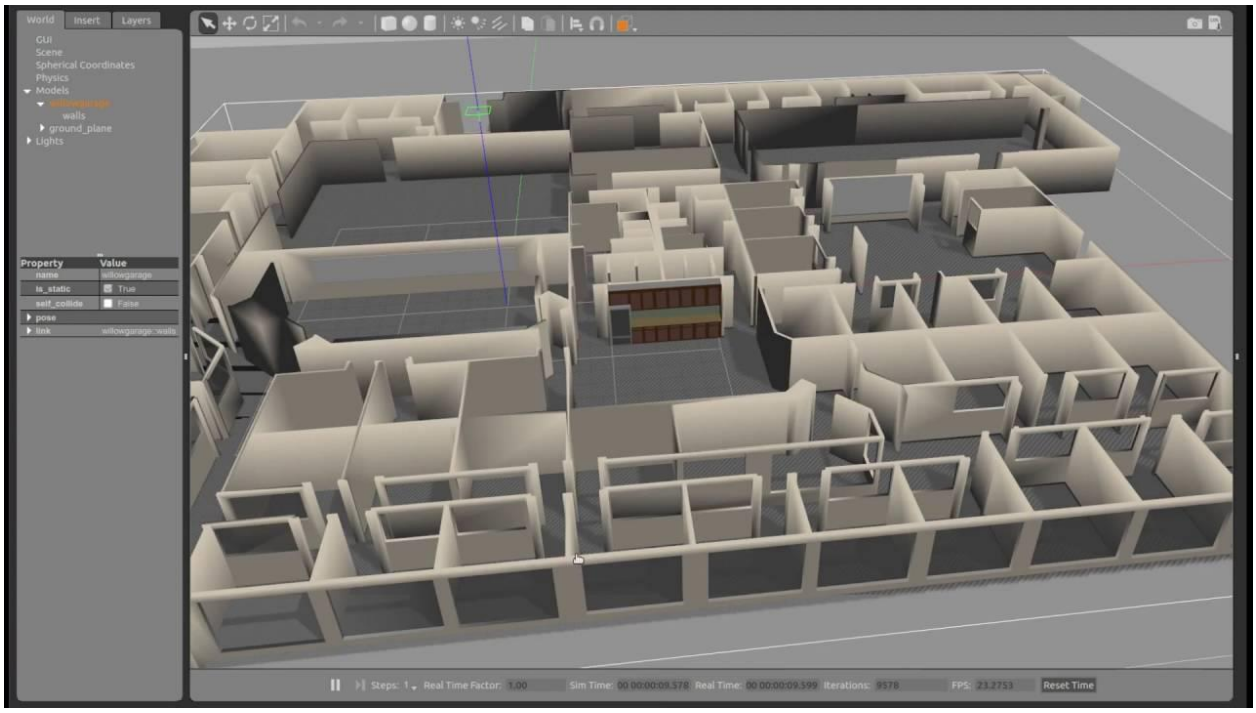

Рисунок 3.2.1. Запущенная симуляция Gazebo и его графический интерфейс.

- **Модели** представляют собой отдельные элементы в симуляции. Это могут быть как статичные предметы окружения, такие, как мебель или стены, а также более сложные или движущиеся объекты, например, роботы или люди.
- **gzserver –** Сервер симуляции, считывающий содержимое .world файлов, создавая и предоставляя доступ к симуляции.
- **gzclient** – Клиент, подключается к серверу симуляции gzserver, визуализирует содержимое сгенерированного мира и предоставляет графический интерфейс для взаимодействия с ним.

4. ОБЗОР ИСПОЛЬЗОВАННОЙ СИМУЛЯЦИИ

Тестирование роботов в реальных условиях занимает много времени, и, более того, тестирование новых изменений системы в больничной среде может представлять угрозу безопасности из-за непредсказуемости непроверенного кода. Тестирование в виртуальной среде со смоделированным миром увеличивает покрытие сценариев тестирования, значительно снижает риск как вывода тестируемого оборудования из строя, так и несчастных случаев, а также значительно сокращает время, затрачиваемое на разработку. Однако, моделирование виртуальных миров со специфичным для выполняемой задачи окружением является трудоемким процессом и требует специальных навыков в 3D-моделировании. По этой причине в качестве модели среды для тестирования системы была использована среда AWS (Amazon Web Services) Robomaker Hospital World [18], разработанная для использования в симуляторе сред Gazebo, имитирующую помещения госпиталя. На рисунках 4.1 и 4.2 продемонстрированы помещения и модели из симуляции. На рисунке 4.4 изображен вид сверху на всю симуляцию.

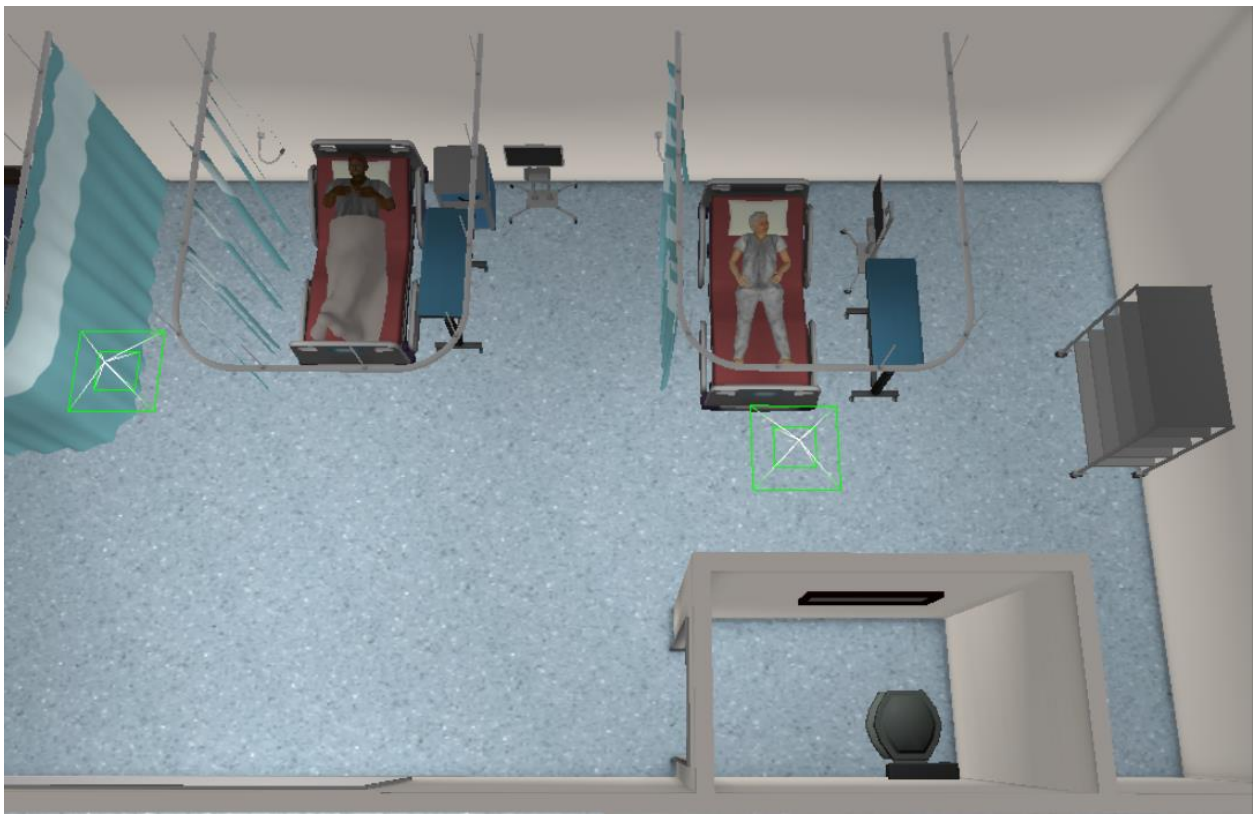

Рисунок 4.1. Участок помещения палаты в симуляции Gazebo

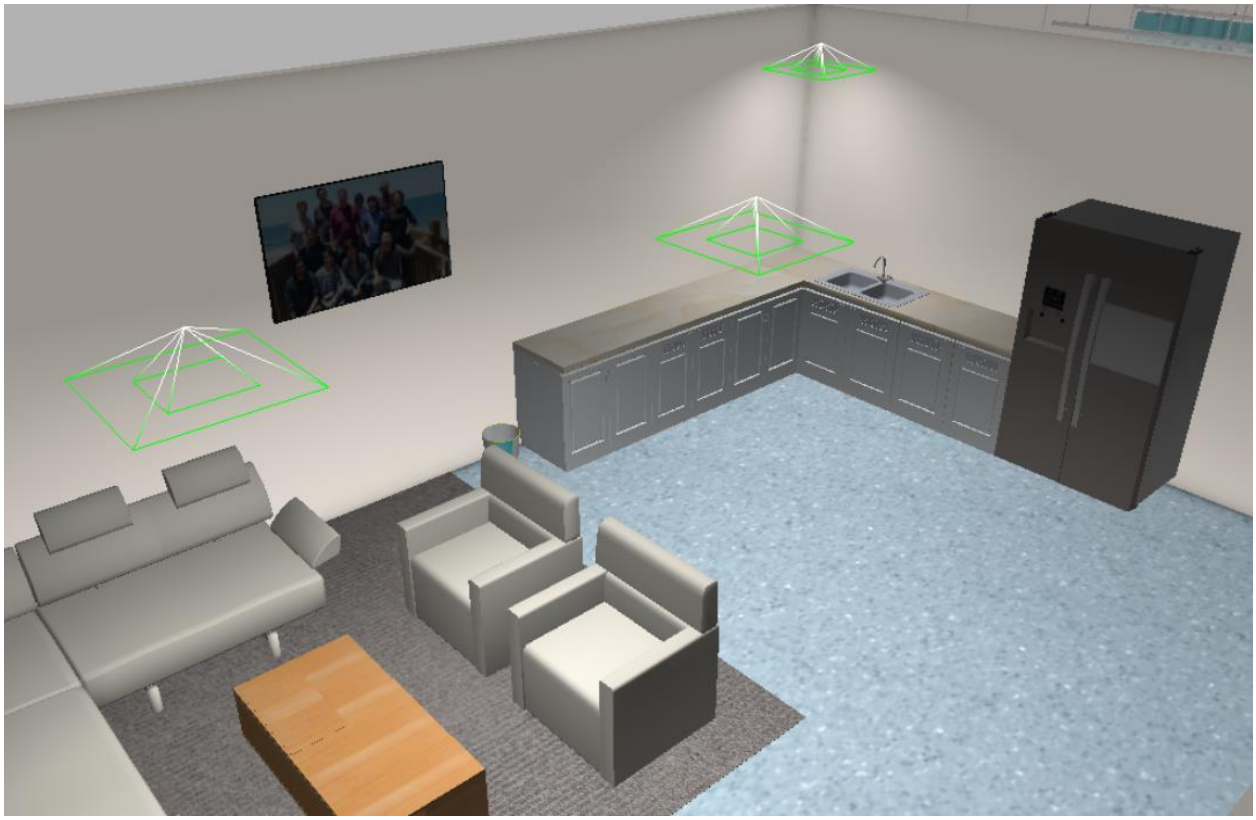

Рисунок 4.2. Участок комнаты отдыха для персонала в симуляции Gazebo

Использованная виртуальная модель больницы обладает большой площадью и состоит из множества комнат, включая смотровые комнаты, палаты пациентов, два склада и комнату отдыха для персонала. Большие и сложные благодаря наполнению объектами миры подходят для тестирования систем с множеством роботов с автономной навигации. В комнатах расположены больничные койки, стулья и мебель, что делает тестирование алгоритмов обхода препятствий более реалистичным.

Рисунок 4.3. Вид сверху на симуляцию Robomaker Hospital World

5. СИСТЕМА НАВИГАЦИИ

В данной главе описан процесс настройки и отладки модуля навигации для используемых в разработанной системе роботов, а также описано устройство самого модуля навигации во фреймворке ROS.

Основная задача модуля навигации ROS - передвижение робота из начальной позиции в целевую, не допуская столкновения с окружающей средой. В состав модуля навигации ROS входят пакеты, включающие в себя реализации нескольких алгоритмов, связанных с навигацией, и использующихся в имплементации автономной навигации для мобильных роботов. Среди алгоритмов, реализованных в модуле навигации и готовых к

использованию “из коробки”, можно выделить наиболее стандартные и популярные алгоритмы SLAM [19], A* (англ. A star), алгоритм Дейкстры (англ. Dijkstra) [20], AMCL (англ. Adaptive Monte Carlo Localization) [21] [22] [23]. Для работы модулю навигации требуются такие данные, как целевое положение робота, данные одометрии робота с датчиков, к примеру, IMU и GPS, и другие потоки данных с датчиков, например, показатели лазерного дальномера или трехмерное облако точек от таких датчиков, как Kinect.

Результатом работы пакета навигации будут команды скорости, которые приведут робота к заданной целевой позиции.

Структуру модуля навигации ROS можно изобразить в виде диаграммы на рисунке 1.

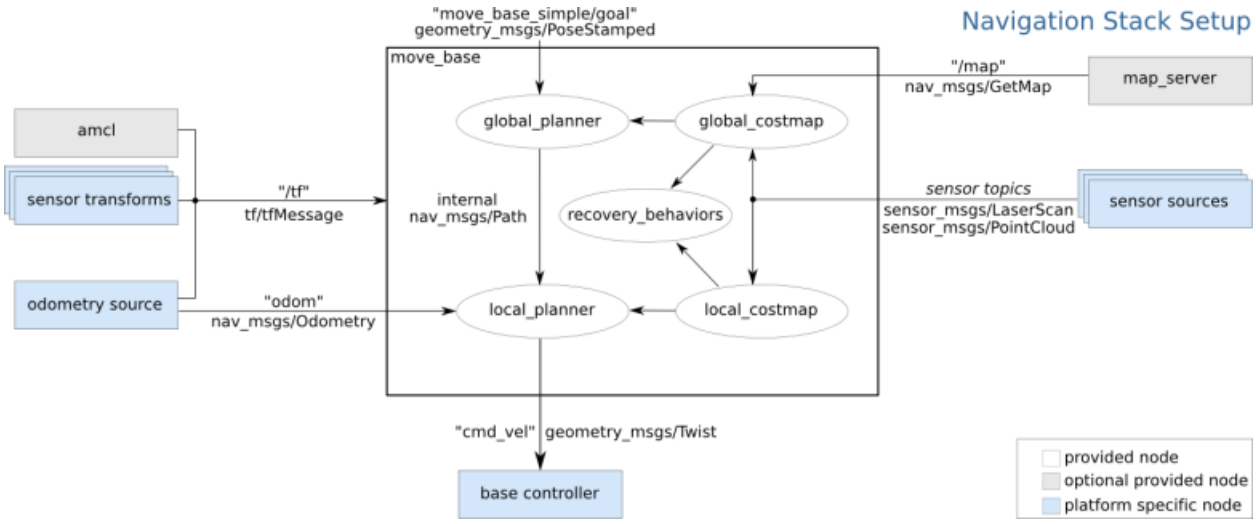

Рисунок 1. Конфигурация модуля навигации ROS

Перед использованием модуля навигации следует убедиться, что настройка физических характеристик робота удовлетворяет следующим требованиям:

1. Робот классифицируется как голономный, или относится к роботам с дифференциальным приводом. Управление движением робота реализуется посредством отправки роботу команд следующего вида: линейные скорости x и y, угловая скорость θ.

2. На мобильной платформе робота должен быть установлен лазерный дальномер, причем таким образом, чтобы он располагался параллельно поверхности, по которой будет передвигаться робот.
3. Наиболее оптимальными для использования со стеком навигации являются роботы квадратных и круглых форм. Несмотря на то, что модуль навигации может быть настроен для роботов с любой формой, использование роботов произвольной формы (отличной от круглой/квадратной) может привести к ухудшению корректности работы модуля навигации.

Согласно структуре модуля, продемонстрированной на рисунке 1, для работы модулю навигации ROS необходимы следующие источники входных данных:

1. Источник показателей одометрии робота (odometry source). Данные одометрии робота описывают позицию робота относительно его начальной позиции. Основными источниками одометрии являются колесные энкодеры, IMU и камеры 2D/3D (одометрия). Значения одометрии, публикуемые в стек навигации, должны быть обернуты в сообщение с типом nav_msgs/Odometry.
2. Источник показателей датчиков робота. Обычно ожидаются данные, полученные при помощи лазерных дальномеров или сенсоров Kinect, публикуемые в модуль навигации с типом sensor_msgs/LaserScan или sensor_msgs/PointCloud. Эти данные, в сочетании с одометрией робота, используются для построения глобальной и локальной карт стоимости.
3. Отношения (преобразования) между системами координат робота, в качестве источника для которых используется ROS пакет tf/tf2 [24] [25].

Также обязательным для модуля навигации внешним компонентом является контроллер базы (англ. Base controller), отвечающий за преобразование результата работы модуля, представляющим из себя

сообщения типа geometry_msgs/Twist, в соответствующие команды скорости для моторов мобильной базы робота.

6.1. Пакет move_base

Основной задачей ROS пакета move_base [26] является использование данных, полученных при помощи остальных компонентов модуля навигации, для приведения робота в движение посредством их обработки и отправки в контроллер базы, упомянутый выше. Пакет move_base отвечает за координацию таких компонентов, как: глобальный планировщик (англ. Global planner), локальный планировщик (англ. Local planner), поведение при восстановлении (англ. Recovery behavior), локальная карта стоимости (англ. Global costmap), глобальная карта стоимости (англ. Local costmap). Конфигурация и использование в проекте вышеперечисленных компонентов, а также тесно связанных с move_base пакетов map-server [27], amcl и gmapping, представлена ниже.

6.2 Конфигурация планировщика пути

В качестве локального планировщика пути был использован пакет teb_local_planner [28] [29] [30]. Данный пакет реализует планировщик оптимальных локальных траекторий для навигации и управления мобильными роботами в качестве подключаемого модуля для пакета навигации ROS. Начальная траектория, созданная глобальным планировщиком, оптимизируется во время выполнения в отношении минимизации времени прохождения роботом траектории, предотвращения столкновения с препятствиями и соответствия ограничениям, таким как соблюдение максимальных скоростей и ускорений. Подход основан на методе Elastic Band [31], предложенном исследователем Оуссама Хатиб (англ. Oussama Khatib) в 1993 году.

В системе ROS пакет публикует данные в следующие топики:

- ~/global_plan — Глобальный план, в соответствии с которым планировщик строит путь.
- ~/local_plan — Локальный план, которому следует teb_local_planner.
- ~/teb_poses — Список дискретных позиций (geometry_msgs/PoseArray), из которых состоит текущий локальный план.
- ~/teb_marker — Teb_local_planner предоставляет дополнительную информацию о сцене планирования с помощью маркеров с разными пространствами имен. Пространства имен PointObstacles и PolyObstacles служат для визуализации всех точечных и многоугольных препятствия, учитывающихся в процессе построения и оптимизации пути. Пространство имен TebContainer служит для визуализации найденных и оптимизированных траекторий, лежащих в альтернативных топологиях.

Вышеперечисленные топики служат преимущественно в целях визуализации.

- ~/teb_feedback — Содержит запланированную траекторию, включая данные о скорости, времени и списке препятствий. Используется в основном в целях отладки.

Планировщик teb_local_planner подписан на следующие топики:

- ~/odom — Данные об одометрии робота, предоставляющая локальному планировщику информацию о текущей скорости робота.
- ~/obstacles — Массив, содержащий список препятствий в виде точек, линий или многоугольников (в дополнение к препятствиям на карте затрат или вместо них).
- ~/via_points — Позволяет отправлять пользовательские промежуточные точки.

Настройка планировщика пути заключается в подборе оптимальных параметров, достигая тем самым желаемого поведения робота в процессе

навигации. Параметры содержатся в .yaml файле, но явное указание всех параметров не является обязательным, поскольку для каждого параметра существует значение по умолчанию, используемое планировщиком при отсутствии пользовательского значения параметра. Для того, чтобы .yaml файл был использован планировщиком, необходимо указать его название при инициализации планировщика при запуске узла move_base. Ниже представлен список используемых в системе параметров планировщика teb_local_planner с кратким описанием предназначения параметра, используемым значением и значением параметра по умолчанию. Список параметров разбит на категории в соответствии с тем, на что влияет изменение этих параметров.

6.2.2. Параметры конфигурации робота

Таблица 6.2.2.1. – список параметров конфигурации робота

Наименование параметра	Описание	Значение по умолчанию	Использованные значения для:	
			PMB-2	Ridgeback
acc_lim_x	Максимальное линейное ускорение робота в м/с^2	0.5	0.2	10.0
acc_lim_theta	Максимальное угловое ускорение робота в рад/с^2	0.5	4	20.0
max_vel_x	Максимальная поступательная скорость робота в м/с	0.4	0.7	0.5
max_vel_x_backwards	Максимальная абсолютная линейная скорость робота при движении назад в м/с	0.2	0.15	0.2
max_vel_theta	Максимальная угловая скорость робота в рад/с	0.3	8	1.75
footprint_model/type	Тип формы робота, используемый для оптимизации. Параметр оказывает значительное влияние на производительность планировщика. Возможными типами являются: “point”, “circle”, “line”, “two_circles” и “polygon”.	“point”	“circular”	“point”
footprint_model/radius	Только для типа footprint_model/type = “circle”. Содержит значение радиуса круга. Центр круга расположен на оси вращения робота.	0.2	0.275	-

footprint_model/ line_start	Только для типа “line”. Содержит координаты начала отрезка линии.	[-0.3, 0.0]	-	-
footprint_model/ line_end	Только для типа "line". Содержит координаты конца отрезка линии.	[0.3, 0.0]	-	-
footprint_model/ front_offset	Только для типа “two_circles”. Характеризует смещение центра передней окружности по оси x робота (в положительном направлении).	0.2	-	-
footprint_model/ front_radius	Только для типа “two_circles”. Содержит радиус передней окружности.	0.2	-	-
footprint_model/ rear_offset	Только для типа “two_circles”. Характеризует смещение центра задней окружности по оси x робота (в отрицательном направлении).	0.2	-	-
footprint_model/ rear_radius	Только для типа “two_circles”. Содержит радиус задней окружности.	0.2	-	-
footprint_model/ vertices	Только для типа “polygon”. Содержит список координат вершин многоугольника. Замыкать многоугольник повторением первой вершины в конце списка не требуется.	[[0.25,-0.05], [...], ...]	-	-

6.2.3. Параметры допустимого отклонения от цели

Таблица 6.2.3.1. – список параметров допустимого отклонения от цели

Наименование параметра	Описание	Значение по умолчанию	Использованные значения для:	
			PMB-2	Ridgeback
xy_goal_tolerance	Допустимое евклидово расстояние до координат цели (в метрах)	0.2	0.2	0.2
yaw_goal_tolerance	Допустимое отклонение ориентации робота на момент достижения цели (в радианах)	0.2	0.2	0.2
free_goal_vel	Запрет/разрешение достижения роботом конечной цели без снижения скорости (логический тип).	false	false	false

6.2.4. Параметры конфигурации траектории

Таблица 6.2.4.1. – список параметров конфигурации траектории

Наименование параметра	Описание	Значение по умолчанию	Использованные значения для:	
			PMB-2	Ridgeback
dt_ref	Желаемое временное разрешение траектории	0.3	0.3	-
dt_hysteresis	Гистерезис автоматического изменения траектории в зависимости от текущего значения dt_ref	0.1	0.1	-
min_samples	Минимальное количество рассматриваемых экземпляров траекторий	3	5	-
global_plan_overwrite_orientation	Запрет/разрешение изменения ориентаций промежуточных целей глобального пути	true	true	-
global_plan_viapoint_sep	Разрешение на удаление промежуточных точек из глобального плана, что меняет разрешение пути.	-0.1	1.0	-
max_global_plan_lookahead_dist	Максимальная длина подмножества точек глобального пути, учитываемого при оптимизации траектории	3.0	5.0	-
force_reinit_new_goal_dist	Повторная инициализация траектории при условии, что предыдущая цель была обновлена с большей разницей, чем значение данного параметра (в метрах)	1.0	1.0	-
feasibility_check_no_poses	До какой позиции на прогнозируемой траектории должна проверяться достижимость точки на каждом интервале выборки	4	5	-
publish_feedback	Разрешить/запретить публикацию данных с текущей полной траектории и списком учитываемых препятствий (логический тип)	false	false	-
shrink_horizon_backup	Разрешить/запретить планировщику временно сужать горизонт (50%) в случае обнаружения недопустимой траектории	true	true	-
allow_init_with_backwards_motion	Разрешить/запретить движение задним ходом в случае, если цель находится за роботом в рамках локальной карты стоимостей (логический тип). Не рекомендуется разрешать движение задним ходом в случае, если робот не оборудован лазерным дальномером (или иным датчиком, позволяющим обнаружить	false	false	-

	препятствия) на задней части корпуса.			
exact_arc_length	Разрешить/запретить использование планировщиком точной длины дуги в вычислениях скорости робота, скорости поворота и ускорения. Если значение равно “false”, то используется евклидово приближение.	false	false	-
shrink_horizon_min_duration	Минимальная продолжительность сужения горизонта в случае обнаружения недопустимой траектории (см. параметр shrink_horizon_backup.	false	false	-

6.2.5. Параметры обработки препятствий

Таблица 6.2.5.1. – список параметров обработки препятствий

Наименование параметра	Описание	Значение по умолчанию	Использованные значения для:	
			PMB-2	Ridgeback
min_obstacle_dist	Минимальное желаемое расстояние от препятствий в метрах	0.5	0.06	-
include_costmap_obstacles	Если true, то будут учитываться препятствия на локальной карте стоимости	true	True	-
costmap_obstacles_behind_robot_dist	Ограничение препятствий на локальной карте затрат, учитываемые при планировании позади робота (в метрах).	1.0	1.0	-
obstacle_poses_affected	Каждое препятствие привязано к ближайшей позе на траектории, чтобы сохранить дистанцию. Также могут быть учтены дополнительные соседи.	30	30	-
inflation_dist	Буферная зона вокруг препятствий с ненулевой стоимостью штрафа (должна быть больше, чем min_obstacle_dist)	0.6	0.6	-
include_dynamic_obstacles	Если true, то движущиеся препятствия учитываются в	false	true	-

	процессе оптимизации пути (скорость движения препятствий считается постоянной величиной)			
legacy_obstacle_association	Выбор версии стратегии для соединения траектории позы с препятствиями.	false	false	-
obstacle_association_force_inclusion_factor	Радиус, в рамках которой задействована стратегия объединения не унаследованных препятствий (англ. non-legacy obstacles).	1.5	1.5	-
obstacle_association_cutoff_factor	Ограничение действия стратегии (игнорируются все препятствия, лежащие вне радиуса [значение данного параметра] *min_obstacle_dist). Учитывается только при obstacle_association_force_inclusion_factor = true.	5	5	-

Следующие параметры актуальны только при использовании пакета costmap_converter.

costmap_converter_plugin	Имя плагина, использующегося для преобразования занятых ячеек карты стоимости в другие структуры (например, в точки/линии/многоугольники). Отсутствие плагина (при пустом значении параметра) отключит преобразование, и все препятствия будут рассматриваться как отдельные точки.	“”	costmap_converter::CostmapToPolygonsDBSConcaveHull	-
costmap_converter_spin_thread	Если установлено значение true, преобразователь карты стоимости вызывает свою очередь обратного вызова (англ. callback) в другом потоке.	true	true	-
costmap_converter_rate	Частота, с которой плагин costmap_converter обрабатывает текущую карту стоимости (в гц.)	5.0	5.0	-

6.2.6 Параметры оптимизации

Таблица 6.2.6.1. – список параметров оптимизации

Наименование параметра	Описание	Значение по умолчанию	Использованные значения для:	
			PMB-2	Ridgeback
no_inner_iterations	Количество итераций оптимизатора, вызываемых в каждой итерации внешнего цикла (англ. outerloop)	5	5	-
no_outer_iterations	Количество итераций внешнего цикла, каждая из которых изменяет траекторию в соответствии с разрешением dt_ref и вызывает внутренний оптимизатор	4	4	-
penalty_epsilon	Допустимое отклонение для штрафных функций приближений с жесткими ограничениями	0.1	0.1	-
weight_max_vel_x	Значение веса оптимизации для максимально допустимой скорости линейного движения по оси x робота	2.0	2.0	-
weight_max_vel_theta	Значение веса оптимизации для максимально допустимой угловой скорости робота	1.0	10.0	-
weight_acc_lim_x	Значение веса оптимизации для максимально допустимого ускорения движения робота по оси x	1.0	1.0	-
weight_acc_lim_theta	Значение веса оптимизации для максимально допустимого углового ускорения робота	1.0	1.0	-
weight_kinematics_nh	Значение веса оптимизации для кинематики неголомных (англ. non-holonomic) роботов	1000.0	1000.0	-
weight_kinematics_forward_driv	Значение веса оптимизации, влияющий на выбор роботом движения вперед по оси x. При низких значениях чаще допускается движение задним ходом	1.0	500.0	-
weight_kinematics_turning_radius	Значение веса оптимизации для ограничения минимального радиуса поворота	1.0	1.0	-
weight_optimal time	Значение веса оптимизации для сокращения траектории относительно времени перехода/выполнения	1.0	1.0	-
weight_obstacle	Значение веса оптимизации для удержания минимального расстояния от препятствий	50.0	50.0	-

weight_viapoint	Значение веса оптимизации для сокращения траектории относительно времени перехода/выполнения	1.0	0.75	-
weight_inflation	Значение веса оптимизации для минимизации расстояния до промежуточных точек (или опорного пути)	0.1	0.1	-
weight_adapt_factor	Некоторые специальные веса многократно масштабируются с помощью этого коэффициента. Итеративное увеличение весов вместо априорной установки огромного значения приводит к лучшим численным условиям основной задачи оптимизации	2.0	2.0	-

6.2.7. Параллельное планирование в отдельных топологиях

Таблица 6.2.7.1. – список параметров параллельного планирования

Наименование параметра	Описание	Значение по умолчанию	Использованные значения для:	
			PMB-2	Ridgeback
enable_homotopy_class_planning	Разрешить/запретить параллельное планирование в различных топологиях (значительно влияет на производительность)	true	false	-
enable_multithreading	Разрешить/запретить многопоточное планирование (при true каждая траектория вычисляется в отдельном потоке)	true	true	-
max_number_classes	Максимальное количество учитываемых различных траекторий	4	4	-
selection_cost_hysteresis	“Стоимость” траектории, которую должен иметь новый кандидат относительно ранее выбранной траектории, чтобы новая траектория была выбрана	1.0	1.0	-
selection_obst_cost_scale	Дополнительное масштабирование стоимости препятствий для выбора “лучшего” кандидата	100.0	100.0	-
selection_viapoint_cost_scale	Дополнительное масштабирование стоимости промежуточных точек для выбора “лучшего” кандидата	1.0	1.0	-
selection_alternative_time_cost	При значении true временные затраты (сумма квадратов разниц времени)	false	false	-

	заменяются общим временем перехода (суммой разниц времени)			
roadmap_graph_no_samples	Количество образцов, сгенерированных для создания графа дорожной карты	15	15	-
roadmap_graph_area_width	Ширина региона между началом пути и целевой точкой, из которой подбираются случайные ключевые/путевые точки	6	6	-
h_signature_prescaler	Параметр масштабирования (h-сигнатура), используемый для разграничения классов гомотопии	1.0	0.5	-
h_signature_threshold	Предполагается, что две h-сигнатуры равны, если разница между действительными и сложными частями ниже заданного порога	0.1	0.1	-
obstacle_heading_threshold	Значение скалярного произведения между положениями препятствия и цели, используется для принятия решения для учета этих препятствий в процессе исследования	1.0	0.45	-
visualize_hc_graph	Визуализация графа, простроенного для исследования различных траекторий (может быть использован для визуализации в rviz)	false	false	-
viapoints_all_candidates	Разрешить/запретить присоединение различных топологий к набору промежуточных точек	true	true	-
switching_blocking_period	Время, которое должно истечь, прежде чем будет разрешен переход на новый класс эквивалентности (в сек.)	0.0	0.0	-

6.2.8 Прочие параметры

Таблица 6.2.8.1. – список прочих параметров

Наименование параметра	Описание	Значение по умолчанию	Использованные значения для:	
			PMB-2	Ridgeback
odom_topic	Название топика сообщения одометрии, предоставленное водителем или симулятором робота	”odom”	/mobile_base_controller/odom	/ridgeback/odom
map_frame	Глобальная система координат планирования	”odom”	“map”	“map”

6.3. Конфигурация карты стоимости

Картой стоимости в рамках терминологии ROS называют структуру, предоставляемую пакетом costmap_2d [32], предназначенную для управления и обработки информации о том, куда может перемещаться робот. Для работы пакета (и, соответственно, построения карты стоимости) требуются данные внешних датчиков робота и статическая карта мира. Объект Costmap_2d::Costmap2DROS использует вышеперечисленные данные для обновления информации о препятствиях на карте. Объект costmap_2d::Costmap2DROS предоставляет только двухмерный интерфейс. Это означает, два препятствия, находящиеся в одних и тех же координатах по осям X и Y, но с разными координатами по оси Z, будут иметь идентичное значение стоимости (занятости) в соответствующей ячейке в карте стоимости объекта costmap_2d::Costmap2DROS.

Основной интерфейс предоставляется в классе costmap_2d::Costmap2DROS. Он содержит costmap_2d::LayeredCostmap, используемый для управления отдельными слоями карты стоимости. Каждый слой создается в costmap2DROS с помощью pluginlib и добавляется в LayeredCostmap. Сами слои можно компилировать индивидуально, что позволяет вносить произвольные изменения в карту затрат. Также класс costmap_2d::Costmap2D реализует базовую структуру данных для хранения и доступа к двумерной карте стоимости.

Карта стоимости получает данные с датчиков робота, автоматически подписываясь на соответствующие топики. Показатели с датчиков используются для обновления информации о препятствиях на карте стоимости (вставка новых препятствий и очистка старых. Операция обновления осуществляется изменением стоимости (значения) того или иного индекса в массиве. Операция очистки, в свою очередь, реализована с использованием трассировки лучей (англ. raytracing) через сетку от датчика в направлении

положения каждого уже обнаруженного препятствия. В случае, если для хранения информации о препятствиях используется трехмерная структура, информация о препятствиях проецируется в двумерную структуру (игнорируются позиции по оси Z).

Каждая ячейка на карте стоимости может иметь значение от 0 до 255, однако структура, на которой она основана, ограничена тремя значениями:

- Занятая ячейка (англ. Occupied cell):

  В зависимости от значения параметра mark_threshold и значения в соответствующей ячейке на карте стоимости ячейке назначается стоимость costmap_2d::LETHAL_OBSTACLE и она считается занятой
- Неизвестная ячейка (англ. Unknown cell):

  В зависимости от значения параметра unknown_threshold и значения в соответствующей ячейке на карте стоимости ячейке назначается стоимость costmap_2d::NO_INFORMATION и она считается неизвестной
- Свободная ячейка (англ. Free cell):

  Ячейка считается свободной, если не подходит под вышеперечисленные значения

Карта затрат обновляется с частотой, определяемой значением параметра update_frequency. В каждом цикле считываются данные датчиков и выполняются операции маркировки и удаления для базовой структуры карты стоимости, после чего эта структура проецируется в карту стоимости, где соответствующие значения затрат назначаются, как описано выше. Затем выполняется вычисление значений “инфляции” (англ. inflation) для каждого препятствия в каждой ячейке со стоимостью costmap_2d::LETHAL_OBSTACLE. В данном процессе значения стоимости

распространяются вокруг каждой занятой ячейки до заданного пользователем радиуса инфляции. Подробности этого процесса инфляции описаны ниже.

Инфляцией называют процесс распространения значений стоимости из занятых препятствиями ячеек вокруг препятствия с затуханием данных значений по мере отдаления от источника значений. Зона инфляции разделена на пять регионов, основываясь на вероятности столкновения робота с препятствием в каждой из них:

- Зона с “летальной” стоимостью (англ. Lethal cost) характеризует ячейку, в позиции которой на реальной карте есть препятствие.
- Зона со “вписанной” стоимостью (англ. Inscribed cost) покрывает ячейки, находящиеся на расстоянии меньшем, чем вписанный радиус робота, от препятствия. Вписанным радиусом роботам называют радиус окружности, вписанной в фигуру (англ. footprint), характеризующую положение робота на карте.
- Зона с “возможно описанной” стоимостью (англ. Possibly circumscribed cost). В данном случае описанная вокруг фигуры робота окружность используется в качестве границы, пересечение которой с препятствием на реальной карте возможно приведет к столкновению робота с этим препятствием – это зависит от ориентации робота на момент пересечения.
- Зона с “неизвестной” стоимостью (англ. Unknown cost) на карте затрат означает, о данной ячейке данных получено не было.
- Зоны со “свободной” стоимостью (англ. Freespace cost) достаточно сильно удалены от препятствий и робот будет свободно перемещаться по этим зонам.
- Всем остальным ячейкам присваивается значение в промежутке от значений “свободной” до “возможно описанной” стоимостей в

зависимости от их удаленности от “летальной” ячейки и заданной пользователем функции затухания.

В системе ROS пакет публикует данные в следующий топик:

- ~/costmap — значения карты стоимости
- ~/costmap_updates — значения обновленных участков карты стоимости
- ~/voxel_grid — Значения карты стоимости, если базовая карта стоимости основана на вокселях. Публикуется, если была явно запрошена публикация карты стоимости вокселей.

Пакет слушает только топик ~/footprint, и перезаписывает полученными данными текущее значение формы робота.

Работа пакета costmap_2d, аналогично планировщику, может быть настроена при помощи параметров. Ниже представлены списки параметров для настройки общей, локальной карты стоимости, их краткое описание, значения по умолчанию и использованные в системе значения.

6.3.1. Общие параметры карты стоимости

Таблица 6.3.1.1 – список общих параметров карты стоимости

Наименование параметра	Описание	Использованное значение
obstacle_range	Максимальное расстояние (в метрах) между роботом и препятствием, при котором данные о препятствии учитываются на карте стоимости	15
raytrace_range	Если расстояние между роботом и препятствием меньше, чем значение данного параметра, пространство между роботом и этим препятствием будет сочтено как свободная зона	1.0
max_obstacle_height	Максимальная высота показателей датчика о препятствиях, которые будут учитываться при обновлении карты стоимости.	2

min_obstacle_height	Минимальная высота показателей датчика о препятствиях, которые будут учитываться при обновлении карты стоимости.	0.05
robot_radius	Радиус следа робота (англ. footprint)	0.23
inflation_radius	Радиус вокруг препятствий, в пределах которого применяется функция масштабирования стоимости.	0.5
transform_tolerance	Максимально допустимая задержка между обновлением отношений фреймов дерева преобразований tf, в пределах которой считается, что все преобразования выполнены корректно	0.2
map_type	Тип используемой карты стоимости	costmap
observation_sources	Источники данных о препятствиях	scan

6.3.2. Параметры глобальной карты стоимости

Таблица 6.3.2.1. – список общих параметров глобальной карты стоимости

Наименование параметра	Описание	Использованное значение
global_frame	Фрейм (система координат), в котором будет работать карта стоимости.	scan
robot_base_frame	Наименование фрейма робота, соответствующего его базе (base_link)	base_footprint
update_frequency	Частота обновления карты затрат	5.0
publish frequency	Частота публикации данных о текущем состоянии карты затрат	1.0
static map	Параметр должен указывать, используется ли статичная карта	true

7. РАЗРАБОТКА МЕНЕДЖЕРА ЗАДАЧ

Менеджер задач предназначен для назначения роботам задач, получения и обработки обратной связи от роботов, отслеживания прогресса выполнения задач и хранения их текущих статусов.

Процесс работы менеджера задач можно описать следующим образом. Во время инициализации менеджера создаются и заполняются из заранее подготовленных файлов списки возможных приоритетов, станций и типов

задач. Заполнение списка роботов также происходит на данном этапе, однако список роботов, функционирующих в системе, хранится на сервере параметров ROS. Сервер параметров представляет из себя словарь, доступ к значениям которого осуществляется по ключу (списку роботов соответствует ключ “robots”). Данные списки будут также использованы в графическом интерфейса менеджера в качестве возможных значений при заполнении формы задачи. Задача назначается посредством графического интерфейса, через который можно выбрать тип задачи, станцию (точка на карте, преимущественно расположены в помещениях), где задача будет выполняться и куда должен будет последовать робот, приоритет задачи, в соответствии с которой формируется последовательность задач выбранного робота, идентификатор робота, которому будет назначена задача, и, опционально, время завершения задачи - чтобы определить, как долго робот будет ожидать, когда он прибудет на станцию. Если время ожидания не указано, робот приступит к выполнению следующей задачи (или перейдет в состояние WAIT_FOR_GOAL) сразу после выполнения текущей задачи. Состояния описаны более подробно в разделе 8. Структура данных задачи описана в таблице 7.1.

Таблица 7.1. – структура данных, описывающая задачу

Поле	Описание
id	Уникальный идентификатор задачи
priority	Целочисленное значение, определяющее очередь задачи и приоритет робота, выполняющего данную задачу
isDone	Логическое значение, устанавливается true, когда задача выполнена
isCurrent	Логическое значение, устанавливается true в процессе выполнения задачи
goalId	Уникальный идентификатор станции на карте
taskType	Тип выполняемой задачи
robotName	Имя робота, которому назначается задача
waitTime	Опциональный параметр, характеризующий время, которое робот будет ожидать после выполнения задачи

После назначения задачи диспетчер задач отправляет ее роботу, который, получив новую задачу, сортирует свой список задач в соответствии с их приоритетами и приступает к выполнению задачи с наивысшим приоритетом из списка. Перед отправкой задачи менеджер обновляет хранящиеся списки задач, дополняя их новой задачей, и сортирует списки, в качестве ключа сортировки используя поле priority. Объект Task используется преимущественно для хранения списков задач, для отправки же задачи роботу используется объектное представление сообщения TaskMsg.msg. Далее значения, полученные из заполненных пользователем полей графического интерфейса, сохраняются в объект сообщения, которое публикуется в топик “/goals”, на который подписан управляющий узел в системе робота.

Задачи можно назначать роботу, не дожидаясь завершения предыдущих, и каждый робот имеет свой собственный список задач. Менеджер задач хранит все задачи и отображает текущий статус задач в графическом интерфейсе, разделяя их на выполняемые, завершенные и незавершенные задачи. Статусы задач обновляются при переходах роботов из одного состояния в другое. Робот уведомляет менеджер задач после завершения задачи и когда он начинает выполнение следующей задачи. Реализовано два типа задач: ожидание и автономное движение к станции.

7.1. Разработка графического интерфейса

Для реализации графического пользовательского интерфейса для менеджера задач был использован фреймворк rqt, в свою очередь основанный на фреймворке Qt. Rqt [33] - это фреймворк в системе ROS, который предоставляет различные инструменты с графическим интерфейсом пользователя в виде плагинов. Архитектура rqt представляет из себя систему подключаемых модулей, позволяя внедрять модули графического интерфейса на основе Qt для использования в ROS. В данной работе использовалась версия PyQt5. Макет интерфейса был создан при помощи инструмента Qt

Designer [34]. Qt Designer - это инструмент Qt для проектирования и создания графических пользовательских интерфейсов (GUI) с виджетами Qt. Qt Designer позволяет “собрать” макет будущего графического интерфейса из отдельных элементов, сразу отображая его внешний вид, и на его основе сгенерировать .py файл, содержащий класс, в котором объявляются и инициализируются все выбранные элементы.

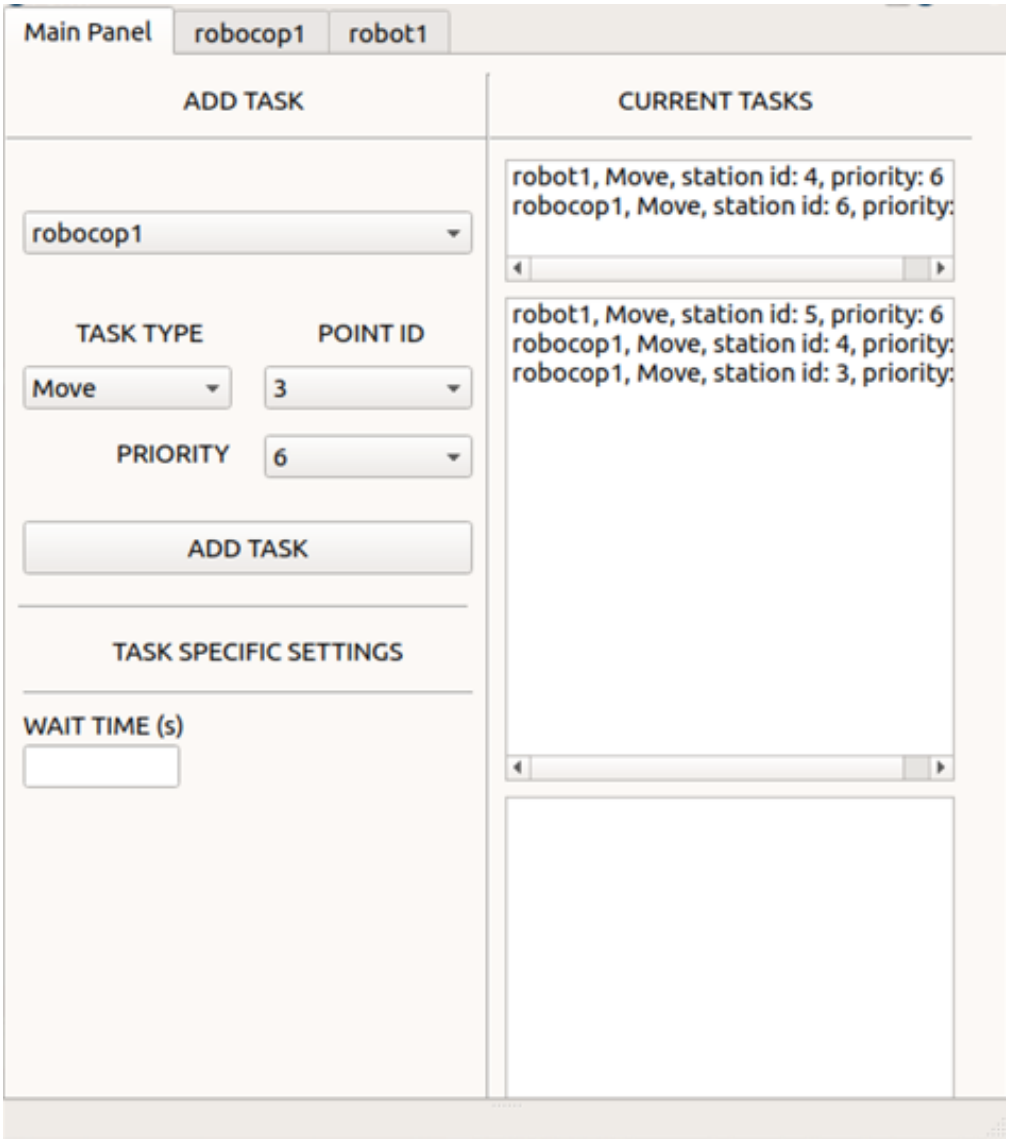

Рисунок 7.1.1. Главное меню графического интерфейса

Пример файла, лежащего в основе графического интерфейса для менеджера задач и использованного в работе, приведен в приложении Г.

Графический интерфейс можно разделить на два основных элемента – главное меню и вкладку робота. Каждый из этих элементов инициализирован как объект QWidget.

Главное меню (рисунок 7.1.1.) содержит форму для заполнения параметров задачи, состоящую из выпадающих списков для выбора имени робота, типа задачи, идентификатора станции, приоритета задачи и (опционально) времени ожидания робота по прибытии на станцию. Под основными параметрами формы расположена кнопка “ADD TASK”, по нажатию на которую происходит дальнейшая обработка данных из формы и отправка задачи роботу.

Выпадающий список представлен объектом QtWidgets.QComboBox. Также главное меню содержит три разделенных списка, в первом из которых отображены выполняемые в данный момент задачи, во втором – задачи, находящихся в очереди на выполнение, в третьем - уже выполненные задачи. Каждая строка в списке содержит следующую информацию: имя робота, на которого назначена задача, тип задачи, идентификатор станции и приоритет данной задачи. Для отображения списков был использован объект QtWidgets.QListWidget.

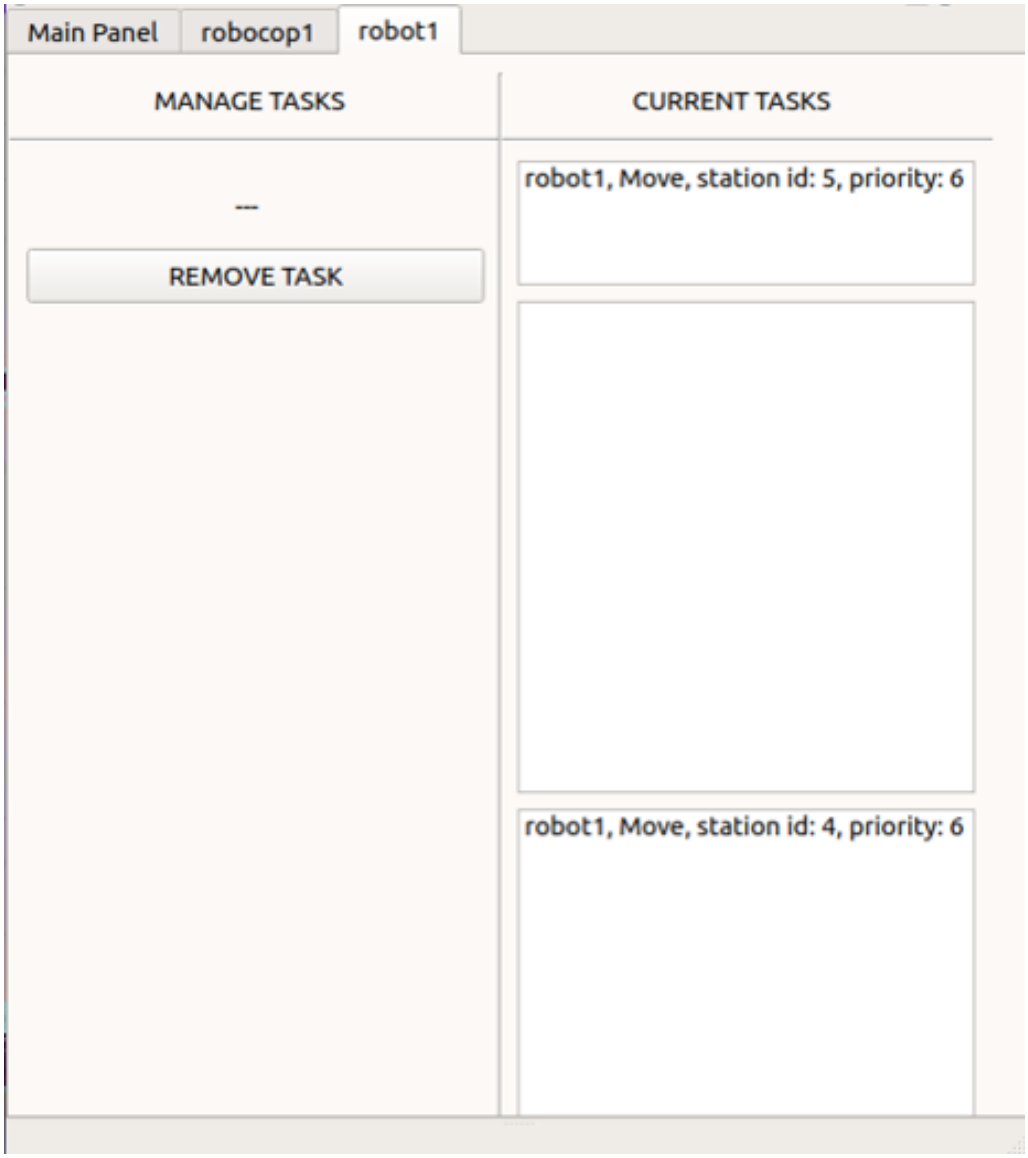

Рисунок 7.1.2. Вкладка робота в графическом интерфейса

Вкладка робота (рисунок 7.1.2.) содержит кнопку для удаления задачи и списки задач, аналогичные тем, что расположены в главном меню, но в них отображаются только те задачи, которые были назначены роботу, которому соответствует вкладка. Для каждого робота, добавленного в параметр “robots” сервера параметров, создается своя вкладка.

8. РАЗРАБОТКА АВТОМАТА СОСТОЯНИЙ

Общее поведение робота определяется конечным автоматом состояний с различными переходами из одного состояния в другое. Структура конечных

автоматов была реализована с использованием ROS пакета smach, бесплатного и доступного для использования в среде ROS (Robot Operating System) [35] [36]. Smach - это пакет, предоставляющий возможность описания архитектуры программы в виде набора состояний, связанных переходами между собой, позволяя описывать сложное поведение роботов в виде конечных автоматов (или контейнеров состояний), определять их иерархию с помощью вложенных конечных автоматов, самоанализа состояний, переходов состояний и передачи данных между ними во время выполнения. Такой подход позволяет полностью определять поведение робота, разделяя его в состояния и управлять состояниями с помощью условных переходов и передачи данных из состояния в состояние во время перехода.

Реализация автомата состояний smach содержится в файле GoalsState.py (приложение А). Для того, чтобы создать новое состояние, нужно создать соответствующий этому состоянию класс, унаследовавшись от класса smach.State. В минимальном виде класс состояния, помимо конструктора init, должен содержать переопределенный метод execute(self, userdata). Этот метод должен содержать логику поведения робота в данном состоянии. В качестве параметра он принимает объект userdata, содержащий данные, переданные в это состояние из предыдущего. Метод должен возвращать строку, в системе smach называемую исходом (англ. outcome). С каждым состоянием может быть связано несколько возможных исходов. Исходом является определяемая пользователем строка, описывающая завершение состояния. Пример набора исходов может выглядеть следующим образом: [“успешно”, “неуспешно”, “отлично”]. Переход к следующему состоянию производится на основе исхода предыдущего состояния, таким образом, исход определяет, в какое состояние нужно перейти.

Состояния являются частью автомата состояний. Для того, чтобы создать автомат состояний, нужно объявить и инициализировать объект smach.StateMachine, в конструктор которого необходимо передать массив

возможных исходов данного автомата состояний (поскольку автомат может быть вложен в другой автомат и так же, как и состояние, должен возвращать определенный исход), и, опционально, список ключей для пользовательских данных. При помощи ключей пользовательских данных осуществляется передача данных между состояниями. Состояние может потребовать некоторые входные данные для выполнения своей работы и/или у него могут быть некоторые выходные данные, которые состояние должно предоставить другим состояниям. Входные и выходные данные состояния называются пользовательскими данными состояния. Входные данные, необходимые для запуска состояния, перечисляются в списке **input_keys**. Таким образом состоянием будет ожидаться, что эти поля будут существовать в пользовательских данных. В методе execute будет передана копия структуры userdata, заполненная в процессе работы предыдущего состояния. Состояние может считывать данные из всех полей пользовательских данных, перечисленных в списке input_keys, но не может выполнять запись ни в одно из этих полей. Список **output_keys** перечисляет все выходные данные, которые предоставляет текущее состояние следующему. Состояние может записывать данные во все поля структуры userdata, которые перечислены в списке output_keys.

После создания объекта smach.StateMachine и определения его исходов и пользовательских данных, в него можно добавить состояния. Это можно сделать, вызвав его метод add(), принимающий строку, соответствующую названию состояния, класс, и список переходов в виде 'исход_состояния':'название_следующего_состояния'. В качестве состояния можно передать другой автомат состояний – в таком случае этот автомат будет называться вложенным. При помощи вложенных автоматов состояния можно создавать иерархию автоматов.

Визуализация автомата состояний, использованного в работе, продемонстрирована на рисунке 8.1. Визуализация получена при помощи ROS пакета smach_viewer [37]. Далее будут рассмотрены реализованные состояния.

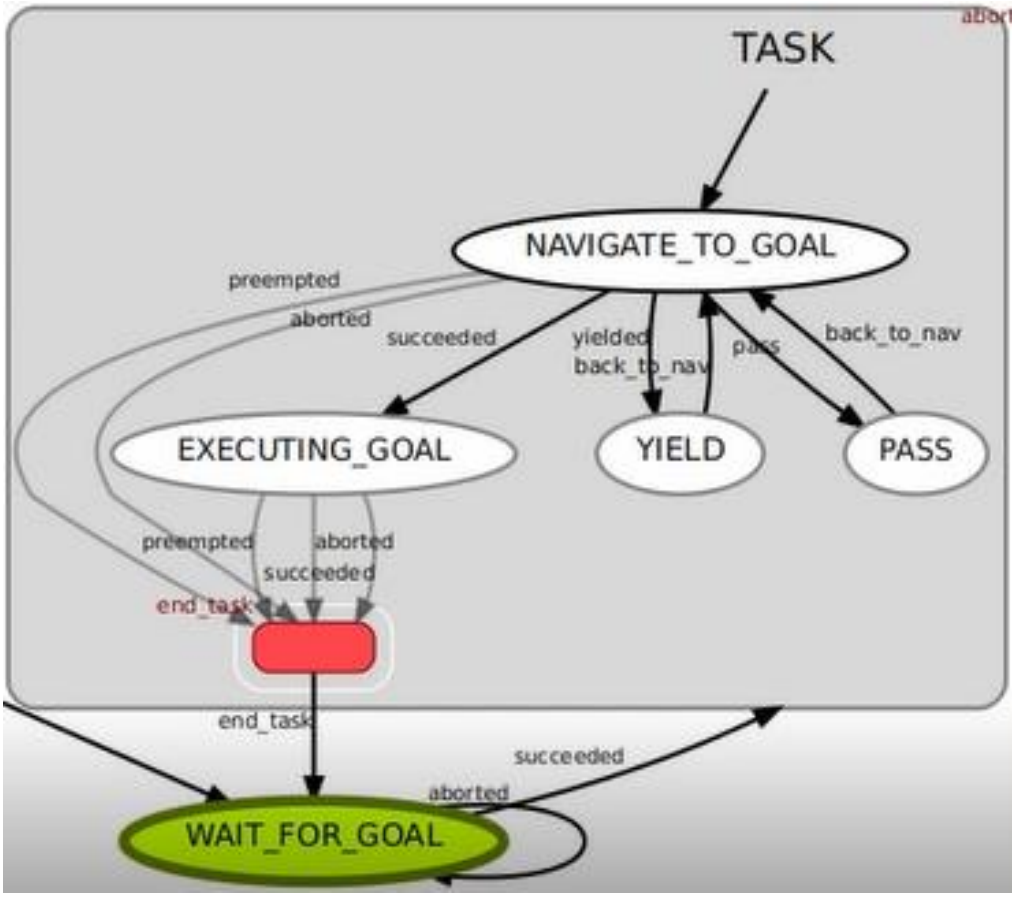

Рисунок 8.1. Визуализация автомата состояний с активным состоянием WAIT_FOR_GOAL (выделено зеленым)

Состояние **WAIT_FOR_GOAL** представляет из себя исходное состояние, в котором находятся бездействующие роботы. Поведение робота в данном состоянии характеризует ожидание новой задачи. Во время инициализации состояния создается подписчик на топик “goals”, после чего робот находится в ожидании получения сообщения посредством этого топика. Топик “goals” используется менеджером задач для передачи назначенной задачи роботу на выполнение. Если в очереди задач робота уже присутствуют невыполненные задачи, то робот сразу приступит к выполнению наиболее приоритетной из них. Вне зависимости от того, как робот получил задачу, состояние завершается исходом “succeeded”, что переводит автомат в

состояние **NAVIGATE_TO_GOAL**. Данное состояние принадлежит (и является единственным состоянием) внешнего автомата, а все последующие состояния принадлежат автомату TASK.

Состояние **NAVIGATE_TO_GOAL** отвечает за навигацию робота к станции задачи. Во время инициализации данного состояния происходит подключение к action-серверу [38] пакета move_base. Также объявляется публикатор в топик “/taskStatus”, предназначенный для оповещения менеджера задач об изменении статуса выполняемой задачи, и подписчик на топик “/tooClose”, предназначенный для получения оповещения о том, что необходимо уступить дорогу другому роботу. Условия правил движения будут подробно рассмотрены в главе 8. В методе execute() класса данного состояния происходит создание и заполнение объекта класса MoveBaseGoal, который в дальнейшем будет передан action-серверу move_base. В данный объект передаются координаты станции задачи и название фрейма, относительно которого были определены эти координаты. Если навигация происходит в рамках выполнения новой задачи, менеджер задач оповещается о том, что задачу необходимо перевести в статус текущей для данного робота, в обратном случае (если продолжается прерванный процесс навигации) статус задачи не меняется. Данные о том, была ли прервана навигация, хранятся в списке пользовательских данных userdata. После того, как объект передан в move_base и робот начал движение к станции, ожидается результат работы action-сервера, отвечающего за навигацию. В зависимости от результата навигации состояние может завершиться со следующими исходами:

- При успешном достижении цели: “succeeded”
- При прерывании процесса навигации (например, робот застрял): “aborted”
- При получении в топик “/taskStatus” сообщения “pass”: “pass”
- При получении в топик “/taskStatus” сообщения “yield”: “yielded”

Исходы “pass” и “yielded” означают, что роботу нужно уступить дорогу другому роботу, тем самым только временно прерывая навигацию. После того, как робот уступит дорогу, активное состояние автомата вернется в NAVIGATE_TO_GOAL, и робот продолжит навигацию к станции. В случае, если состояние завершается с исходом “succeeded”, робот перейдет в состояние **EXECUTING_GOAL**.

Состояние **EXECUTING_GOAL** описывает поведение робота, прибывшего на станцию и выполняющего задачу. В зависимости от выбранного типа задачи и наличия значения опционального параметра wait_time, робот либо сообщит менеджеру, что задача выполнена и перейдет в состояние WAIT_FOR_GOAL, либо будет ожидать на станции в течение времени, определенного параметром wait_time.

В случае, если состояние NAVIGATE_TO_GOAL завершилось с исходом “pass”, робот перейдет в состояние **PASS,** прерывая тем самым процесс навигации, и остановится, пропуская робота, выполняющего задачу с более высоким приоритетом. Поскольку состояние PASS предназначено для того, чтобы позволить проехать другому роботу, не пересекая его маршрут, робот, находящийся в данном состоянии, бездействует и ожидает, пока не перестанет выполняться условие соответствующего правила движения, после чего перейдет обратно в состояние NAVIGATE_TO_GOAL и продолжит навигацию.

Если состояние NAVIGATE_TO_GOAL завершилось с исходом “yield”, то роботу нужно уступить путь другому роботу, съехав с его маршрута, избежав тем самым ситуации, когда два робота пытаются взаимно объехать друг друга. Данному поведению соответствует состояние **YIELD**. Для этого выполняются следующие действия:

- При помощи пакета tf вычисляется точка, лежащая на расстоянии 1.5 метра от пересечения перпендикуляра, опущенного от позиции

уступающего робота к прямой, лежащей по направлению движения робота, которому уступается путь

- Уступающий робот начинает движение к этой точке, по прибытии ожидая, пока робот с более высоким приоритетом не проедет мимо уступающего робота, т.е. пока знак его положения по оси y (относительно уступающего робота) не сменится на противоположный.

Для навигации к точке, куда должен отъехать уступающий робот, так же создается подключение к action-серверу move_base. После вышеописанных действий состояние робота снова изменится на NAVIGATE_TO_GOAL, и робот продолжит навигацию к станции своей задачи.

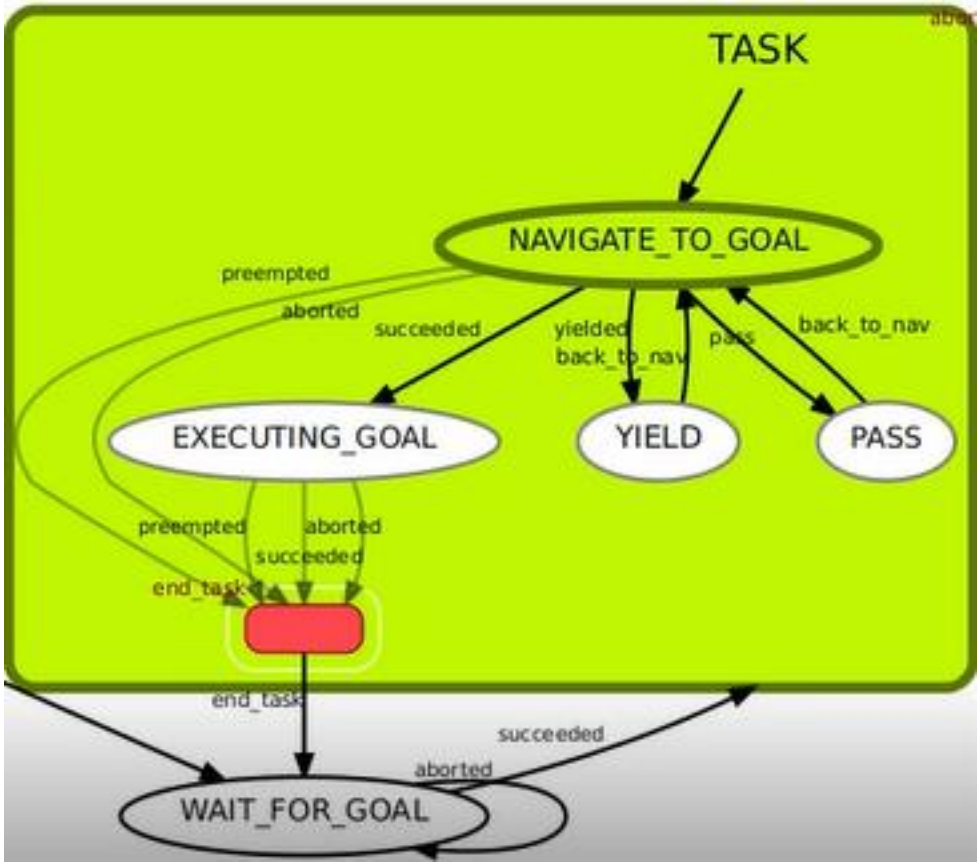

Рисунок 8.2. Визуализация автомата состояний с активным состоянием NAVIGATE_TO_GOAL

Состояния PASS и YIELD сохраняют идентификатор станции, к которой следовал робот, и заново добавляет его в список userdata, чтобы было

возможным продолжить навигацию после выхода из состояния. Для этих состояний единственным исходом является исход “back_to_nav”.

Вложенный автомат TASK всегда завершается с исходом “end_task”, и задача переводится в статус “done”.

9. РАЗРАБОТКА МОДЕЛИ ПРАВИЛ ДВИЖЕНИЯ

Выполняя свои задачи, роботы должны перемещаться от одной станции к другой, обеспечивая максимально свободное перемещение как роботов, выполняющих высокоприоритетные задачи, так и людей (пациентов, гостей и персонала больницы), приоритет которых всегда выше, чем у роботов. Чтобы удовлетворить этим условиям, была разработана модель правил движения роботов, которая реализована в конечном автомате, рассмотренном в разделе 7. В настоящее время реализованы два дополнительных состояния, каждое из которых описывает поведение робота, уступающего место роботу, выполняющему задачу с более высоким приоритетом.

Далее представлен список реализованных и протестированных правил передвижения роботов:

1. Когда два робота приближаются друг к другу, робот с задачей с более низким приоритетом, приблизившись на 4 метра, должен уступить дорогу второму роботу, изменив направление движения на 90 градусов относительно направления движения робота с более высоким приоритетом и отъезжая на 150 см в этом направлении. Считается, что роботы движутся навстречу друг другу, если разница между углом поворота одного робота относительно системы координат карты и углом поворота второго робота относительно той же системы меньше или равна 0,27 радиана, и положение одного робота относительно другого по оси X (относительно системы координат робота) положительно, т. е.

роботы направлены друг к другу. После того, как робот с более высоким приоритетом проедет мимо уступающего робота (т.е. он окажется позади точки, где уступающий робот покинул свой первоначальный маршрут), уступающий робот должен продолжить движение к станции своей задачи. Это правило соответствует состоянию YIELD.

2. Если два робота движутся таким образом, что робот с более низким приоритетом пересекает путь робота с более высоким приоритетом (например, робот с более низким приоритетом выезжает из комнаты в коридор, по которому движется другой робот, пересекая выход из комнаты), робот с более низким приоритетом должен остановиться и дождаться, пока другой робот уйдет. Предполагается, что маршруты роботов пересекаются, когда векторы, образованные двумя точками, где первая точка соответствует координатам текущего положения робота, а вторая расположена на расстоянии 2 метра по направлению движения робота, пересекаются и абсолютное значение угла пересечения находится в диапазоне от 1,47 до 2,87. Робот с более низким приоритетом продолжает движение, когда условие пересечения больше не выполняется. Это правило соответствует состоянию PASS.

Состояние YIELD предназначено для случаев, когда два робота движутся навстречу друг другу, и один из них (выполняющий задачу с более низким приоритетом) должен уступить место роботу с задачей с более высоким приоритетом. Конечный автомат робота переходит в состояние YIELD, когда выполняются условия для первого правила движения. По мере того, как роботы продолжают приближаться друг к другу, их локальные планировщики начинают создавать локальный путь, который рассматривает другого робота как препятствие, что при отсутствии подходящих правил движения может привести к тому, что роботы начнут двигаться параллельно в том же

направлении. избегать друг друга, отклоняясь от своих глобальных путей. Поскольку условия для входа в это состояние подразумевают, что робот находится на пути другого робота, поведение робота в состоянии YIELD реализовано таким образом, чтобы робот отъехал в сторону на достаточное расстояние, чтобы локальный планировщик пути робота с более высоким приоритетом не перестроил свой локальный план пути и, таким образом, существенно не изменил время выполнения задачи этого робота из-за удлиненного маршрута.

Состояние PASS (второе правило) также предназначено для того, чтобы один робот пропустил другого, но, в отличие от состояния YIELD, описанного выше, оно заключается в полной остановке и ожидании робота с более низким приоритетом в условиях, когда его продолжающееся движение будет пересекать локальный путь робота с более высоким приоритетом. Конечный автомат переходит в состояние PASS, когда выполняются условия для второго правила движения. Поведение робота в состоянии PASS - это полная остановка и ожидание, пока робот, которому задан маршрут, не пройдет перед роботом с более низким приоритетом или пока условие перехода в это состояние больше не будет выполняться.

Отслеживание выполнения условий правил выполняется в отдельном узле (приложение Б). Проверка условий выполняется в цикле на основе позиции робота, на котором работает узел, и всех остальных роботов в системе. Когда начинают выполняться условия одного из правил, в узел робота, на котором запущен автомат состояний, отправляется сообщение посредством публикации в топик. Вычисление расстояний между роботами и получение относительных позиций реализовано с использованием ROS пакета tf2 и утилит из класса tf.transformations.

10. ТЕСТИРОВАНИЕ РАБОТЫ СИСТЕМЫ

Для тестирования работы системы была использована виртуальная модель больницы, описанная в разделе 4. В начале тестовых запусков два робота TIAGo Base и один робот Clearpath Ridgeback находятся в разных комнатах на виртуальной карте больницы. На рисунке 9.1 изображена визуализация Rviz [39] [40] исходного положения роботов на карте.

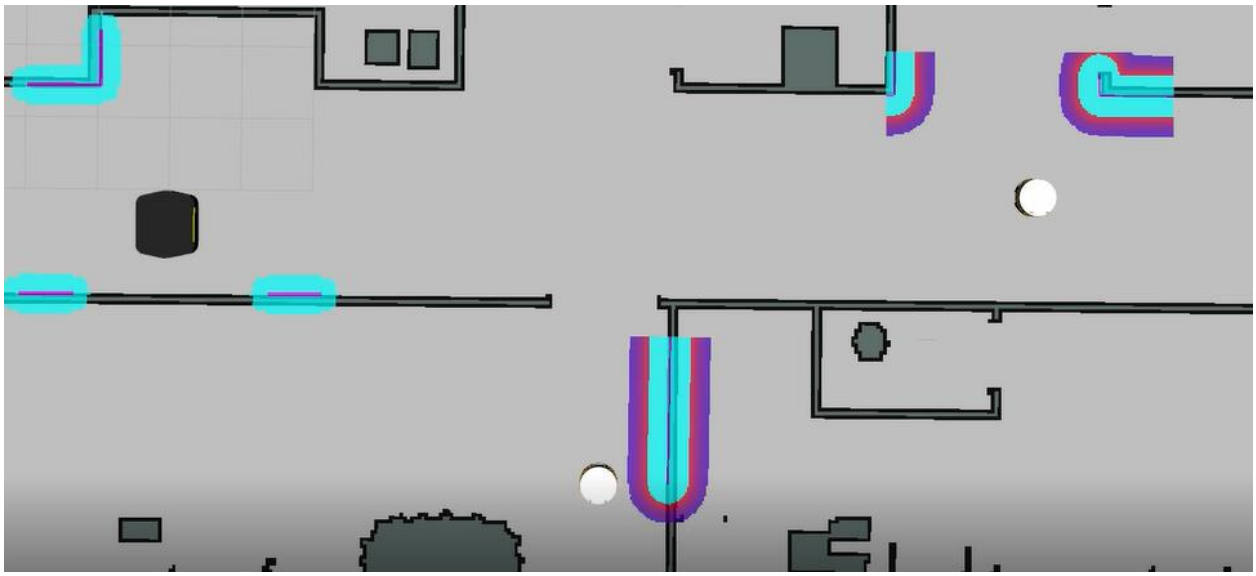

Рисунок 9.1. Исходное положение роботов в визуализации Rviz

Вначале конечные автоматы всех роботов находятся в состоянии WAIT_FOR_GOAL. Затем роботам назначаются их последовательности задач через графический интерфейс диспетчера задач. Задачи назначаются с разными приоритетами таким образом, чтобы одни роботы в процессе навигации были вынуждены уступить дорогу другим, то есть, по крайней мере, хотя бы один раз был осуществлен переход автомата из состояния NAVIGATE_TO_GOAL в состояние YIELD или PASS. После того, как роботы получают задачи, их состояние меняется на NAVIGATE_TO_GOAL вложенного конечного автомата TASK, и они начинают движение к станциям, связанным с их задачами. Во время навигации робот, выполняющий задачу с

более низким приоритетом, встречает робота с более высоким приоритетом. Пример ситуации, в которой выполняется правило 1 (переход в состояние YIELD), визуализированный с помощью пакета rviz, продемонстрирован на рисунке 9.2.

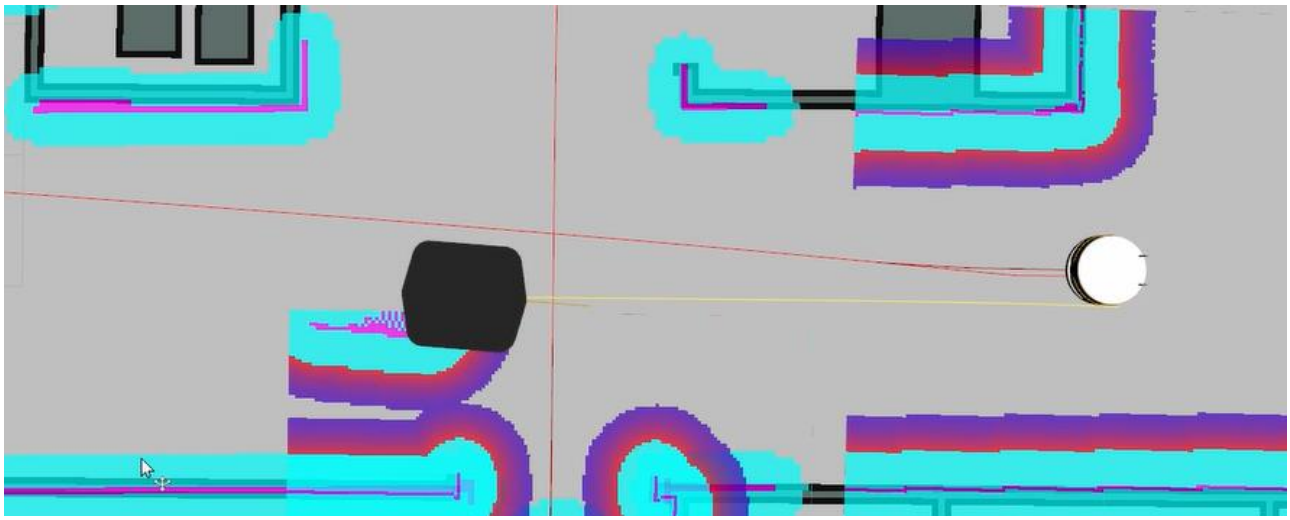

Рисунок 9.2. Робот TIAGo Base (справа) переходит в состояние YIELD

Пути, построенные планировщиками путей, показаны кривыми, желтая из которых обозначает локальный план робота Ridgeback, а красная – робота TIAGo Base. Робот с более низким приоритетом, которым является TIAGo Base, переходит в состояние YIELD и уступает место роботу с более высоким приоритетом, который находится в левой части рисунка. На рисунке 9.3. изображено, как роботы TIAGo Base уступают путь роботу Ridgeback. TIAGo Base, приближающийся к более высокоприоритетному роботу Ridgeback, переходит в состояние YIELD, поскольку условия для первого правила начали выполняться, и поворачивает в сторону, чтобы пропустить его, таким образом локальный план (желтая линия) робота Ridgeback не меняется.

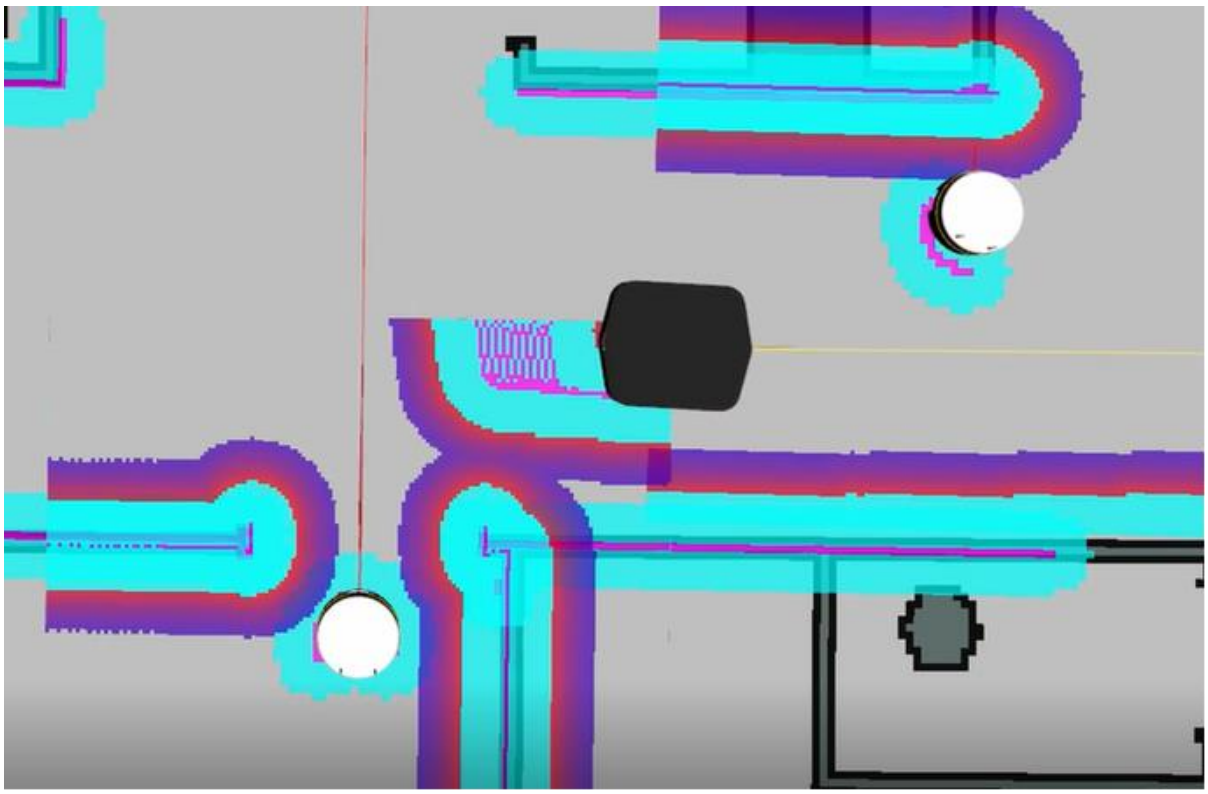

Рисунок 9.3. Робот TIAGo Base (справа) в состоянии YIELD

Локальный путь TIAGo Base (красная линия над TIAGo Base в правой части рисунка 9.3.) изменился в соответствии с правилом YIELD. После того, как робот с более высоким приоритетом прошел достаточное расстояние, состояние уступающего робота переходит обратно в NAVIGATE_TO_GOAL, и робот продолжает следовать к станции. Эксперимент продолжается до тех пор, пока все роботы не выполнят свои задачи и не вернутся в исходное состояние WAIT_FOR_GOAL.

ЗАКЛЮЧЕНИЕ

В данной работе представлена роботизированная система для контроля роботов, выполняющих задачи в больнице, передвигаясь в соответствии с моделью правил движения. Автономные роботы TIAGo Base и Clearpath Ridgeback перемещаются от станции к станции в симуляторе больницы, выполняя задачи с приоритетами, которые назначаются через графический интерфейс, и уступают дорогу другим роботам, выполняющим задачи с более высоким приоритетом. Использование правил движения с приоритетами во время навигации сокращает общую длину пути, пройденного роботами, выполняющими высокоприоритетные задачи, поскольку им не требуется объезжать других роботов, выполняющих задачи с более низким приоритетом.

В ходе выполнения работы были настроены модули навигации для используемых роботов, разработан конечный автомат состояний на основе библиотеки smach, моделирующий состояния для выполнения роботами задач и состояния, соответствующие модели правил движения. Также был разработан менеджер задач, позволяющий назначать роботам навигационные задачи, выполняемые в порядке приоритетов, и графический интерфейс для взаимодействия с ним. Таким образом, были выполнены все поставленные задачи.

Система была разработана для использования в больницах, и при дальнейших улучшениях она может применяться в других окружениях, где также возникают проблемы, связанные с транспортировкой предметов в помещениях.

Исходный код и файл данной ВКР опубликован в репозитории ЛИРС на платформе GitLab: https://gitlab.com/LIRS_Projects/hospital-simulation.

СПИСОК ЛИТЕРАТУРЫ

1. Magid E. et al. Automating pandemic mitigation //Advanced Robotics. – 2021. – С. 1-18.
2. Fragapane G. I. et al. An agent-based simulation approach to model hospital logistics //Int J Simul Model. – 2019. – Т. 18. – №. 4. – С. 654-665.
3. Galin R., Meshcheryakov R. Automation and robotics in the context of Industry 4.0: the shift to collaborative robots //IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2019. – Т. 537. – №. 3. – С. 032073.
4. Sagitov A. et al. Design of simple one-arm surgical robot for minimally invasive surgery //2019 12th International Conference on Developments in eSystems Engineering (DeSE). – IEEE, 2019. – С. 500-503.
5. Robot Operating System (ROS) [Электронный ресурс] / ROS Melodic documentation — Режим доступа — URL: http://wiki.ros.org/ (дата обращения: 11.06.2021).
6. Gazebo simulation [Электронный ресурс] / Gazebo 9.0 documentation — Режим доступа — URL: http://gazebosim.org/ (дата обращения: 11.06.2021).
7. TIAGo Base [Электронный ресурс] / TIAGo Base overview — Режим доступа — URL: https://pal-robotics.com/robots/tiago-base/ (дата обращения: 11.06.2021).
8. TIAGo Base [Электронный ресурс] / ROS Melodic, TIAGo Base package — Режим доступа — URL: http://wiki.ros.org/Robots/PMB-2 (дата обращения: 11.06.2021).
9. Clearpath Ridgeback [Электронный ресурс] / Clearpath Ridgeback overview — Режим доступа — URL: https://clearpathrobotics.com/ridgeback-indoor-robot-platform (дата обращения: 11.06.2021).

10. Clearpath Ridgeback [Электронный ресурс] / Clearpath Ridgeback ROS page — Режим доступа — URL: http://wiki.ros.org/Robots/Ridgeback (дата обращения: 11.06.2021).
11. Takahashi M. et al. Developing a mobile robot for transport applications in the hospital domain //Robotics and Autonomous Systems. – 2010. – Т. 58. – №. 7. – С. 889-899.
12. Ge S. S., Cui Y. J. Dynamic motion planning for mobile robots using potential field method //Autonomous robots. – 2002. – Т. 13. – №. 3. – С. 207-222.
13. Zakharov K., Saveliev A., Sivchenko O. Energy-Efficient Path Planning Algorithm on Three-Dimensional Large-Scale Terrain Maps for Mobile Robots //International Conference on Interactive Collaborative Robotics. – Springer, Cham, 2020. – С. 319-330.
14. Fung W. K. et al. Development of a hospital service robot for transporting task //IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003. – IEEE, 2003. – Т. 1. – С. 628-633.
15. Klee S. D. et al. Multi-robot task acquisition through sparse coordination //2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – IEEE, 2015. – С. 2823-2828.
16. Melo F. S., Veloso M. Decentralized MDPs with sparse interactions //Artificial Intelligence. – 2011. – Т. 175. – №. 11. – С. 1757-1789.
17. Sun Y., Coltin B., Veloso M. Interruptible autonomy: Towards dialog-based robot task management //Proc. Workshop 27th AAAI Conf. Artificial Intelligence. – 2013.
18. AWS Robomaker [Электронный ресурс] / AWS Robomaker Hospital World —Режим доступа — URL: https://github.com/aws-robotics/aws-robomaker-hospital-world (дата обращения: 11.06.2021).

19. gmapping [Электронный ресурс] / Gmapping ROS documentation — Режим доступа — URL: http://wiki.ros.org/gmapping (дата обращения: 11.06.2021).
20. Wang H., Yu Y., Yuan Q. Application of Dijkstra algorithm in robot path-planning //2011 second international conference on mechanic automation and control engineering. – IEEE, 2011. – С. 1067-1069.
21. amcl [Электронный ресурс] / Amcl ROS documentation —Режим доступа — URL: http://wiki.ros.org/amcl (дата обращения: 11.06.2021).
22. Lavrenov R. O. et al. Development and implementation of spline-based path planning algorithm in ROS/Gazebo environment //Trudy SPIIRAN. – 2019. – Т. 18. – №. 1. – С. 57-84.
23. Magid E. et al. Combining Voronoi graph and spline-based approaches for a mobile robot path planning //International Conference on Informatics in Control, Automation and Robotics. – Springer, Cham, 2017. – С. 475-496.
24. tf [Электронный ресурс] / Tf ROS documentation —Режим доступа — URL: http://wiki.ros.org/tf (дата обращения: 11.06.2021).
25. tf2 [Электронный ресурс] / Tf2 ROS documentation —Режим доступа — URL: http://wiki.ros.org/tf2 (дата обращения: 11.06.2021).
26. move_base [Электронный ресурс] / move_base ROS documentation — Режим доступа — URL: http://wiki.ros.org/move_base (дата обращения: 11.06.2021).
27. map_server [Электронный ресурс] / map_server ROS documentation — Режим доступа — URL: http://wiki.ros.org/map_server (дата обращения: 11.06.2021).
28. Rösmann C. et al. Trajectory modification considering dynamic constraints of autonomous robots //ROBOTIK 2012; 7th German Conference on Robotics. – VDE, 2012. – С. 1-6.
29. Rösmann C. et al. Efficient trajectory optimization using a sparse model //2013 European Conference on Mobile Robots. – IEEE, 2013. – С. 138-143.

30. teb_local_planner [Электронный ресурс] / teb_local_planner ROS documentation —Режим доступа — URL: http://wiki.ros.org/teb_local_planner (дата обращения: 11.06.2021).
31. Quinlan S., Khatib O. Elastic bands: Connecting path planning and control //[1993] Proceedings IEEE International Conference on Robotics and Automation. – IEEE, 1993. – С. 802-807.
32. costmap_2d [Электронный ресурс] / costmap_2d ROS documentation — Режим доступа — URL: http://wiki.ros.org/costmap_2d (дата обращения: 11.06.2021).
33. rqt [Электронный ресурс] / rqt ROS documentation —Режим доступа — URL: http://wiki.ros.org/rqt (дата обращения: 11.06.2021).
34. Qt Designer [Электронный ресурс] / Qt Designer documentation —Режим доступа — URL: https://doc.qt.io/qt-5/qtdesigner-manual.html (дата обращения: 11.06.2021).
35. Bohren J., Cousins S. The smach high-level executive [ros news] //IEEE Robotics & Automation Magazine. – 2010. – Т. 17. – №. 4. – С. 18-20.
36. smach [Электронный ресурс] / smach ROS documentation — Режим доступа — URL: http://wiki.ros.org/smach (дата обращения: 11.06.2021).
37. smach_viewer [Электронный ресурс] / smach_viewer ROS documentation —Режим доступа — URL: http://wiki.ros.org/smach_viewer (дата обращения: 11.06.2021).
38. actionlib [Электронный ресурс] / actionlib ROS documentation — Режим доступа — URL: http://wiki.ros.org/actionlib (дата обращения: 11.06.2021).
39. Kam H. R. et al. Rviz: a toolkit for real domain data visualization //Telecommunication Systems. – 2015. – Т. 60. – №. 2. – С. 337-345.
40. rviz [Электронный ресурс] / rviz ROS documentation — Режим доступа — URL: http://wiki.ros.org/rviz (дата обращения: 11.06.2021).

ПРИЛОЖЕНИЕ А

Автомат состояний и их имплементация
Файл GoalsState.py:

```
#!/usr/bin/env python

import rospy
import smach
import actionlib
import move_base_msgs
from Task import Task
import heapq as heap
import smach_ros
import tf2_ros
import geometry_msgs.msg
import numpy
import tf
import math

from math import sin, cos, radians, pi, sqrt
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal, MoveBaseResult
from std_msgs.msg import String
from smach_ros import SimpleActionState
from actionlib import GoalStatus
from navgoals_smach.msg import TaskMsg
from geometry_msgs.msg import PoseStamped, PoseWithCovarianceStamped
from tf.transformations import euler_from_quaternion, quaternion_from_euler
from math import sin, cos, radians, pi, sqrt

waypoint = {}
currentGoalList = []
allGoalsList = []

class WaitForGoal(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['succeeded','aborted'],output_keys=['task_id_output'])

        self.wfgSubscriber = rospy.Subscriber("/goals", TaskMsg, self.wfgCb)
        self.robot_name = rospy.get_param(rospy.get_name() + "/robot_name")


    def wfgCb(self, data):
        if (self.robot_name != data.robotName.decode('utf-8')):
            pass
        else:
            rospy.loginfo(self.robot_name + "/WaitForGoal state: Get goal!")
            self.isWaiting = False


    def execute(self, userdata):
        userdata.task_id_output = None
        self.isWaiting = True
        self.rate = rospy.Rate(10)
        if len(currentGoalList) == 0:
            rospy.loginfo(self.robot_name  + "/WaitForGoal state: Waiting for goals")
            while self.isWaiting:
                self.rate.sleep()
            return 'succeeded'
        else:
            return 'succeeded'
```

```
class Nav2Goal(smach.State):

    def __init__(self):
        smach.State.__init__(self, outcomes=['succeeded','aborted','preempted','yielded','pass'],
                             output_keys=['task_id_output'],
                             input_keys=['task_id_input'])

        self.robot_name = rospy.get_param(rospy.get_name() + "/robot_name")
        if (self.robot_name != "dingo"):
            self.move_base = actionlib.SimpleActionClient(self.robot_name  + "/move_base", MoveBaseAction)
        else:
            self.move_base = actionlib.SimpleActionClient("/move_base", MoveBaseAction)
        self.move_base.wait_for_server(rospy.Duration(60))
        rospy.loginfo(self.robot_name  + "/Nav2Goal state: Connected to move_base action server")
        self.sendTaskStatusPub = rospy.Publisher('/taskStatus', String, queue_size=10)
        self.tooCloseReceiver = rospy.Subscriber(self.robot_name + '/tooClose', String, self.yieldCb)
        self.readyToYield = False
        self.readyToPass = False
        self.yields = True


    def yieldCb(self, data):


        rospy.loginfo(self.robot_name + "/Nav2Goal state: mb state " + str(self.move_base.get_state()))
        if (data.data == 'pass'):
            rospy.loginfo(self.robot_name + "/Nav2Goal state: got " + data.data)
            rospy.loginfo(self.robot_name + '/Nav2Goal state: state pre pass: ' + str(self.move_base.get_state()))
            self.readyToYield = False
            self.readyToPass = True
            if self.move_base.get_state() == 1:
                rospy.loginfo(self.robot_name + '/Nav2Goal: zaletau pass')
                self.move_base.cancel_goal()
                self.tooCloseReceiver.unregister()

        elif (data.data == 'yield' and self.yields):
            rospy.loginfo(self.robot_name + "/Nav2Goal state: got " + data.data)
            self.readyToYield = True
            self.yields = False
            if self.move_base.get_state() == 1:
                self.move_base.cancel_goal()
                self.tooCloseReceiver.unregister()

            rospy.loginfo(self.robot_name + "/Nav2Goal state: mb state post pass: " + str(self.move_base.get_state()))

        else:
            pass


    def execute(self, userdata):
        self.goal = MoveBaseGoal()
        self.goal.target_pose.header.frame_id = 'map'
        rospy.loginfo(self.robot_name + "Nav2Goal state: executing navgoal with userdata: " +
str(userdata.task_id_input))
        if userdata.task_id_input is not None:
            rospy.loginfo(self.robot_name + "/Nav2Goal state: continue task")
            self.currTask = next((i for i in allGoalsList if i.id == userdata.task_id_input), None)
            if self.currTask is None: raise NameError("Cannot find task by specified id")
            userdata.task_id_output = self.currTask.id
            rospy.loginfo(self.robot_name + "/Nav2Goal state: continue goal id:")
            rospy.loginfo(self.currTask.goalId)
            self.goal.target_pose.pose.position.x = float(waypoint[self.currTask.goalId][0])
```

```
            self.goal.target_pose.pose.position.y = float(waypoint[self.currTask.goalId][1])
            self.goal.target_pose.pose.orientation.w = float(waypoint[self.currTask.goalId][2].rstrip())
        else:
            self.currTask = currentGoalList[0]
            userdata.task_id_output = self.currTask.id
            del currentGoalList[0]
            self.sendTaskStatusPub.publish(str(self.currTask.id) + ' current')
            if not self.currTask.goalId: return 'succeeded'
            rospy.loginfo(self.robot_name + "/Nav2Goal state: next goal id:")
            rospy.loginfo(self.currTask.goalId)
            rospy.loginfo(self.robot_name  + "/Nav2Goal state: next goal coords:")
            rospy.loginfo(waypoint[self.currTask.goalId][0])
            rospy.loginfo(waypoint[self.currTask.goalId][1])
            rospy.loginfo(waypoint[self.currTask.goalId][2])
            self.goal.target_pose.pose.position.x = float(waypoint[self.currTask.goalId][0])
            self.goal.target_pose.pose.position.y = float(waypoint[self.currTask.goalId][1])
            self.goal.target_pose.pose.orientation.w = float(waypoint[self.currTask.goalId][2].rstrip())

        self.move_base.send_goal(self.goal)


        # if self.preempt_requested():
        #     self.move_base.cancel_goal()
        #     self.service_preempt()
        #     self.sendTaskStatusPub.publish(str(self.currTask.id) + ' done')
        #     return 'preemted'
        if userdata.task_id_input is not None:
            tfBuffer = tf2_ros.Buffer()
            listener = tf2_ros.TransformListener(tfBuffer)


            rospy.loginfo(self.robot_name  + "/Nav2Goal state: Waiting to get goal done...")
            while True:
                finished =self.move_base.wait_for_result(rospy.Duration(2))
                if finished:
                    break
                # TODO: Looks like kostyl'
                trans = tfBuffer.lookup_transform('ridgeback/base_link', 'pmb2/base_footprint', rospy.Time())
                x = trans.transform.translation.x
                y = trans.transform.translation.y
                dist = math.sqrt((x ** 2) + (y ** 2))
                if (dist >= 5.0):
                    self.tooCloseReceiver = rospy.Subscriber(self.robot_name + '/tooClose', String, self.yieldCb)
                    break


        finished = self.move_base.wait_for_result()

        if finished:
            rospy.loginfo(self.robot_name  + "/Nav2Goal state: yield: " + str(self.readyToYield) + "finished: " +
str(finished))
            state = self.move_base.get_state()
            if state == GoalStatus.SUCCEEDED:
                rospy.loginfo(self.robot_name  + "/Nav2Goal state: Goal done!")
                return 'succeeded'
            else:
                if self.readyToPass:
                    rospy.loginfo(self.robot_name + "/Nav2Goal state: passing way to another robot...")
                    return 'pass'
                elif self.readyToYield:
                    rospy.loginfo(self.robot_name + "/Nav2Goal state: giving way to another robot...")
                    return 'yielded'
                else:
```

```
                rospy.loginfo(self.robot_name  + "/Nav2Goal state: navigation not succeeded")
                self.sendTaskStatusPub.publish(str(self.currTask.id) + ' done')
                return 'aborted'
        else:
            rospy.loginfo(self.robot_name + "/Nav2Goal state: lol how")
            return 'aborted'


class ExecuteGoal(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['succeeded','aborted','preempted'],
                             input_keys=['task_id_input'])
        self.robot_name = rospy.get_param(rospy.get_name() + "/robot_name")
        self.sendTaskStatusPub = rospy.Publisher('/taskStatus', String, queue_size=10)


    def execute(self, userdata):
        self.currTask = next((i for i in allGoalsList if i.id == userdata.task_id_input), None)

        if self.currTask is None: raise NameError("Cannot find task by specified id")
        if self.currTask.waitTime: rospy.sleep(int(self.currTask.waitTime))

        self.sendTaskStatusPub.publish(str(self.currTask.id) + ' done')

        return 'succeeded'


class Yield(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['back_to_nav'],
                             input_keys=['task_id_input'],
                             output_keys=['task_id_output'])

        self.robot_name = rospy.get_param(rospy.get_name() + "/robot_name")
        self.tfBuffer = tf2_ros.Buffer()
        self.listener = tf2_ros.TransformListener(self.tfBuffer)
        if (self.robot_name != "dingo"):
            self.move_base = actionlib.SimpleActionClient(self.robot_name  + "/move_base", MoveBaseAction)
        else:
            self.move_base = actionlib.SimpleActionClient("/move_base", MoveBaseAction)
        self.move_base.wait_for_server(rospy.Duration(60))
        self.testPub0 = rospy.Publisher(self.robot_name +  '/test0', PoseStamped, queue_size=10)
        self.testPubWrtAnother = rospy.Publisher(self.robot_name +  '/test1', PoseStamped, queue_size=10)
        self.testPubFin = rospy.Publisher(self.robot_name + '/test2', PoseStamped, queue_size=10)
        # self.tooCloseReceiver = rospy.Subscriber(self.robot_name + '/tooClose', String, self.yieldCb)
        # self.readyToHitTheRoad = False

        # def yieldCb(self, data):
        #     if (data == "go"):
        #         rospy.loginfo(self.robot_name + "/Yield state: continue going")
        #         self.readyToHitTheRoad = True
        #         self.move_base.cancel_goal()
        #     else:
        #         pass

    def point_pos(self, x0, y0, d, theta_rad):
        # theta_rad = pi/2 - theta_rad
        return x0 + d*cos(theta_rad), y0 + d*sin(theta_rad)

    def execute(self, userdata):
        rospy.loginfo(self.robot_name + "/yield state: yielding")
        try:
            another_robot = rospy.get_param(self.robot_name + '/yielding_to')
```

```
        another_base = '/base_footprint'
        self_base = '/base_footprint'
        if another_robot == 'ridgeback': another_base = '/base_link'
        if self.robot_name == 'ridgeback': self_base = '/base_link'
        self.goal = MoveBaseGoal()
        self.goal.target_pose.header.frame_id = 'map'
        # amcl_pose = rospy.wait_for_message(self.robot_name + '/amcl_pose')
        trans0 = self.tfBuffer.lookup_transform('map', another_robot + another_base, rospy.Time())
        trans1 = self.tfBuffer.lookup_transform('map', self.robot_name + self_base, rospy.Time())
        x0 = trans0.transform.translation.x
        y0 = trans0.transform.translation.y
        qx0 = trans0.transform.rotation.x
        qy0 = trans0.transform.rotation.y
        qz0 = trans0.transform.rotation.z
        qw0 = trans0.transform.rotation.w

        qx1 = trans1.transform.rotation.x
        qy1 = trans1.transform.rotation.y
        qz1 = trans1.transform.rotation.z
        qw1 = trans1.transform.rotation.w
        # x0 = amcl_pose.pose.pose.position.x
        # y0 = amcl_pose.pose.pose.position.y
        # theta0 = rospy.tf2_ros.getYaw(amcl_pose.PoseWithCovarianceStamped.orientation)
        # theta0 = rospy.tf2_ros.getYaw(amcl_pose.PoseWithCovarianceStamped.orientation)
        # theta0 = rospy.tf2_ros.getYaw(amcl_pose.PoseWithCovarianceStamped.orientation)
        quat = [qx0, qy0, qz0, qw0]
        (r, p, theta0) = euler_from_quaternion(quat)
        quat1 = [qx1, qy1, qz1, qw1]
        (r, p, theta1) = euler_from_quaternion(quat1)

        rospy.loginfo(" theta0 = " + str(theta0) + " theta1 = " + str(theta1))

        trans_between_robots = self.tfBuffer.lookup_transform(another_robot + another_base, self.robot_name +
self_base, rospy.Time())
        x_bet = trans_between_robots.transform.translation.x
        y_bet = trans_between_robots.transform.translation.y
        dist = sqrt((x_bet ** 2) + (y_bet ** 2))

        point_wrt_another_x, point_wrt_another_y = self.point_pos(x0, y0, dist, theta0)
        point_to_destination_x, point_to_destination_y = self.point_pos(point_wrt_another_x, point_wrt_another_y,
1.5, 0)

        # debug
        point0 = PoseStamped()
        point0.header.frame_id = "map"
        point0.pose.position.x = x0
        point0.pose.position.y = y0
        point0.pose.orientation.x = qx0
        point0.pose.orientation.y = qy0
        point0.pose.orientation.z = qz0
        point0.pose.orientation.w = qw0

        point1 = PoseStamped()
        point1.header.frame_id = "map"
        point1.pose.position.x = point_wrt_another_x
        point1.pose.position.y = point_wrt_another_y

        point2 = PoseStamped()
        point2.header.frame_id = "map"
        point2.pose.position.x = point_to_destination_x
        point2.pose.position.y = point_to_destination_y

        self.testPub0.publish(point0)
```

```
            self.testPubWrtAnother.publish(point1)
            self.testPubFin.publish(point2)

            self.goal.target_pose.pose.position.x = float(point_to_destination_x)
            self.goal.target_pose.pose.position.y = float(point_to_destination_y)
            self.goal.target_pose.pose.orientation.w = float(1.0)

            rospy.loginfo("yaw = " + str(theta0))
            # rospy.loginfo(self.robot_name + "/yield state: x0 y0 "
            #             + str(x0) + " " + str(y0)
            #             + " x_bet " + str(x_bet) + " y_bet " + str(y_bet)
            #             + " x_bet " + str(x_bet) + " y_bet " + str(y_bet)
            #             + " x_bet " + str(x_bet) + " y_bet " + str(y_bet)
            #             + " point_wrt_another_x " + str(point_wrt_another_x) + " point_wrt_another_y " +
str(point_wrt_another_y)
            #             + " point_to_destination_x " + str(point_to_destination_x) + " point_to_destination_y " +
str(point_to_destination_y))

            self.move_base.send_goal(self.goal)
        except rospy.ROSException:
            rospy.loginfo(self.robot_name + '/yield state: timemout exceed, pass')
            pass

        self.move_base.wait_for_result(rospy.Duration(4))

        trans_between_robots0 = self.tfBuffer.lookup_transform(self.robot_name + self_base, another_robot +
another_base, rospy.Time())
        y_bet_init = trans_between_robots0.transform.translation.y
        rospy.loginfo(self.robot_name + '/yield: x_bet_init: ' + str(y_bet_init))

        while True:
            # try:
            #     res = rospy.wait_for_message(self.robot_name + '/tooClose', String, 1)
            #     rospy.loginfo(self.robot_name + "/yield state: get " + str(res.data))
            # except:
            #     continue

            trans_between_robots = self.tfBuffer.lookup_transform(self.robot_name + self_base, another_robot +
another_base, rospy.Time())
            x_bet = trans_between_robots.transform.translation.x
            y_bet = trans_between_robots.transform.translation.y
            postDist = sqrt((x_bet ** 2) + (y_bet ** 2))
            rospy.loginfo(self.robot_name + "/yield state: postDist = " + str(postDist))

            rospy.loginfo("X: " + str(x_bet) + " Y: " + str(y_bet))


            if ((y_bet_init > 0 and y_bet < 0) or
                (y_bet_init < 0 and y_bet > 0) or
                (postDist >= 2)):
                userdata.task_id_output = userdata.task_id_input
                rospy.loginfo(self.robot_name + "/yield: back to nav with goal id " + str(userdata.task_id_input))
                if self.move_base.get_state() != 2:
                    self.move_base.cancel_goal()
                return "back_to_nav"

            # rospy.loginfo(self.robot_name + "/yield: prev: x_bet: " + str(x_bet) + " y_bet: " + str(y_bet))

            # if (str(res.data) == "yield"):
            #     pass
            # elif (str(res.data) == "go"):
            #     userdata.task_id_output = userdata.task_id_input
            #     rospy.loginfo(self.robot_name + "/yield: back to nav with goal id " + str(userdata.task_id_input))
```

```
            #     if self.move_base.get_state() != 2:
            #         self.move_base.cancel_goal()
            #     return "back_to_nav"
            # else:
            #     rospy.loginfo(self.robot_name + "/yield state: unknown command: " + str(res.data))


class Pass(smach.State):
    def __init__(self):
        smach.State.__init__(self, outcomes=['back_to_nav'],
                             input_keys=['task_id_input'],
                             output_keys=['task_id_output'])

        self.robot_name = rospy.get_param(rospy.get_name() + "/robot_name")
        self.tfBuffer = tf2_ros.Buffer()
        self.listener = tf2_ros.TransformListener(self.tfBuffer)
        if (self.robot_name != "dingo"):
            self.move_base = actionlib.SimpleActionClient(self.robot_name  + "/move_base", MoveBaseAction)
        else:
            self.move_base = actionlib.SimpleActionClient("/move_base", MoveBaseAction)
        self.move_base.wait_for_server(rospy.Duration(60))



    def execute(self, userdata):
        rospy.sleep(6)
        userdata.task_id_output = userdata.task_id_input
        rospy.loginfo(self.robot_name + "/yield: back to nav with goal id " + str(userdata.task_id_input))
        if self.move_base.get_state() != 2:
            self.move_base.cancel_goal()
        return "back_to_nav"


def fillStationMap():
    with open("/home/ruslan/hospital_ws/src/goal_scenarios/stations.txt", "r") as f:
        for line in f:
            data = line.split(" ")
            key, value = data[0], data[1::]
            waypoint[key] = value


def goalsCb(data):
    taskId = data.taskId
    goalId = data.goalId.decode('utf-8')
    robotName = data.robotName.decode('utf-8')
    priority = data.priority
    taskType = data.taskType
    waitTime = data.waitTime
    if (rospy.get_param(rospy.get_name() + "/robot_name") != robotName):
        rospy.loginfo(rospy.get_param(rospy.get_name() + "/robot_name") + "/goalsCb: Good luck, " + robotName +
"!")
    else:
        rospy.loginfo(rospy.get_param(rospy.get_name() + "/robot_name") + "/goalsCb: Get goal!")
        task = Task(goalId, priority, robotName, taskType, taskId, waitTime)
        currentGoalList.append(task)
        allGoalsList.append(task)
        # Sort by priority
        currentGoalList.sort(key=lambda task: task.priority, reverse=True)


def main():
    rospy.init_node("mb_machine")
    rospy.Subscriber("/goals", TaskMsg, goalsCb)
```

```
    sm_nav = smach.StateMachine(outcomes=['aborted', 'completed', 'preempted'], output_keys=['nav_id_data_cont'])

    with sm_nav:
        smach.StateMachine.add('WAIT_FOR_GOAL',
                               WaitForGoal(),
                               transitions={'succeeded':'TASK',
                                            'aborted':'WAIT_FOR_GOAL'},
                               remapping={'task_id_output':'nav_id_data_cont'})

        sm_task = smach.StateMachine(outcomes=['end_task'],input_keys=['nav_id_data_cont'])

        with sm_task:
            smach.StateMachine.add('NAVIGATE_TO_GOAL',
                                   Nav2Goal(),
                                   transitions={'succeeded':'EXECUTING_GOAL',
                                                'aborted':'end_task',
                                                'preempted':'end_task',
                                                'yielded':'YIELD',
                                                'pass':'PASS'},
                                   remapping={'task_id_output':'nav_id_data',
                                                'task_id_input':'nav_id_data_cont'})

            smach.StateMachine.add('EXECUTING_GOAL',
                                   ExecuteGoal(),
                                   transitions={'succeeded':'end_task',
                                                'aborted':'end_task',
                                                'preempted':'end_task'},
                                   remapping={'task_id_input':'nav_id_data'})

            smach.StateMachine.add('YIELD',
                                   Yield(),
                                   transitions={'back_to_nav':'NAVIGATE_TO_GOAL'},
                                   remapping={'task_id_input':'nav_id_data',
                                              'task_id_output':'nav_id_data_cont'})

            smach.StateMachine.add('PASS',
                                   Pass(),
                                   transitions={'back_to_nav':'NAVIGATE_TO_GOAL'},
                                   remapping={'task_id_input':'nav_id_data',
                                              'task_id_output':'nav_id_data_cont'})

        smach.StateMachine.add('TASK',
                               sm_task,
                               transitions={'end_task':'WAIT_FOR_GOAL'})

    sis = smach_ros.IntrospectionServer('server_name', sm_nav, rospy.get_param(rospy.get_name() + "/robot_name")
+ '/SM_ROOT')
    sis.start()

    outcome = sm_nav.execute()
    rospy.loginfo(rospy.get_param(rospy.get_name() + "/robot_name") + "/mb_machine: Result: %s", outcome)

    rospy.spin()


if __name__ == "__main__":
    fillStationMap()
    main()
```

ПРИЛОЖЕНИЕ Б

Контроль условий, при которых выполняются правила движения
Файл Arbitrator.py:

```
#!/usr/bin/env python
import rospy
import tf2_ros
import geometry_msgs.msg
import math
import logging
from std_msgs.msg import String
from tf.transformations import euler_from_quaternion
from math import sin, cos, radians, pi, sqrt


def point_pos(x0, y0, d, theta_rad):
    return x0 + d*cos(theta_rad), y0 + d*sin(theta_rad)


def is_consecutive(x1_1, y1_1, x1_2, y1_2, x2_1, y2_1, x2_2, y2_2):

    def point(x, y):
        if min(x1_1, x1_2) <= x <= max(x1_1, x1_2):
            # format(x, y)
            return True
        else:
            return False

    A1 = y1_1 - y1_2
    B1 = x1_2 - x1_1
    C1 = x1_1*y1_2 - x1_2*y1_1
    A2 = y2_1 - y2_2
    B2 = x2_2 - x2_1
    C2 = x2_1*y2_2 - x2_2*y2_1

    if B1*A2 - B2*A1 and A1:
        y = (C2*A1 - C1*A2) / (B1*A2 - B2*A1)
        x = (-C1 - B1*y) / A1
        return point(x, y)

    elif B1*A2 - B2*A1 and A2:
        y = (C2*A1 - C1*A2) / (B1*A2 - B2*A1)
        x = (-C2 - B2*y) / A2
        return point(x, y)

    else:
        return False


if __name__ == '__main__':
    rospy.init_node('arbitrator')

    tfBuffer = tf2_ros.Buffer()
    listener = tf2_ros.TransformListener(tfBuffer)

    rate = rospy.Rate(10.0)
    robot_name = rospy.get_param(rospy.get_name() + "/robot_name")

    tooCloseNotifier = rospy.Publisher(robot_name + '/tooClose', String, queue_size=1)
    robots = list(rospy.get_param('robots'))
    robots.remove(robot_name)
```

```
    while not rospy.is_shutdown():
        try:
            for another_robot in robots:
                if another_robot == 'ridgeback':
                    trans = tfBuffer.lookup_transform(robot_name + '/base_footprint', another_robot + '/base_link',
rospy.Time())
                elif robot_name == 'ridgeback':
                    trans = tfBuffer.lookup_transform(robot_name + '/base_link', another_robot + '/base_footprint',
rospy.Time())
                else:
                    trans = tfBuffer.lookup_transform(robot_name + '/base_footprint', another_robot + '/base_footprint',
rospy.Time())

                x = trans.transform.translation.x
                y = trans.transform.translation.y
                dist = math.sqrt((x ** 2) + (y ** 2))


                if (another_robot == 'ridgeback' or robot_name == 'ridgeback'):
                    if (another_robot == 'ridgeback'):
                        transAnotherInMap = tfBuffer.lookup_transform('map', another_robot + '/base_link', rospy.Time())
                        transSelfInMap = tfBuffer.lookup_transform('map', robot_name + '/base_footprint', rospy.Time())
                    else:
                        transAnotherInMap = tfBuffer.lookup_transform('map', another_robot + '/base_footprint',
rospy.Time())
                        transSelfInMap = tfBuffer.lookup_transform('map', robot_name + '/base_link', rospy.Time())
                else:
                    transAnotherInMap = tfBuffer.lookup_transform('map', another_robot + '/base_footprint', rospy.Time())
                    transSelfInMap = tfBuffer.lookup_transform('map', robot_name + '/base_footprint', rospy.Time())
                # rospy.loginfo(robot_name + "/arbitrator: distance to " + another_robot + " x: " + str(x))
                # rospy.loginfo(robot_name + "/arbitrator: distance to " + another_robot + " y: " + str(y))
                if (dist < 4):
                    # YIELD:
                    qxAIM = transAnotherInMap.transform.rotation.x
                    qyAIM = transAnotherInMap.transform.rotation.y
                    qzAIM = transAnotherInMap.transform.rotation.z
                    qwAIM = transAnotherInMap.transform.rotation.w

                    qxSIM = transSelfInMap.transform.rotation.x
                    qySIM = transSelfInMap.transform.rotation.y
                    qzSIM = transSelfInMap.transform.rotation.z
                    qwSIM = transSelfInMap.transform.rotation.w

                    quatAIM = [qxAIM, qyAIM, qzAIM, qwAIM]
                    (r, p, thetaAIM) = euler_from_quaternion(quatAIM)
                    quatSIM = [qxSIM, qySIM, qzSIM, qwSIM]
                    (r, p, thetaSIM) = euler_from_quaternion(quatSIM)

                    #if robots are facing each other:
                    yawDiff = 3.14 - (abs(thetaAIM) + abs(thetaSIM))
                    # rospy.loginfo(robot_name + '/arbitrator: yawDiff = ' + str(yawDiff))
                    if (yawDiff < 0.17):
                        # rospy.loginfo(robot_name + "/arbitrator: distance to " + another_robot + " : " + str(dist))
                        if rospy.has_param('prioritiesByRobots'):
                            prioritiesByRobotsParam = rospy.get_param('prioritiesByRobots')
                            # rospy.loginfo(robot_name + "/arbitrator: " +
str(prioritiesByRobotsParam[robot_name]['priority']) + " < " +
str(prioritiesByRobotsParam[another_robot]['priority']))
                            if (((prioritiesByRobotsParam[robot_name] != False) and
(prioritiesByRobotsParam[another_robot] != False))
                                and
                                (prioritiesByRobotsParam[robot_name]['priority'] <
prioritiesByRobotsParam[another_robot]['priority'])):
```

```
                        # rospy.loginfo(robot_name + "/arbitrator: sending yield message")
                        tooCloseNotifier.publish('yield')
                        rospy.set_param(robot_name + "/yielding_to", another_robot)
                    else:
                        rospy.loginfo(robot_name + "/arbitrator: no \'prioritiesByRobots\' parameter found")
                    # else:
                    #     rospy.loginfo(robot_name + "/arbitrator: path clear (but close)")
                    #     tooCloseNotifier.publish('go')
                else:
                    pass
                # rospy.loginfo(robot_name + "/arbitrator: path clear")
                # tooCloseNotifier.publish('go')

                # PASS:
                selfX1 = transSelfInMap.transform.translation.x
                selfY1 = transSelfInMap.transform.translation.y

                qx0 = transSelfInMap.transform.rotation.x
                qy0 = transSelfInMap.transform.rotation.y
                qz0 = transSelfInMap.transform.rotation.z
                qw0 = transSelfInMap.transform.rotation.w
                quat = [qx0, qy0, qz0, qw0]
                (r, p, theta0) = euler_from_quaternion(quat)


                selfX2, selfY2 = point_pos(selfX1, selfY1, 2, theta0)

                anotherX1 = transAnotherInMap.transform.translation.x
                anotherY1 = transAnotherInMap.transform.translation.y

                qx1 = transAnotherInMap.transform.rotation.x
                qy1 = transAnotherInMap.transform.rotation.y
                qz1 = transAnotherInMap.transform.rotation.z
                qw1 = transAnotherInMap.transform.rotation.w
                quat1 = [qx1, qy1, qz1, qw1]
                (r, p, theta1) = euler_from_quaternion(quat1)

                anotherX2, anotherY2 = point_pos(anotherX1, anotherY1, 2, theta1)

                res = is_consecutive(selfX1, selfY1, selfX2, selfY2, anotherX1, anotherY1, anotherX2, anotherY2)

                qx = trans.transform.rotation.x
                qy = trans.transform.rotation.y
                qz = trans.transform.rotation.z
                qw = trans.transform.rotation.w

                qaaa = [qx, qy, qz, qw]
                (r, p, ttt) = euler_from_quaternion(qaaa)

                if (res and ((1.47 < abs(ttt)) and (abs(ttt) < 1.67))):
                    if rospy.has_param('prioritiesByRobots'):
                        prioritiesByRobotsParam = rospy.get_param('prioritiesByRobots')
                        # rospy.loginfo(robot_name + "/arbitrator: " +
str(prioritiesByRobotsParam[robot_name]['priority']) + " < " +
str(prioritiesByRobotsParam[another_robot]['priority']))
                        if (((prioritiesByRobotsParam[robot_name] != False) and
(prioritiesByRobotsParam[another_robot] != False))
                            and
                            (prioritiesByRobotsParam[robot_name]['priority'] <
prioritiesByRobotsParam[another_robot]['priority'])):
                            # rospy.loginfo(robot_name + "/arbitrator: sending yield message")
                            tooCloseNotifier.publish('pass')
                            rospy.set_param(robot_name + "/passing_to", another_robot)
```

```
                else:
                    rospy.loginfo(robot_name + "/arbitrator: no \'prioritiesByRobots\' parameter found")
            else:
                pass

        else:
            pass
            # rospy.loginfo(robot_name + "/arbitrator: path clear")
            # tooCloseNotifier.publish('go')

    except Exception as e:
        rospy.loginfo(robot_name + "/arbitrator: " + e.message)
        # logging.exception("arbitrator")
        pass

    rate.sleep()
```

ПРИЛОЖЕНИЕ В

Объектное представление типа данных “задача” и соответствующий .msg тип сообщения

Файл Task.py:

```
#!/usr/bin/env python

import rospy
from move_base_msgs.msg import MoveBaseGoal
import itertools

class Task:

    id = itertools.count().next

    def __init__(self, goalId, priority, robotName, taskType, id = None, waitTime = None):
        if id is None:
            self.id = Task.id()
        else:
            self.id = id
        self.priority = priority
        self.isDone = False
        self.isCurrent = False
        self.isPaused = False
        self.goalId = goalId
        self.taskType = taskType
        self.robotName = robotName
        self.waitTime = waitTime


    def setDoneStatus(self):
        self.isDone = True
        self.isCurrent = False
        self.isPaused = False


    def setCurrentStatus(self):
        if self.isDone:
            raise NameError('Cannot set status "Current" to the task with status "Done"')
        else:
            self.isCurrent = True
            self.isPaused = False
```

```
    def setPausedStatus(self):
        if self.isDone:
            raise NameError('Cannot set status "Paused" to the task with status "Done"')
        else:
            self.isCurrent = False
            self.isPaused = True
```

Файл TaskMsg.msg:

```
int64 taskId
string goalId
string robotName
int64 priority
string taskType
string waitTime
```

ПРИЛОЖЕНИЕ Г

Менеджер задач

Файл mymodule.py:

```
#!/usr/bin/env python

import os
import rospy
import rospkg
import sys
import collections

from qt_gui.plugin import Plugin
from python_qt_binding import loadUi
from python_qt_binding.QtWidgets import QWidget
from PyQt5.QtWidgets import QTabWidget
from std_msgs.msg import String
import addTaskDesign
from navgoals_smach.msg import TaskMsg
from Task import Task

waypoint = collections.OrderedDict()
taskList = []
robotTabMap = {}
robotCurrentTaskMap = {}

def fillStationMap():
    with open("/home/ruslan/hospital_ws/src/goal_scenarios/stations.txt", "r") as f:
        for line in f:
            data = line.split(" ")
            key, value = data[0], data[1::]
            waypoint[key] = value


class MyPlugin(Plugin):

    def __init__(self, context):
        super(MyPlugin, self).__init__(context)
        self.setObjectName('MyPlugin')
        self._widget = QTabWidget()
```

```
        ui_file = os.path.join(rospkg.RosPack().get_path('rqt_tasks'), 'resource', 'addTaskDesign.ui')
        ui_robotTab_file = os.path.join(rospkg.RosPack().get_path('rqt_tasks'), 'resource', 'addTaskDesignTab.ui')
        loadUi(ui_file, self._widget)
        self._widget.setObjectName('MyPluginUi')
        self._widget.setTabText(0, 'Main Panel')
        context.add_widget(self._widget)

        self.sendTaskPub = rospy.Publisher('/goals', TaskMsg, queue_size=1)
        # self.sendHighestPriorityRobotPub = rospy.Publisher('/highestPriorityRobot', String, queue_size=1)
        # Obtain completed/started task id
        self.getTaskStatusSub = rospy.Subscriber("/taskStatus", String, self.updateStatusCb)
        # self.getHighestPriorityRobotRequestSub = rospy.Subscriber("/getHighestPriorityRobot", String,
self.handleHighestPriorityRequest)

        fillStationMap()
        self.fillWaypointList()
        self.fillPriorityList()
        self.fillTaskTypeList()
        # self.fillRobotList()

        # self.robot1Tab = QWidget()
        # self.pmb2Tab = QWidget()
        # self.huskyTab = QWidget()
        # self.pmb3Tab = QWidget()

        rlist = rospy.get_param("robots")
        for rName in rlist:
            robotTabMap[rName] = QWidget()
            loadUi(ui_robotTab_file, robotTabMap[rName])
            self._widget.addTab(robotTabMap[rName], rName)
            self._widget.robotList.addItem(rName)

        # robotTabMap['robot1'] = self.robot1Tab
        # robotTabMap['pmb2'] = self.pmb2Tab
        # robotTabMap['pmb3'] = self.pmb3Tab
        # robotTabMap['husky'] = self.huskyTab
        # for robotName, tab in robotTabMap.items():
        #     loadUi(ui_robotTab_file, tab)
        #     self._widget.addTab(tab, robotName)

        self._widget.sendTaskBtn.clicked.connect(self.sendTask)


    def sendTask(self):
        self.robot_name = self._widget.robotList.currentText().encode('utf-8')
        self.priority = self._widget.priorityList.currentText()
        self.taskType = self._widget.taskList.currentText()
        self.pointId = self._widget.pointIdList.currentText().encode('utf-8') if (self.taskType != 'Wait') else None
        self.waitTime = self._widget.waitTimeInput.text().encode('utf-8')
        if (not self.waitTime) and (self.taskType == 'Wait'): raise NameError('Need wait time!')
        self.task = Task(self.pointId, self.priority, self.robot_name, self.taskType, None, self.waitTime)
        self.taskMsg = TaskMsg(self.task.id, self.pointId, self.robot_name, int(self.priority), self.taskType,
self.waitTime)

        taskList.append(self.task)
        taskList.sort(key=lambda task: task.priority, reverse=True)
        # self._widget.queueTaskList.addItem(self.robot_name + ", " + self.taskType + ", station id: " + self.pointId +
", priority: " + self.priority)
        self.fillTaskListWidgets()
        self.sendTaskPub.publish(self.taskMsg)


    def updateStatusCb(self, data):
```

```
        values = data.data.split(" ")
        task = next((i for i in taskList if str(i.id) == values[0]), None)
        if (task is None): raise NameError("Cannot find task with id " + str(values[0]))
        if (values[1] == 'done'):
            task.setDoneStatus()
            robotCurrentTaskMap[task.robotName] = False
        elif (values[1] == 'current'):
            task.setCurrentStatus()
            robotCurrentTaskMap[task.robotName] = task
        elif (values[1] == 'paused'): task.setPausedStatus()
        else: raise NameError("In message: " + data.data + " unrecognized task status")
        self.fillTaskListWidgets()
        rospy.set_param('prioritiesByRobots', robotCurrentTaskMap)


    # def handleHighestPriorityRequest(self, data):
    #     values = data.data.split(" ")
    #     rospy.loginfo("request recieved: " + str(values))
    #     winner = values[0] if (robotCurrentTaskMap[values[0]].priority > robotCurrentTaskMap[values[1]].priority)
else values[1]
    #     self.sendHighestPriorityRobotPub.publish(winner)


    def fillTaskListWidgets(self):
        self.clearAllLists()
        for task in taskList:
            taskText = self.buildTaskListWidgetText(task)
            if task.isDone:
                self._widget.doneTaskList.addItem(taskText)
                robotTabMap[task.robotName].doneTaskList.addItem(taskText)
            elif task.isCurrent:
                self._widget.currentTaskList.addItem(taskText)
                robotTabMap[task.robotName].currentTaskList.addItem(taskText)
            else:
                self._widget.queueTaskList.addItem(taskText)
                robotTabMap[task.robotName].queueTaskList.addItem(taskText)


    def clearAllLists(self):
        self._widget.doneTaskList.clear()
        self._widget.currentTaskList.clear()
        self._widget.queueTaskList.clear()
        for tab in robotTabMap.values():
            tab.doneTaskList.clear()
            tab.currentTaskList.clear()
            tab.queueTaskList.clear()


    # def getCurrentTaskPriorityByRobotName(self, robotName):
    #     self.getCurrentTaskPriorityByRobotNamePub.publish(robotName + " " + )


    def fillWaypointList(self):
        for point in waypoint:
            self._widget.pointIdList.addItem(point[0])

    # def fillWaypointList(self):
    #     for point in waypoint:
    #         self._widget.pointIdList.addItem(point)


    def fillPriorityList(self):
        for i in range (0, 10):
```

```
            self._widget.priorityList.addItem(str(i))


    def fillTaskTypeList(self):
        self._widget.taskList.addItem('Move')
        self._widget.taskList.addItem('Wait')


    # def fillRobotList(self):
    #     self._widget.robotList.addItem('robot1')
    #     self._widget.robotList.addItem('husky')
    #     self._widget.robotList.addItem('pmb2')
    #     self._widget.robotList.addItem('pmb3')


    def buildTaskListWidgetText(self, task):
        return task.robotName + ", " + task.taskType + ", station id: " + self.emptyForNone(task.goalId) + ", priority: "
+ task.priority


    def emptyForNone(self, s):
        if s is None:
            return ''
        return str(s)
```

Файл addTaskDesign.py:

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(558, 636)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.line = QtWidgets.QFrame(self.centralwidget)
        self.line.setGeometry(QtCore.QRect(0, 40, 551, 16))
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.line_2 = QtWidgets.QFrame(self.centralwidget)
        self.line_2.setGeometry(QtCore.QRect(270, 10, 21, 631))
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Minimum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.line_2.sizePolicy().hasHeightForWidth())
        self.line_2.setSizePolicy(sizePolicy)
        self.line_2.setFrameShape(QtWidgets.QFrame.VLine)
        self.line_2.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line_2.setObjectName("line_2")
        self.robotList = QtWidgets.QComboBox(self.centralwidget)
        self.robotList.setGeometry(QtCore.QRect(10, 90, 261, 25))
        self.robotList.setObjectName("robotList")
        self.taskList = QtWidgets.QComboBox(self.centralwidget)
        self.taskList.setGeometry(QtCore.QRect(10, 180, 121, 25))
        self.taskList.setObjectName("taskList")
        self.pointIdList = QtWidgets.QComboBox(self.centralwidget)
        self.pointIdList.setGeometry(QtCore.QRect(150, 180, 121, 25))
        self.pointIdList.setObjectName("pointIdList")
        self.priorityList = QtWidgets.QComboBox(self.centralwidget)
        self.priorityList.setGeometry(QtCore.QRect(150, 220, 121, 25))
        self.priorityList.setObjectName("priorityList")
```

```
        self.sendTaskBtn = QtWidgets.QPushButton(self.centralwidget)
        self.sendTaskBtn.setGeometry(QtCore.QRect(10, 270, 261, 31))
        self.sendTaskBtn.setObjectName("sendTaskBtn")
        self.listWidget = QtWidgets.QListWidget(self.centralwidget)
        self.listWidget.setGeometry(QtCore.QRect(290, 60, 251, 571))
        self.listWidget.setObjectName("listWidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(10, 10, 251, 31))
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(290, 10, 251, 31))
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(20, 150, 121, 20))
        self.label_3.setObjectName("label_3")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(150, 150, 121, 20))
        self.label_4.setObjectName("label_4")
        self.label_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_5.setGeometry(QtCore.QRect(40, 220, 101, 20))
        self.label_5.setObjectName("label_5")
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.sendTaskBtn.setText(_translate("MainWindow", "ADD TASK"))
        self.label.setText(_translate("MainWindow", "                    ADD TASK"))
        self.label_2.setText(_translate("MainWindow", "                CURRENT TASKS"))
        self.label_3.setText(_translate("MainWindow", "      TASK TYPE"))
        self.label_4.setText(_translate("MainWindow", "        POINT ID"))
        self.label_5.setText(_translate("MainWindow", "       PRIORITY"))
```