

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS**

**Разработка алгоритма принятия решения в задаче навигации мобильного робота в
условиях динамического окружения**

Обучающийся / Student Кирбаба Денис Дмитриевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет систем управления и
робототехники
Группа/Group R3438
Направление подготовки/ Subject area 15.03.06 Мехатроника и робототехника
Образовательная программа / Educational program Робототехника 2020
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Бакалавр
Руководитель ВКР/ Thesis supervisor Бжихатлов Ислам Асланович, кандидат технических
наук, Университет ИТМО, факультет систем управления и робототехники, доцент
(квалификационная категория "ординарный доцент")

Обучающийся/Student

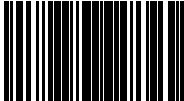
Документ подписан	
Кирбаба Денис Дмитриевич	
13.05.2024	

(эл. подпись/ signature)

Кирбаба Денис
Дмитриевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Бжихатлов Ислам Асланович	
12.05.2024	

(эл. подпись/ signature)

Бжихатлов
Ислам
Асланович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Кирбаба Денис Дмитриевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет систем управления и робототехники
Группа/Group R3438
Направление подготовки/ Subject area 15.03.06 Мехатроника и робототехника
Образовательная программа / Educational program Робототехника 2020
Язык реализации ОП / Language of the educational program Русский
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка алгоритма принятия решения в задаче навигации мобильного робота в условиях динамического окружения
Руководитель ВКР/ Thesis supervisor Бжихатлов Ислам Асланович, кандидат технических наук, Университет ИТМО, факультет систем управления и робототехники, доцент (квалификационная категория "ординарный доцент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Разработка системы принятия решения в задаче навигации мобильного робота в условиях динамического окружения

Задачи, решаемые в ВКР / Research tasks

1. Аналитический обзор задачи навигации и методов её решения 2. Обзор существующих алгоритмов принятия решения 3. Разработка алгоритма принятия решения для навигации в динамическом окружении 4. Аппробация разработанного алгоритма в среде имитационного моделирования на примере дифференциального робота 5. Сравнительный анализ полученных результатов

Краткая характеристика полученных результатов / Short summary of results/findings

Результатом работы является система принятия решения для навигации в динамическом окружении. Разработанная система состоит из двух компонент: система контроля и система учета динамических объектов. Система контроля основана на архитектуре поведенческого дерева, с помощью которого обеспечивается непрерывная проверка состояния робота и системы навигации. При обнаружении неисправностей алгоритм вызывает действия для восстановления работоспособности робота. Данная система обеспечивает прирост безопасности и автономности. Система учета динамических объектов адаптирует навигационные процессы на основе информации о динамических объектах. Это позволяет уменьшить число столкновений и время достижения конечной цели.

Обучающийся/Student

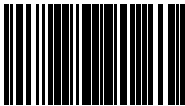
Документ подписан	
Кирбаба Денис Дмитриевич	
13.05.2024	

(эл. подпись/ signature)

Кирбаба Денис
Дмитриевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Бжихатлов Ислам Асланович	
12.05.2024	

(эл. подпись/ signature)

Бжихатлов
Ислам
Асланович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Кирбаба Денис Дмитриевич

Факультет / Faculty факультет систем управления и робототехники

Группа / Group R3438

Направление подготовки / Subject area 15.03.06 - Мехатроника и робототехника

Образовательная программа / Educational program Робототехника

Язык реализации ОП / Language of the educational program Русский

Квалификация / Degree level Бакалавр

Тема ВКР / Thesis topic Разработка алгоритма принятия решения в задаче навигации мобильного робота в условиях динамического окружения

Руководитель ВКР / Thesis supervisor Бжихатлов Ислам Асланович, кандидат технических наук, Университет ИТМО, факультет систем управления и робототехники, доцент (квалификационная категория "ординарный доцент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Задача навигации мобильным роботом в условиях динамического окружения является актуальной проблемой. Применение высокоуровневых структур управления роботом может быть одним из способов её решения.

Целью выпускной квалификационной работы является разработка алгоритма принятия решения наземного мобильного робота для задачи навигации в условиях динамического окружения.

Задачи работы:

1. Изучение информации о существующих алгоритмах принятия решений
2. Разработка алгоритма принятия решения, который применим к задаче навигации наземным роботом в динамическом окружении. Алгоритм должен удовлетворять следующим требованиям:
 - а) Автономная навигация в динамическом окружении. Выполнение задач восприятия, локализации, построения карты местности, планирования пути и управление движением
 - б) Безопасность людей и другого оборудования в окружении. Обнаружение статических и динамических объектов и соответствующая адаптация поведения
 - в) Эффективное функционирование вблизи статических и динамических объектов
 - г) Оценка состояния робота и системы навигации и соответствующая адаптация поведения

3. Аппробация разработанного алгоритма в среде имитационного моделирования на примере дифференциального робота
4. Сравнительный анализ полученных результатов

Требования к мобильному роботу:

1. Наличие одноплатного компьютера, способного взаимодействовать с датчиками мобильного робота, запускать ROS2 и обеспечивать связь с сервером обработки данных
2. Наличие лазерного датчика расстояния со следующими минимальными параметрами:
 - а) Линейный диапазон измерений: 0.2 - 8.0 м
 - б) Точность линейных измерений: 0.01 м
 - в) Угловой диапазон измерений: 0 - 360°
 - г) Разрешение углового диапазона: 1°
 - д) Частота сканирования: 5 Гц
3. Наличие инерциального измерительного блока:
 - а) 3-х осевой гироскоп
 - б) 3-х осевой акселерометр
4. Наличие датчика столкновения
5. Дифференциальный тип привода

Требования к серверу обработки данных:

1. Частота процессора не менее 2.5 ГГц, количество ядер не менее 4
2. ОЗУ не менее 8 ГБ
3. Твердотельный накопитель объемом не менее 128 Гб
4. Сетевые интерфейсы для обеспечения связи с мобильным роботом

Дата выдачи задания / Assignment issued on 31.01.2024

Срок представления готовой ВКР / Deadline for final edition of the thesis 30.05.2024

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ	8
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	9
ВВЕДЕНИЕ	10
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	11
1.1 Автономная навигация	11
1.1.1 Восприятие окружения	13
1.1.2 Локализация и построение карты местности	14
1.1.3 Планирование пути	15
1.1.4 Управление движением	16
1.2 Методы принятия решений	17
1.2.1 Конечный автомат	18
1.2.2 Поведенческое дерево	19
1.2.3 Частично наблюдаемые марковские процессы принятия решений	20
1.3 Навигация в динамическом окружении	22
1.3.1 Обнаружение динамических объектов	22
1.3.2 Слежение за динамическими объектами	23
1.3.3 Отображение динамических объектов на карте стоимости	26
2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ	28
2.1 ROS2	28
2.2 Навигационная система Nav2	29
2.3 Модель робота Turtlebot3	31
2.4 Симуляционная среда Gazebo	32
2.5 Behaviortree.CPP	32
2.6 Groot	33
2.7 Используемое ПО	34
3 РЕАЛИЗАЦИЯ СИСТЕМЫ	36

3.1	Предпосылки разрабатываемой системы	36
3.2	Стандартная архитектура для навигации	37
3.3	Компоненты разработанной системы принятия решений	38
3.4	Структура программной реализации системы	39
3.5	Система контроля.....	40
3.5.1	Архитектура системы	40
3.5.2	Модуль контроля датчиков	41
3.5.3	Модуль навигационных сервисов.....	41
3.5.4	Модуль хранения данных	42
3.5.5	Модуль переключения управления	43
3.5.6	Основное поведенческое дерево	43
3.5.7	Дополнительные модули.....	49
3.6	Система учета динамических объектов	51
3.6.1	Принципы обхода препятствий в Nav2.....	51
3.6.2	Описание работы предлагаемой системы.....	53
3.6.3	Архитектура системы	54
3.6.4	Модуль обнаружения	54
3.6.5	Модуль слежения и оценки	56
3.6.6	Модуль добавления слоя	58
3.6.7	Модуль пользовательских интерфейсов системы	60
3.7	Общая архитектура системы принятия решений	61
4	МОДЕЛИРОВАНИЕ И ОЦЕНИВАНИЕ	62
4.1	Модификация модели мобильного робота.....	62
4.2	Симуляционные миры	63
4.3	Картографирование местности.....	65
4.4	Тестирование.....	66
4.4.1	Тестирование системы контроля.....	67
4.4.2	Тестирование системы учета динамических объектов ..	71
5	ЗАКЛЮЧЕНИЕ	74
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	75

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

SLAM - Simultaneous Localization and Mapping

ANN - Artificial Neural Network

ПИД-регулятор - Пропорционально-интегрально-дифференцирующий регулятор

ROS2 - Robot Operating System 2

DWB - Dynamic Window Approach для ROS2

TEB - Timed Elastic Band

LIDAR - Light Detection and Ranging

IMU - Inertial Measurement Unit

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

SLAM - метод одновременного построения карты местности и определения на ней положения мобильного робота.

ANN - искусственная нейронная сеть - это модель машинного обучения, состоящая из взаимосвязанных узлов, которые обрабатывают входные данные с помощью взвешенных связей, позволяя решать сложные задачи распознавания образов, классификации и прогнозирования.

ПИД-регулятор - это компонент системы управления, регулирующий выходную переменную на основе ошибки между текущим и желаемым сигналами. Сигнал управления при этом состоит из линейной комбинации трех составляющих - пропорциональной, интегральной и дифференцирующей.

ROS2 - это набор программных библиотек и инструментов, которые помогают создавать приложения для роботов.

DWB - это плагин для локального планировщика, реализующий подход динамического окна (Dynamic Window Approach).

TEB - это плагин для локального планировщика, реализующий подход эластичных лент (Elastic Band Approach).

LIDAR - это устройство, которое использует лазерные импульсы для измерения расстояния до объектов в окружающей среде.

IMU - это устройство, используемое для измерения и отслеживания ориентации, скорости и ускорения робота в трехмерном пространстве.

ВВЕДЕНИЕ

В последнее время роботы стали важной частью нашей жизни. Они используются во многих областях и применяются для различных целей, в которых позволяют повысить эффективность, быстродействие и безопасность.

Самым простым способом внедрения роботов для выполнения задач является использование стационарных роботов. Основываясь на требованиях высокой надежности, они статичны, неинтерактивны и работают в неизменяющихся условиях вдали от людей. Работа данных роботов довольно предсказуема, так как они управляются заранее синтезированными программами для выполнения спланированных задач.

Очевидно, что область применения стационарных роботов ограничена и для многих задач они не смогут быть применены. Соответственно появляется необходимость в мобильных роботах, которые могут осуществлять движение в среде и функционировать с объектами. В связи с этим при работе мобильных роботов задача навигации имеет важное значение. Роботу необходимо обрабатывать информацию об окружении и строить верные безопасные маршруты в реальном времени для достижения заданных целей.

Так как мобильный робот функционирует в динамической среде в непосредственной близости от людей, оборудования и других объектов, робот должен достаточно быстро реагировать на изменения и не допускать возникновения критических ситуаций. Для реализации такой высокоуровневой стратегии управления используются алгоритмы принятия решений.

Основная задача данной работы - разработка алгоритма принятия решения для задачи навигации мобильным роботом при функционировании в динамическом окружении. Разработанный алгоритм должен обеспечивать безопасное и эффективное поведение робота.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Автономная навигация

Задача навигации является основополагающей при функционировании мобильного робота.

Автономная навигация - это способность робота воспринимать информацию о внешнем мире через датчики, строить по этим данным модель окружения, оценивать свою конфигурацию в этом окружении, строить достижимую траекторию движения до заданной цели навигации и подавать такое управление на моторы, которое способствовало бы движению по построенной траектории.

Автономность навигации заключается в том, что планировщик траектории, являющийся частью системы навигации робота, может без вмешательства человека-оператора формировать путь до цели, учитывая объезд препятствий.

Исходя из данного определения задачу автономной навигации можно разделить на 4 отдельных подзадачи:

- восприятие окружения;
- локализация робота и построение карты местности;
- планирование пути;
- управление движением.

На Рисунке 1 представлена диаграмма задачи навигации, на ней можно увидеть как компоненты системы связаны между собой.

Любая система для автономной навигации мобильного робота реализует в какой-то форме данные компоненты. Архитектуры таких систем принято выполнять в иерархичной структуре, соответствующей парадигме "Ощущать, думать, действовать"[1].

Структура данной архитектуры представлена на Рисунке 2.

На уровне восприятия происходит анализ данных с сенсоров и соответствующая локализация и построение карты местности, а на уровне управления - непосредственная подача сигналов на двигатели робота.

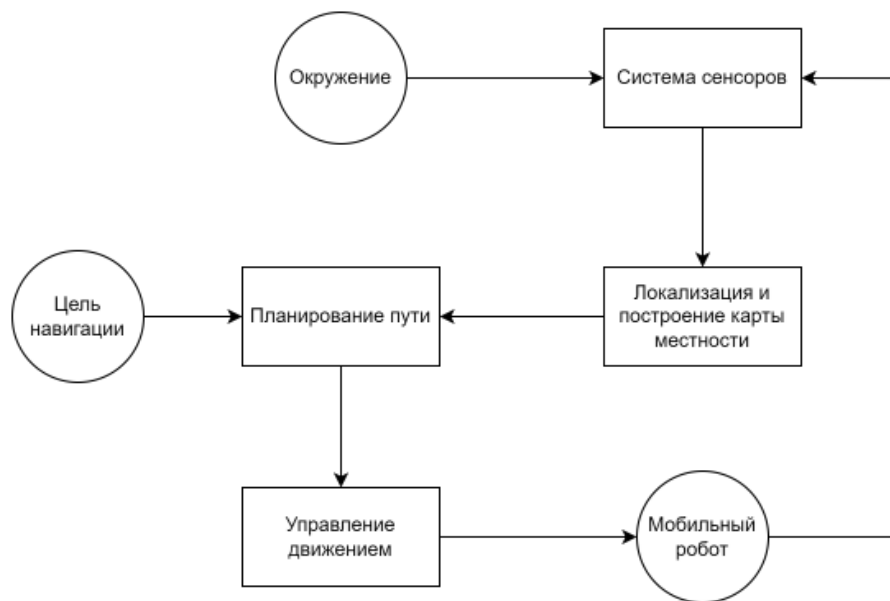


Рисунок 1 — Диаграмма задачи навигации

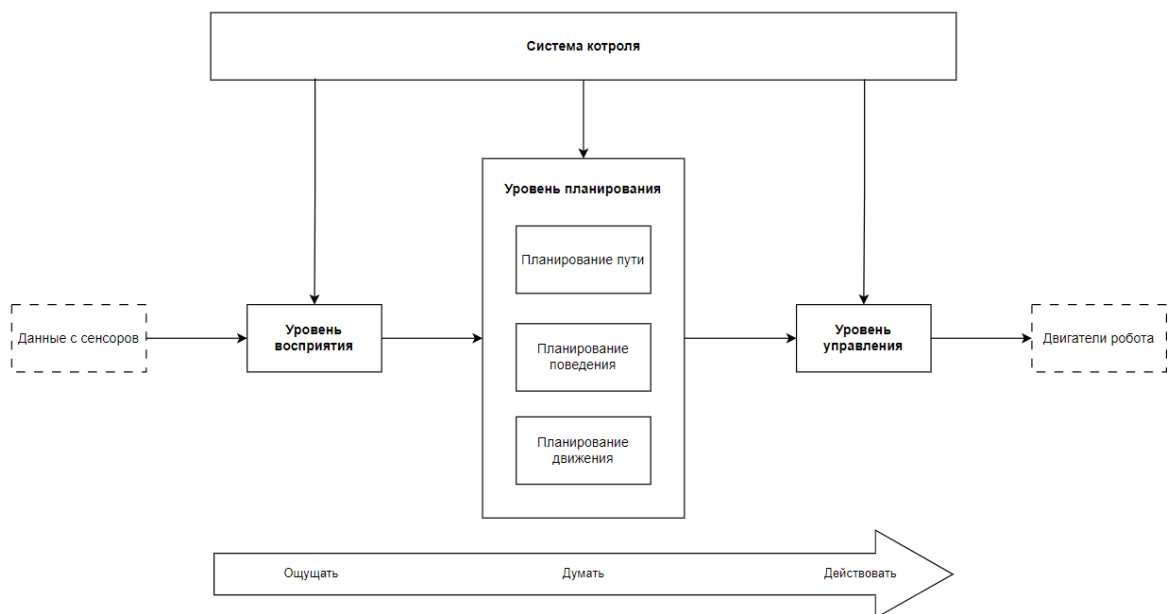


Рисунок 2 — Диаграмма стандартной архитектуры системы автономной навигации робота

Рассмотрим подробнее уровень планирования. Как видно, он разделен на 3 компонента:

- планирование пути (глобальный планировщик) - реализует построение траектории движения робота от его начальной точки до конечной;
- планирование поведения - выполняет задачу по корректировке траектории, построенной глобальным планировщиком, в соответствии

- с локальным состоянием окружения (объезд появившихся препятствий, в том числе динамических, следование правилам движения и заданным ограничениям);
- планирование движения (локальный планировщик) - основной задачей является исполнение задачи слежения за траекторией, полученной от поведенческого слоя.

Также важным компонентом такой архитектуры является система контроля. Она обрабатывает информацию о состоянии всех компонентов и производит проверки для определения неполадок во время функционирования системы. В случае если была обнаружена неисправность или нежелательное поведение системы, модуль контроля обеспечивает приостановку нормального функционирования системы и делает попытки по восстановлению её работоспособности. Это позволяет повысить безопасность и автономность системы.

В последующих секциях данной главы мы разберем подробнее подзадачи навигации и представим современные способы их решения.

1.1.1 Восприятие окружения

Системы восприятия окружения позволяют роботу получать информацию об окружающей его среде. Получаемая информация может иметь разнообразный вид, исходя из этого все системы можно разделить на следующие группы [2]:

1. Системы, определяющие геометрические и другие параметры окружения. Такие системы позволяют получить данные о пространственном расположении, расстоянии, форм и размеров объектов в окружении. Выходные данные системы представлены в простой форме (например, расстояние до объекта в метрах), а значит анализ информации не требует больших вычислительных ресурсов, а используемые алгоритмы имеют относительно небольшую пространственную и вычислительную сложность. Примерами таких систем являются лазерные дальномеры (LIDAR), ультразвуковые дальномеры (SONAR), акселерометры, одометры, камеры глубины и другие

2. Системы компьютерного зрения. Единицей информации в таких системах - изображение с камеры, поступающие с определенной частотой. Этап обработки таких данных обычно подразумевает выделение характеристик или свойств, которые будут использованы для выполнения задачи навигации. Стоит отметить, что алгоритмы технического зрения требуют больших вычислительных мощностей
3. Системы силомоментного осязания роботов. Данные системы основаны на измерении сил и моментов сил. Получение обратной связи от окружения в форме силовых покателей может быть важным аспектом системы навигации. Примером такой системы для мобильного робота является датчик столкновения
4. Системы, использующие несколько различных типов сенсоров. Применение разных видов датчиков позволяет получать данные разной природы и обеспечивает избыточность данных. Однако в таком случае появляется необходимость решать задачу объединения данных

1.1.2 Локализация и построение карты местности

Задача SLAM является важной частью в автономной навигации робота. Алгоритмы SLAM принимают на вход данные с различных сенсоров робота (в основном камеры, лазерные дальномеры и инерциальные измерительные приборы), производят их обработку и объединение. Результатом их работы является карта окружения с отмеченным положением робота на ней.

Алгоритмы SLAM можно разделить на 2 группы [3]:

1. Визуальный SLAM - использует изображения с камер. Применяемые алгоритмы: PTAM, ORB-SLAM, DTAM, DSO
2. Лазерный SLAM - использует данные с лазерных дальномеров. Применяемые алгоритмы: ICP, NDT, LOAM, FGR

Однако на практике локализация и построение карты местности производится с использованием нескольких сенсоров. Это позволяет существенно увеличить точность и надежность SLAM, так как преимущества отдельных датчиков складываются, а индивидуальные недостатки сглаживаются.

1.1.3 Планирование пути

Планирование пути – поиск последовательности конфигураций робота для передвижения его из точки А в точку Б.

На задачу планирования пути оказывают влияние следующие факторы:

- габариты, кинематические и динамические ограничения робота;
- тип окружения (статическое или динамическое);
- наличие карты местности (существует заранее созданная карта местности или же используется SLAM);
- степень неопределенности при работе сенсоров (зашумленная или неполная информация) и при передвижении робота;
- требования к используемому алгоритму:
 - вид функционала качества;
 - пространственная и временная сложности;
 - точность решения.

Математическое описание задачи

Конфигурация робота - минимальное множество параметров, которые определяют степени свободы робота относительно фиксированной системы координат. В данной работе мы рассматриваем двумерную систему координат, тогда конфигурация будет иметь вид:

$$q = (x, y, \theta). \quad (1)$$

Конфигурационное пространство C - пространство всех возможных конфигураций робота:

$$C : \mathbb{R}^2 \times SO(2). \quad (2)$$

Конфигурационное пространство разбивают на 2 непересекающихся множества:

$$C = C_{\text{преп}} \cup C_{\text{своб}}, \quad C_{\text{преп}} \cap C_{\text{своб}} = \emptyset. \quad (3)$$

На Рисунке 3 представлен пример разбиения конфигурационного пространства.

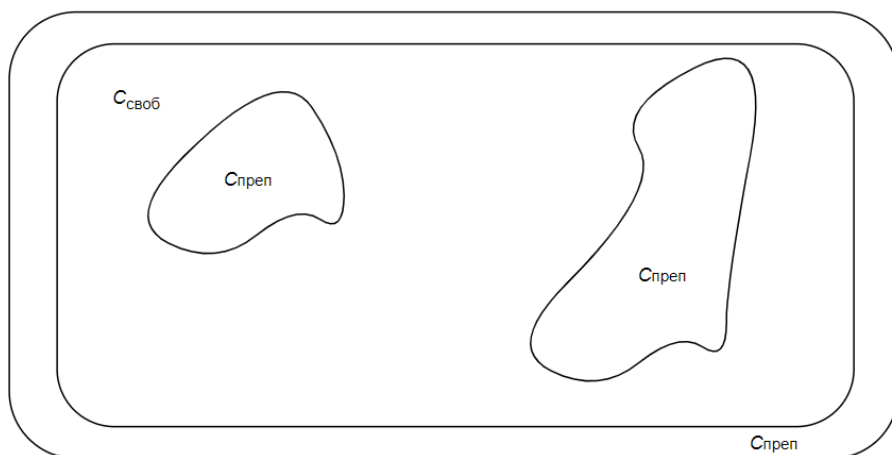


Рисунок 3 — Конфигурационное пространство

Тогда задачу планирования можно представить в форме поиска непрерывного пути:

$$\tau : [0,1] \rightarrow C_{\text{своб}}, \quad \tau(0) = q_{\text{старт}}, \quad \tau(1) = q_{\text{цель}}. \quad (4)$$

Классификация алгоритмов планирования

Приведем основные виды алгоритмов планирования пути [4]:

1. На основе графов: диаграмма видимости, диаграмма Вороного, вероятностная дорожная карта, метод быстро исследующих случайных деревьев (RRT)
2. На основе клеточной декомпозиции: алгоритмы Дейкстры, A^* , D^*
3. На основе потенциальных полей
4. Оптимизационные методы
5. На основе интеллектуальных технологий: муравьиный алгоритм, ANN, метод роя частиц, реактивные методы

1.1.4 Управление движением

Управление движением решает задачу перевода высокоуровневых задач навигации в конкретные сигналы управления, подаваемые непосредственно на двигатели робота.

В данной задаче можно выделить 3 основных аспекта:

1. Задача слежения

2. Наличие обратных связей управления
3. Учет ограничений робота

Способность реализовывать намеченный план передвижения является основополагающей в задаче навигации. Для решения задачи слежения применяются методы теории автоматического управления.

Дадим математическое описание задачи слежения [5].

Пусть $y(t)$ - выходной сигнал замкнутой системы, $g(t)$ - эталонный сигнал, $u(t)$ - сигнал управления. Тогда целью управления задачи слежения будет:

$$\lim_{t \rightarrow \infty} (y(t) - g(t)) = 0. \quad (5)$$

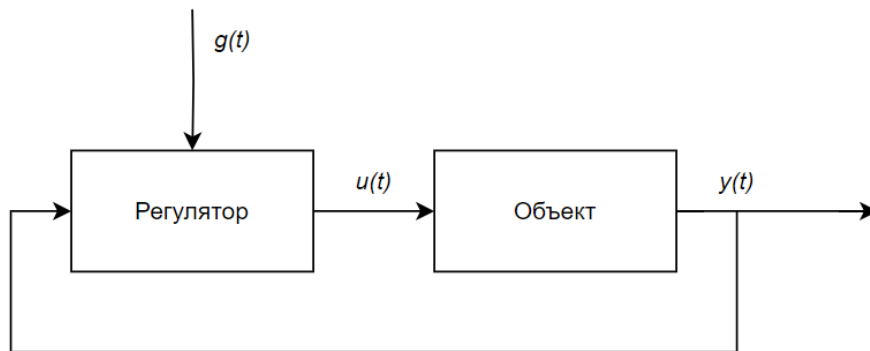


Рисунок 4 — Блок-схема системы управления выполняющей слежение за эталонным сигналом

Итак, при решении задачи слежения используются методы управления с обратной связью, которые на основе данных с сенсоров корректируют траекторию движения мобильного робота. Использование ПИД-регулятора является одним из наиболее простых решений для данной задачи.

1.2 Методы принятия решений

Методы принятия решений представляют собой системы, которые на высоком уровне решают, какая стратегия поведения робота будет выполняться на основе данных о состоянии робота и его окружения. Все описанные ниже системы принятия решений основаны на использовании состояний, в который может находиться робот.

1.2.1 Конечный автомат

Конечные автоматы описывают поведение в форме состояний, которые вызывают выполнение соответствующих действий. Используя конечный автомат можно разделить задачу управления на отдельные блоки, получив наглядное представление того, в каких случаях система будет оказываться в определенном состоянии.

Такая структуризация упрощает дальнейшую отладку (например, если робот застрял в определенном состоянии, легко определить, какой переход или какое условие не работает).

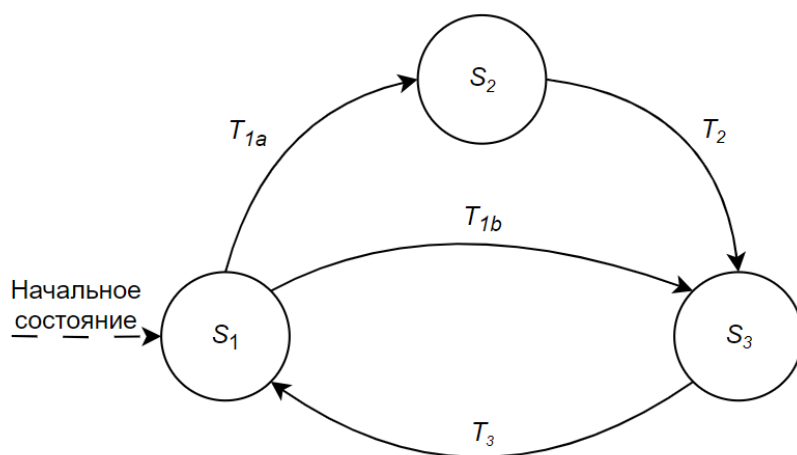


Рисунок 5 — Графическое представление конечного автомата

На Рисунке 5 представлено графическое описание конечного автомата, состоящее из дискретных состояний, представленных узлами графа, и переходов, представленных ребрами. Когда срабатывает переход, конечный автомат изменяет свое текущее состояние и совершает переход в другое состояние в соответствии с функцией перехода.

В математической форме конечный автомат имеет вид кортежа [6]:

$$A = (S, \Sigma, \delta, s_0, F). \quad (6)$$

где S - конечное непустое множество состояний, Σ - входной алфавит (конечное непустое множество символов), $\delta : S \times \Sigma \rightarrow S$ - функция перехода, s_0 - начальное состояние, F - конечное (терминальное) состояние.

Главный недостаток конечного автомата - возникающие сложности при реализации сложных поведений. По мере добавления новых состояний в конечный автомат существующие состояния должны тщательно пересматриваться и корректироваться для обеспечения согласованности и точности.

1.2.2 Поведенческое дерево

Поведенческие деревья - это еще один способ управления поведением автономных систем. Они впервые получили широкое распространение в индустрии компьютерных игр, где они в основном используются в целях моделирования искусственного интеллекта для неигровых персонажей [7]. Каждое дерево имеет один корень и множество дочерних, родительских и листовых узлов. Листовые узлы определяют определенные действия или условия, а нелистовые узлы являются узлами управления и контролируют путь прохода по дереву [8].

Выполнение происходит путем подачи сигнала на корневой узел дерева. Затем этот сигнал передается в дочерние узлы, которые либо определяют куда дальше пойдет сигнал (узлы управления), либо выполняют действия или проверяют условия. Листовые узлы возвращают сигнал трех видов: успех, неудача, выполнение. В зависимости от типа возвращаемого сигнала также меняется проход дерева поведения.

Рассмотрим 4 типа узлов управления: последовательный узел, селектор, узел условия и декоратор.

Узел последовательности реализует функционал логического И и обозначается символом стрелки.

Селектор - это эквивалент логического условия ИЛИ, обозначается вопросительным знаком.

Узел условия не имеет дочерних узлов и возвращает тип сигнала в зависимости от выполнения предписанного условия. Имеет обозначение в виде овала. Декоратор - это узел контроля, который может быть переопределен разработчиком для реализации соответствующего поведения, обозначается знаком ромба.

На Рисунке 6 приведен пример структуры поведенческого дерева, выполняющего задачу движения робота к цели.



Рисунок 6 — Пример использования поведенческого дерева

Отметим преимущества поведенческих деревьев:

1. Модульность - малое число связей между отдельными состояниями. Это дает возможность создавать компоненты независимо друг от друга и встраивать их в систему
2. Иерархическая структура - действия подразделяются на уровни
3. Высокая степень реактивности (отзывчивости) - возможность быстро и эффективно реагировать на изменения в среде
4. Наглядное графическое представление системы

1.2.3 Частично наблюдаемые марковские процессы принятия решений

Марковские процессы принятия решений (МППР) и их дальнейшее развитие в виде частично наблюдаемых марковских процессов принятия решений (ЧНМППР) является еще одним подходом к управлению поведением робота. Их структура похожа на структуру конечных автоматов, однако МППР наделены вероятностными переходами между состояниями.

МППР определяется в виде кортежа (S, A, T, R) . Как и у конечного автомата МППР имеет множество возможных состояний S и множество стохастических переходов между ними T , каждому из которых присвоена некоторая вероятность. Также присутствуют множества действий A и наград

$R : S \times A \rightarrow \mathbb{R}$, которые соответствуют действиями.

В ходе функционирования робота в среде, оптимизируется функционал, который максимизирует ожидаемую награду, полученную роботом за время действия. В ходе оптимизационной задачи робот находит требуемое поведение π [9].

Различие между МППР и частично наблюдаемых МППР заключается в том, что текущее состояние системы может быть не определено и системе требуется проводить дополнительные наблюдения по которым оценивается вероятность быть в определенном состоянии [10]. Математически ЧНМППР описывается как кортеж (S, A, T, R, Z, O) , где помимо компонент МППР присутствуют также Z - дополнительные наблюдения, а O - это функция вероятности обнаружить Z в состоянии S .

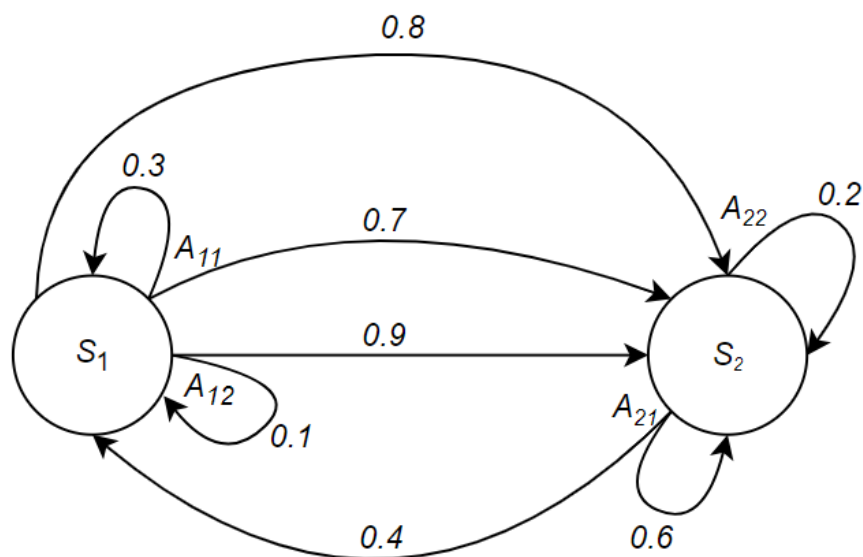


Рисунок 7 — Пример частично наблюдаемого МППР

ЧНМППР обычно используются для применения в неопределенном окружении, когда роботу требуется сталкиваться с неизвестными сценариями и подстраивать свое поведение для достижения цели. В основе данной системы принятия решений лежит вероятностный подход, а значит нет никаких гарантий что поведение системы будет детерминированным (будет одинаковым каждый раз при одних и тех же условиях), что может быть недопустимых в некоторых областях применения.

1.3 Навигация в динамическом окружении

Для применения методов навигации к условиям динамического окружения требуется модифицировать алгоритмы. Так как в случае наличия динамических объектов в среде требуется каким-то образом учитывать их движение во избежание критических ситуаций [11].

1.3.1 Обнаружение динамических объектов

Для обнаружения динамических объектов в окружении можно применить метод основанный на отделении заднего плана (представляющего статическую карту местности) и переднего плана (динамические объекты) [12]. Для этого используется представление окружения в виде карты занятости с отмеченными значениями стоимости, которая обрабатывается как изображение в градациях серого.

Операция выделения переднего плана используется во многих задачах компьютерного зрения, например для распознавания объектов. Существует множество алгоритмов, в данной работы мы рассмотрим метод с низкой вычислительной сложностью, использующий бегущие усредняющие фильтры:

$$F_{t+1}(x,y) = (1 - \alpha)F_t(x,y) + \alpha C_t(x,y), \quad (7)$$

где (x,y) - координаты пикселя изображения, F_t - изображение переднего плана в момент времени t , C_t - текущее входное изображение карты стоимости в момент времени t , α - коэффициент определяющий влияние карты стоимости на результат.

Для более точного выделения переднего плана используется комбинация из двух фильтров (медленный и быстрый) с добавлением медианного фильтра для 8 соседних пикселей:

$$F_f(t+1) = \beta((1 - \alpha_f)F_f(t) + \alpha_f C(t)) + \frac{1 - \beta}{8} \sum_{i \in N} F_{f,i}(t), \quad (8)$$

$$F_s(t+1) = \beta((1 - \alpha_s)F_s(t) + \alpha_s C(t)) + \frac{1 - \beta}{8} \sum_{i \in N} F_{s,i}(t), \quad (9)$$

где $F_f(t)$ и $F_s(t)$ - изображения переднего плана в момент времени t , α_f и α_s - коэффициенты влияний карты стоимости $C(t)$ на выходы фильтров, при чем

$$0 \leq \alpha_s \leq \alpha_f \leq 1. \quad (10)$$

А коэффициент β определяет соотношение между вкладом центральной клетки и влиянием соседних клеток на фильтры.

После применения данных фильтров к каждому пикселю применяется два пороговых фильтра для определения принадлежности пикселя к динамическому объекту:

1. Активация быстрого фильтра: $F_f(t) > c_1$
2. Разница между быстрым и медленным фильтром должна превышать порог c_2 : $F_f(t) - F_s(t) > c_2$

Результатом этой пороговой операции является бинарная карта, на которой отмечены все динамические препятствия. Далее применяются метод кластеризации с вычислением центров и границ объектов. Для этого успешно применяются методы компьютерного зрения из библиотеки *OpenCV*, например [13].

1.3.2 Слежение за динамическими объектами

Следующим пунктом реализуется слежение за выделенными объектами на бинарной карте.

Задача сопоставления объектов во времени является видом задачи о назначении (одна из фундаментальных задач комбинаторной оптимизации), которая решается венгерским алгоритмом [14]. Алгоритм минимизирует общее евклидово расстояние между центрами уже отслеживаемыми объектами и новым набором центров выделенных объектов.

Математическая формулировка выглядит следующим образом. Существует матрица стоимостей C , элементы которой $C_{i,j}$ определяют стоимость сопоставления ранее отслеженного объекта $i \in A$ с только что обнаруженным объектом $j \in B$. Функция стоимости представляет собой евклидову норму для расчета расстояния между объектами из двух множеств. Цель

оптимизации найти такую конфигурацию пар, которая будет обладать минимальной суммой стоимостей пар, то есть:

$$\min \sum_{i \in A} \sum_{j \in B} C_{i,j} X_{i,j}, \quad (11)$$

где X - булева матрица выбранных пар.

Теперь произведем оценку положения и скорости динамических объектов. Для решения такой задачи используется фильтр Калмана.

Фильтр Калмана - это алгоритм позволяющий оценивать некоторые неизвестные переменные с учетом поступления новых наблюдений с течением времени [15]. Фильтр Калмана оценивает вектор состояния линейных динамических систем в форме вход-состояние-выход.

Пусть дана дискретная модель в форме ВСВ:

$$\begin{cases} x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}, \\ y_k = Cx_k + \nu_k, \end{cases} \quad (12)$$

где A - матрица состояний, B - матрица управляющих воздействий, C - матрица наблюдений, x_k - вектор состояний на шаге k , y_k - вектор наблюдений на шаге k , u_k - вектор управления на шаге k , $w_k \sim \mathcal{N}(0, Q)$ - вектор внешних возмущений, $\nu_k \sim \mathcal{N}(0, R)$ - вектор помех измерений.

Фильтр Калмана производит оценку вектора состояния x_k на дискретном шаге k имея вектор начального состояния x_0 , значения векторов наблюдений y_1, y_2, y_3, \dots , а также матриц системы A, B, C, Q, R . В процессе работы метода повторяются два шага:

1. Шаг предсказания:

а) Оценка вектора состояния:

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_k \quad (13)$$

б) Оценка ковариационной матрицы:

$$P_k = AP_{k-1}A^T + Q \quad (14)$$

2. Шаг обновления:

а) Невязка вектора измерения:

$$\tilde{y}_k = y_k - C\hat{x}_k \quad (15)$$

б) Усиление фильтра Калмана:

$$K_k = P_k C^T (C P_k C^T + R)^{-1} \quad (16)$$

в) Обновленная оценка вектора состояния:

$$\hat{x}_{k_{upd}} = \hat{x}_k + K_k \tilde{y}_k \quad (17)$$

г) Обновленная оценка ковариационной матрицы:

$$P_{k_{upd}} = (I - K_k C) P_k \quad (18)$$

При адаптации фильтра Калмана к задаче слежения за объектами в двумерном пространстве вектор состояния примет вид [16]:

$$x_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} = \begin{bmatrix} p_{k-1} + v_{k-1} dt + \frac{1}{2} \tilde{a}_{k-1} dt^2 \\ v_{k-1} + \tilde{a}_{k-1} dt \end{bmatrix}, \quad (19)$$

где p и v - двумерные векторы положения и скорости в пространстве, \tilde{a} - ускорение приложенное к объекту.

Матричный вид:

$$x_k = A x_{k-1} + B \tilde{a}_{k-1}, \quad (20)$$

где \tilde{a}_{k-1} - неизвестный нам сигнал, $A = \begin{bmatrix} I_{2 \times 2} & I_{2 \times 2} dt \\ 0_{2 \times 2} & I_{2 \times 2} \end{bmatrix}$, $B = \begin{bmatrix} \frac{1}{2} I_{2 \times 2} dt^2 \\ I_{2 \times 2} dt \end{bmatrix}$.

В данном методе применяется модель постоянной скорости для объектов, а значит между дискретными шагами $k - 1$ и k скорость является константой.

Исходя из вида вектора состояния, внешние силы (w) вызывают постоянное ускорение (\tilde{a}), которое распределено по закону $\mathcal{N}(0, Q)$.

Значит ковариационная матрица распределения внешних возмущений:

$$Q = B B^T \sigma_a^2 = \begin{bmatrix} \frac{1}{4} I_{2 \times 2} dt^4 & \frac{1}{2} I_{2 \times 2} dt^3 \\ \frac{1}{2} I_{2 \times 2} dt^3 & I_{2 \times 2} dt^2 \end{bmatrix} \sigma_a^2, \quad (21)$$

где $\sigma_a^2 = \begin{bmatrix} \sigma_{ax}^2 & \sigma_{ay}^2 \end{bmatrix}^T$ - дисперсия вектора w .

Вектор наблюдений состоит только из положения объекта:

$$y_k = C x_k + \nu_k, \quad (22)$$

где ν_k - вектор помех измерений порождаемый распределением $\mathcal{N}(0, R)$, а матрица вектора измерений $C = \begin{bmatrix} I_{2 \times 2} & 0_{2 \times 2} \end{bmatrix}$, $R = I_{2 \times 2}$ - ковариационная матрица распределения помех измерений.

Начальные значения матрицы P могут быть выбраны следующим образом:

$$P = \begin{bmatrix} I_{2 \times 2} & O_{2 \times 2} \\ O_{2 \times 2} & 10 \times I_{2 \times 2} \end{bmatrix}, \quad (23)$$

где ковариация ошибок для скоростей установлена выше, поскольку они не измеряются.

1.3.3 Отображение динамических объектов на карте стоимости

Для того чтобы планировщик пути учитывал информацию о полученной скорости и положении отслеживаемых динамических объектов требуется отметить зоны штрафов вокруг динамических объектов на карте стоимости [17].

Значения штрафов вокруг объектов будут распределены по двумерному гауссовскому закону. При этом ковариационная матрица распределения должна иметь такие значения, чтобы область штрафа была больше для более быстрых объектов. Также учитывается направление скорости для отображения большей зоны штрафа вдоль направления движения объекта. Это достигается путем смешивания двух двумерных нормальных распределений.

Итак, имея набор динамических объектов с определенными положением и скоростью относительно глобальной системы координат, для каждого из них проведем следующие операции.

Пусть $c(x, y)$ - координаты центра объекта, определяем локальную систему координат с центром в c , осью абсцисс сонаправленной с вектором скорости и перпендикулярной ей осью ординат.

Тогда область штрафа определяется в виде функции:

$$\Phi_{c, \sum_{front}, \sum_{back}}(q) = \delta(x_q) \Phi_{c, \sum_{front}}(q) + (1 - \delta(x_q)) \Phi_{c, \sum_{back}}(q), \quad (24)$$

где $q = (x_q, y_q)$ - координаты точки в глобальной системе координат, $\Phi_{c, \sum_{front}}(q)$ - функция Гаусса для зоны штрафа перед объектом, $\Phi_{c, \sum_{back}}(q)$ - функция Гаусса для зоны штрафа позади объекта, $\delta(x)$ - функция параметра

для объединения гауссиан:

$$\delta(x) = \begin{cases} 1, & \text{если } x \geq 0 \\ 0, & \text{иначе} \end{cases} \quad (25)$$

Функции Гаусса имеют вид:

$$\Phi_{c,\Sigma}(q) = A \exp\left[-\frac{(\|q - c\|_2 \cos(\vartheta - \vartheta_c))^2}{2\sigma_x^2} - \frac{(\|q - c\|_2 \sin(\vartheta - \vartheta_c))^2}{2\sigma_y^2}\right], \quad (26)$$

где $q = (x_q, y_q)$ - координаты точки и $c = (x_c, y_c)$ - координаты центра объекта, ϑ - угол, образованный вектором положения точки относительно глобальной оси абсцисс, ϑ_c - угол между направлением движения динамического объекта и глобальной осью абсцисс, $A = 255$ - амплитуда, устанавливается максимальным значением карты стоимости, σ_x^2 и σ_y^2 диагональные значения ковариационной матрицы Σ , которые определяют форму зоны штрафов около объектов.

Для двух Гауссиан ковариационные матрицы определяются так:

$$\sum_{front} = \begin{pmatrix} \sigma_{x_{front}}^2 & 0 \\ 0 & \sigma_{y_{front}}^2 \end{pmatrix}; \sum_{back} = \begin{pmatrix} \sigma_{x_{back}}^2 & 0 \\ 0 & \sigma_{y_{back}}^2 \end{pmatrix} \quad (27)$$

Таким образом, варьируя значения σ_x^2 и σ_y^2 , можно изменять форму зоны штрафов.

Также, для учета скорости объекта дисперсии могут быть переопределены следующим образом:

$$\begin{cases} \sigma_{x_{front}}^2 = (1 + r)\sigma_{x_{front}}^2 \\ \sigma_{y_{front}}^2 = (1 - \frac{r}{2})\sigma_{y_{front}}^2 \\ \sigma_{x_{back}}^2 = (1 - r)\sigma_{x_{back}}^2 \\ \sigma_{y_{back}}^2 = (1 - \frac{1}{4})\sigma_{y_{back}}^2 \end{cases}, \quad (28)$$

где $r = \frac{v_{cur}}{v_{max}}$ - соотношение текущей скорости и максимальной скорости объекта.

2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

В этой главе описываются принципы работы с ROS2 (Robotic Operating System) - программное обеспечение, использующееся для реализации алгоритмов навигации, моделирования их работы и дальнейшем использовании на физическом роботе. Также рассмотрена навигационная система Nav2 для задач автономной навигации, среда для симуляции Gazebo и аппаратной платформе Turtlebot3, выбранной для тестирования решения, разработанного в данной работе. В конце главы дан список с версиями используемого ПО.

2.1 ROS2

ROS2 - это программное обеспечение с открытым исходным кодом для создания приложений для роботов. Данное ПО не является операционной системой, поскольку для ее работы необходима базовая ОС. ROS2 может поддерживать связь между различными программами (называемыми узлами) с помощью стандартизированных интерфейсов и сетевых протоколов. Узлы могут взаимодействовать между собой через 3 типа коммуникационных архитектур:

1. Темы (topics) - это каналы, по которым узлы обмениваются сообщениями. Общение через темы происходит по логике публикация-подписка (publish-subscribe). Узел-издатель публикует сообщение в теме, и все узлы, подписанные на эту тему, получают это сообщение. Особенностью данного типа коммуникации является то, что по нему обычно передаются непрерывный поток данных (например, данные с датчиков).
2. Сервисы (services) - тип коммуникации, реализующий синхронную связь между узлами по принципу запрос-ответ (request-response). Узел-сервер отвечает только при поступлении запроса от узла-клиента, который отправляет запросы и получает на них соответствующие ответы. Соединение между двумя узлами теряется после того как был обработан один запрос.

3. Действия (actions) - тип асинхронной коммуникации, использующийся для выполнения продолжительных во времени действий. Клиенты действий отправляют запрос на сервер для достижения некоторой цели и помимо получения результата в конце выполнения, во время выполнения действия сервер отправляет клиенту информацию о ходе выполнения.

Одним из главных плюсов ROS2 является простота интеграции пользовательских библиотек в основное ПО, а также наличие множества высококачественных библиотек с открытым исходным кодом для решения различных задач робототехники. Таким образом ROS2 одним из наиболее популярным ПО для создания роботизированных приложений [18].

2.2 Навигационная система Nav2

Навигационная система Nav2 состоит из множества пакетов, решающих различные задачи навигации мобильным роботом. Данное ПО достаточно распространено среди разработчиков и является стандартным способом для реализации мобильной навигации с широким спектром поддерживаемых роботов. Поддерживаются такие типы роботов, как голономные, дифференциально-приводные, человекоподобные (имеющие ноги) и с приводом Аккермана (подобные автомобилю). Чтобы мобильный робот мог использовать пакеты Nav2, он должен соответствовать некоторым условиям. Настройка требует от мобильного робота наличия датчика лазерного сканирования, источника одометрии (например, инерциального измерительного блока (IMU) и/или энкодеров колес), карты с информацией о свободных и занятых пространствах (карта занятости), а также набора преобразований, связывающих различные узлы робота и окружение, для планирования и навигации. Архитектура Nav2 представлена на Рисунке 8.

Пакеты Nav2 содержат инструменты для сохранения и загрузки карт занятости, локализации робота, планирования траектории, реализации построенной траектории, предоставления локальной и глобальной карт стоимости, реализации некоторого поведения и восстановления робота [19]. В эти пакеты часто интегрируется метод одновременной локализации и картографирования (SLAM) из пакета SLAM Toolbox [20]. Возможности планирования

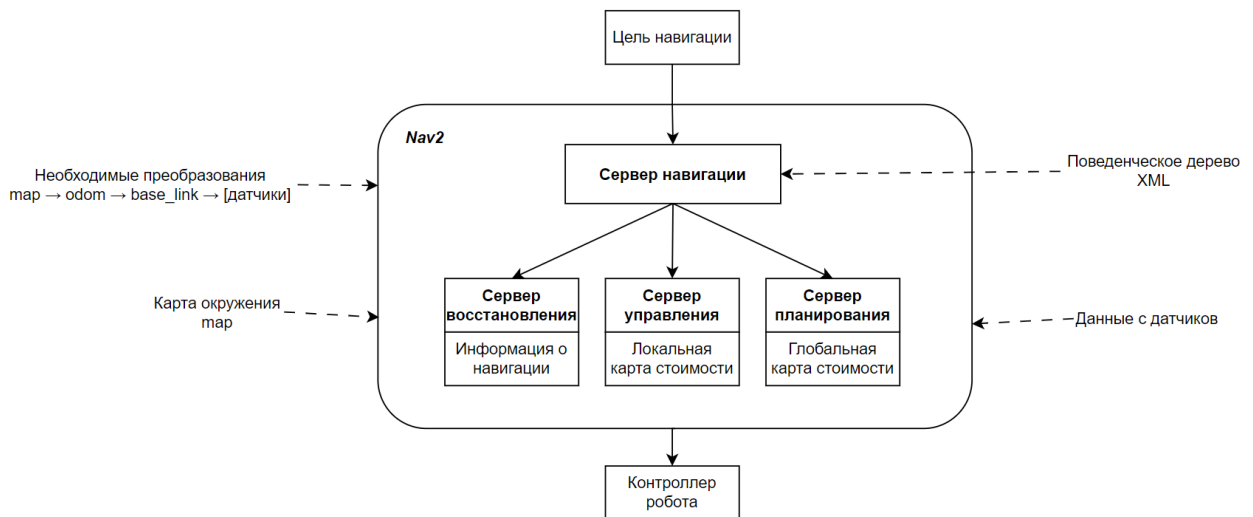


Рисунок 8 — Архитектура Nav2

и слежения за траекторией, которые соответствуют глобальному и локальному планировщику, дополнительно поддерживаются готовыми плагинами планирования, которые используют алгоритмы A* и Дейкстры для глобального планирования траектории и подходы динамического окна (DWB) или эластичных лент (TEB) для построения и выполнения локального плана. Последовательность работы системы осуществляется по иерархической структуре, реализованной с помощью поведенческого дерева. Основное поведенческое дерево вызывает действия соответствующих планировщиков друг за другом в асинхронном режиме, как показано на Рисунке 9.

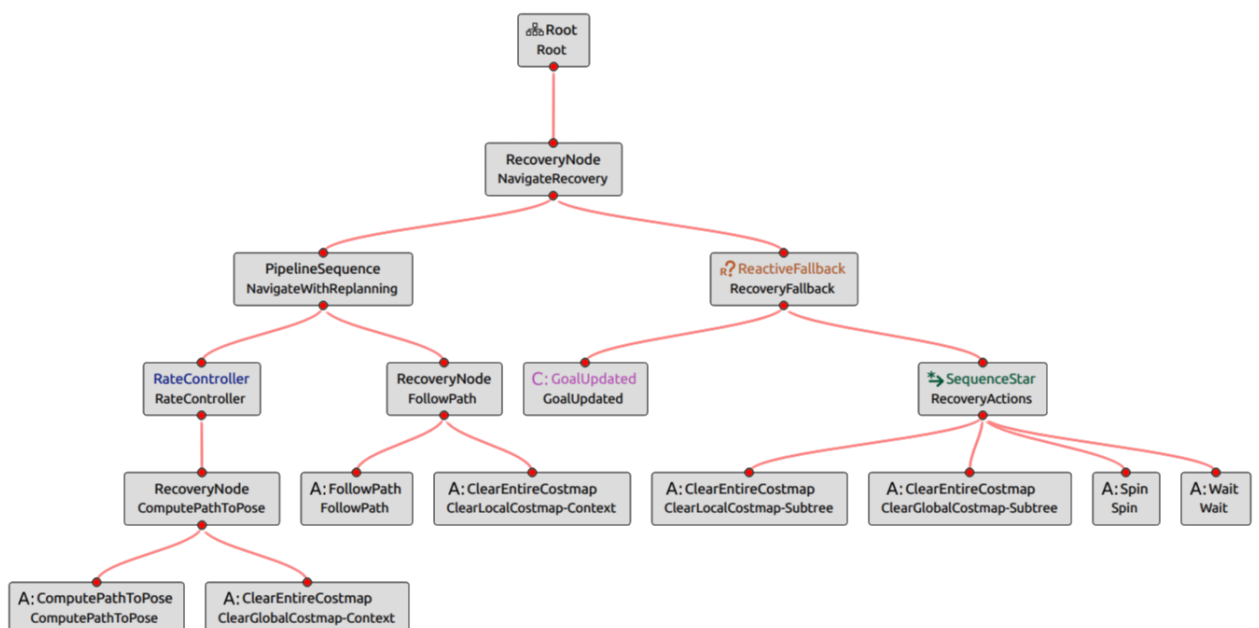


Рисунок 9 — Основное поведенческое дерево Nav2

Данной поведенческое дерево используется Nav2 по умолчанию и имеет набор уже реализованных поведений восстановления (вращение (spin), ожидание (wait), очистка карты (clear map)). Эти поведения выполняются, если робот не может продвинуться к поставленной цели.

Также в Nav2 используется система управления жизненным циклом узлов (представленная в ROS2) для контролируемого запуска, активации, деактивации и выключения узлов. Подход, основанный на жизненном цикле, позволяет системе следить за запуском, выполнением и сбоями узлов системы.

2.3 Модель робота Turtlebot3

Turtlebot3 - это стандартная мобильная исследовательская платформа для управления роботом. Робот хорошо интегрирован в экосистему ROS и используется как система для разработки и интеграции новых методов навигации. Робот оснащен двумя двигателями с подключенными к ним энкодерами (дифференциальный привод). Третье колесо стабилизирует робота (т.н. caster wheel).

Производитель ROBOTIS предоставляет модель с открытым исходным кодом для моделирования робота в физических симуляторах, таких как Gazebo, что позволяет ускорить разработку и тестирование создаваемых робототехнических приложений.

Мобильный робот Turtlebot3 обладает всеми необходимыми датчиками для использования Nav2. Технические характеристики робота приведены в Таблице 1.

Характеристика	Значение
Максимальная линейная скорость, м/с	0.26
Максимальная угловая скорость, рад/с	1.82
Вес, кг	1.8
Габариты робота, мм	281x306x141
Одноплатный компьютер	Raspberry Pi 4
Процессор	32-bit ARM Cortex-M7
Лазерный дальномер	360° LDS-02
RGB-камера	Raspberry Pi Camera
Гироскоп	3-х осевой
Акселерометр	3-х осевой

Таблица 1 — Основные технические характеристики Turtlebot3 (waffle-pi)

2.4 Симуляционная среда Gazebo

Разработка и тестирование системы осуществляется с помощью симулятора Gazebo и модели Turtlebot3. Gazebo хорошо интегрирован в ROS2 и предлагает множество интерфейсов ROS2 для управления симуляцией.

Симулятор Gazebo точно моделирует физику робота и может публиковать показатели одометрии в соответствующую тему, также как и показания лазерного сканера и данные с инерциального измерительного блока с помощью соответствующих плагинов.

Симулятор обеспечивает более легкую повторяемость тестовых примеров, так как в любой момент можно создать различные препятствия и вызвать сбои датчиков, что приводит к ускорению разработки в целом.

Использование симулятора позволяет замедлить или ускорить время работы со средой. Регулирование времени позволяет более тщательно наблюдать за быстрыми процессами в замедленном режиме, а также наблюдать за устойчивостью системы в течение времени при длительных испытаниях.

2.5 Behaviortree.CPP

ПО Navigation2 использует поведенческое дерево для реализации слоя планирования. Используемое дерево является расширением библиотеки

Behaviortree.CPP.

Эта библиотека предлагает способ создания, выполнения, мониторинга и редактирования поведенческих деревьев.

В дополнение к типам управляющих узлов, упомянутых в разделе 1.2.2, библиотека добавляет в каталог управляющих узлов концепцию реактивности. Реактивные узлы последовательности и реактивные селекторы отличаются от обычных тем, что обрабатывают узлы, которые возвращают состояние "Выполняется". Вместо того чтобы ждать пока узел завершит действие, вся последовательность перезапускается, что полезно для непрерывной проверки условия.

Также данная библиотека реализует способ коммуникации между узлами дерева. Для того чтобы узлы дерева могли общаться, библиотека предоставляет разработчику две возможности:

1. Любой из узлов может использовать общую доску, реализованную в виде словаря (ключ/значение), которую могут читать и записывать все узлы дерева.
2. Любые два узла могут быть соединены через порты, что позволяет напрямую связываться между двумя узлами через ключ/значение.

2.6 Groot

Groot - это программа для создания, редактирования, мониторинга и отладки поведенческих деревьев с графическим интерфейсом. Программа позволяет создавать деревья поведения в XML-файлах, которые могут быть напрямую загружены в библиотеку *BehaviorTree.CPP* и использоваться ею.

Groot может следить за исполнением деревьев поведения в реальном времени и позволяет анализировать реакцию дерева на различные сценарии. На Рисунке 10 показан небольшой фрагмент поведенческого дерева.

В этом графическом интерфейсе узлы условий помечены буквой "С" перед названием узла, а узлы действия - буквой "А". Цвета указывают на поток управления и возвращаемые результаты соответствующих узлов. Зеленая рамка вокруг узла показывает, что сигнал управления был обработан и вернул "Успех" а красная рамка сигнализирует о том, что узел вернул "Неудачу".

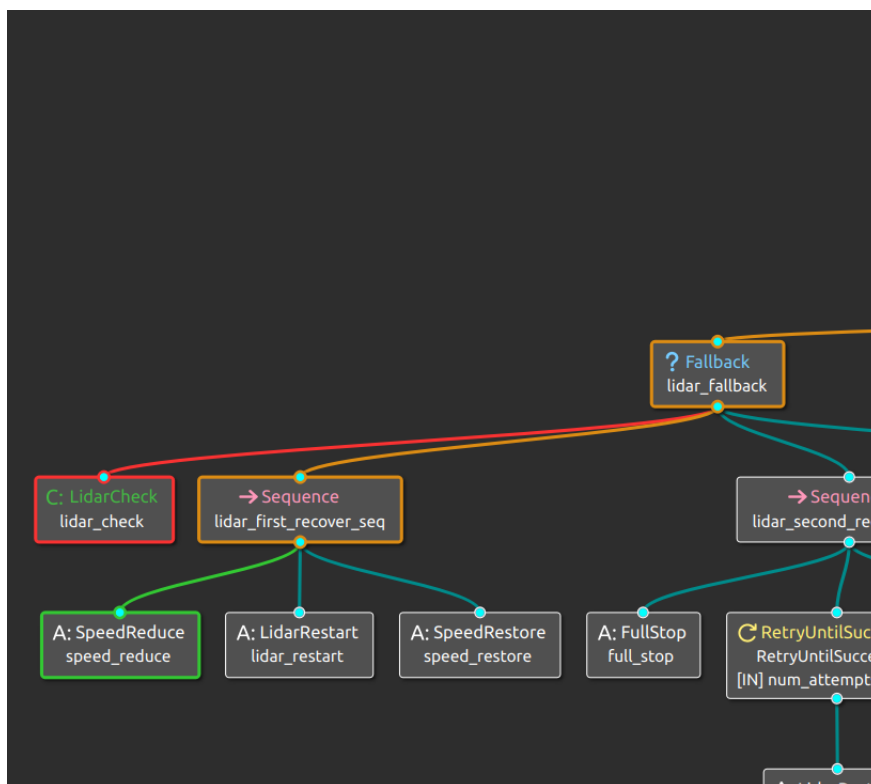


Рисунок 10 — Основное поведенческое дерево Nav2

В изображенном примере узел "lidar_check" вернул "Неудачу" далее селектор начал выполнять узел последовательности под названием "lidar_first_recover_seq". Поток управления еще не вернулся обратно в последовательность, поэтому граница последовательности окрашена в оранжевый цвет (состояние "Выполнение"). Изображенное дерево выполняет узел действия в нижней части рисунка с именем "speed_reduce".

2.7 Используемое ПО

Версии программного обеспечения, использованные для реализации и тестирования системы, перечислены ниже в Таблице 2.

Название	Версия
ОС	Linux, Ubuntu 22.04.4 LTS
ROS	ROS2 Humble
Nav2	Nav2, humble-devel (с внесенными изменениями)
Симулятор	Gazebo 11
Робот	Turtlebot3 waffle-pi (с внесенными изменениями)
БТ	BehaviorTree.cpp 3.8
С++	С++ 14
Python	3.10.12
Система сборки	CMake, colcon
OpenCV	4.9.0
SciPy	1.13.0
NumPy	1.26

Таблица 2 — Используемое программное обеспечение

3 РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1 Предпосылки разрабатываемой системы

Nav2 является отличной основой для реализации автономной навигации для мобильных роботов.

Однако, если сравнивать функциональные возможности данного ПО с предлагаемой архитектурой автономной системы, представленной на Рисунке 2, то можно увидеть, что в Nav2 отсутствует компонент системного контроля.

Отсутствие компоненты системного контроля приводит к ограниченному набору обстоятельств, в которых робот может надежно выполнять навигацию. Одна из основных проблем, с которой сталкиваются мобильные роботы - создание надежного представления среды. Непредвиденные события могут серьезно ограничить надежность представления среды.

Примерами таких событий, которые в настоящее время не обрабатываются должным образом, являются проскальзывание колес, изменение ориентации под воздействием внешних факторов или необнаруживаемые препятствия. Такие ситуации провоцируют небезопасное поведение и требуют вмешательства человека-оператора и восстановления работоспособности робота вручную.

Также для навигации в динамическом окружении в системе Nav2 отсутствуют компоненты учета динамических объектов для более безопасного функционирования вблизи людей и других агентов. Об актуальности данной проблемы говорит то, что в рамках программы Summer of Code 2021, участником предлагалась задача по разработке решения динамического обнаружения препятствий для Nav2 на основе 2D LIDAR [21].

Итак, для успешной навигации в динамическом окружении в разрабатываемую систему требуются внести следующие изменения:

1. Увеличение степени автономности - так как среда является динамической, то возможность доступа оператора к роботу уменьшается и является важным уменьшить степень вовлеченности оператора при работе робота

2. Слежение за важными узлами робота и возможность быстро реагировать на критические ситуации - робот должен гарантировать безопасность объектов в окружении, поэтому требуется определять ситуации (например, поломка LIDAR) при которых робот теряет возможность безопасного функционирования
3. Планирование пути с учетом динамических объектов - требуется изменить систему навигации для учета динамических объектов

Требуемая система принятия решений разрабатывалась с учетом пунктов выше.

3.2 Стандартная архитектура для навигации

В качестве основы была взята стандартная архитектура для решения задач навигации при использовании ROS2. Архитектура состоит из следующих компонент:

- Nav2 - набор пакетов для навигации;
- Turtlebot3 - модель мобильного робота;
- AMCL - пакет для локализации робота в 2D, основан на методе Монте-Карло;
- SLAM Toolbox - пакет для задачи SLAM, используется для изначального картирования местности;
- Map server - пакет для выполнения операций с построенными картами местностей (сохранение, загрузка, редактирование).

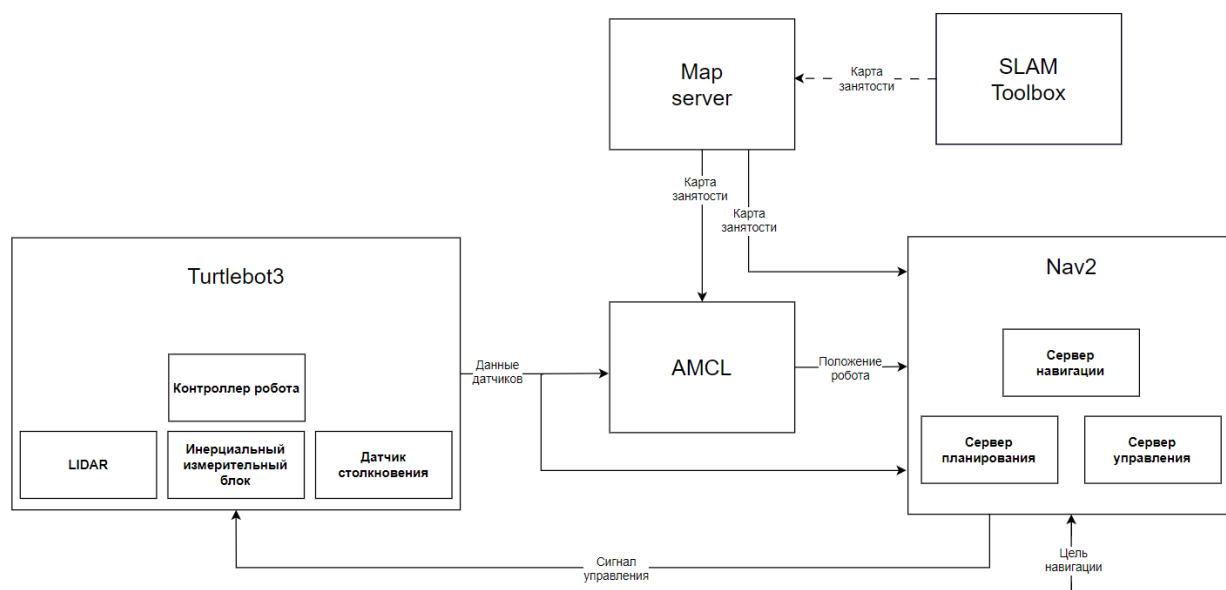


Рисунок 11 — Архитектура стандартной системы навигации Nav2

3.3 Компоненты разработанной системы принятия решений

Разработанная система принятия решений включает в себя две подсистемы:

1. Система контроля - выполняет задачу контроля за функционированием робота. В основе системы - поведенческое дерево, которое позволяет переключать различные поведения в соответствии с принципами модульности, иерархичности, реактивности и наглядности. Система контроля анализирует следующее:
 - а) Состояние датчика LIDAR;
 - б) Состояние датчика IMU;
 - в) Состояние одометрии;
 - г) Уровень заряда батареи;
 - д) Состояние планировщика траектории;
 - е) Столкновения с объектами;
 - ж) Ориентация в пространстве.
2. Система учета динамических объектов - выполняет задачи:
 - а) Разделение объектов на карте местности на статические и динамические

- б) Слежение за обнаруженными динамическими объектами и оценка их скоростей
- в) Добавление области стоимости для динамических объектов на карту стоимости



Рисунок 12 — Компоненты разработанной системы принятия решений

3.4 Структура программной реализации системы

Для удобства проектирования исходного кода системы были созданы четыре рабочих пространства (рабочее пространство - это каталог, содержащий пакеты ROS2):

- *tb3_ws* - рабочее пространство, содержащее пакеты описывающие модель робота Turtlebot3, позволяющие выполнять Nav2 на роботе, а также проводить симуляцию в Gazebo. Данные пакеты были скачаны и собраны с [22] для того чтобы была возможность модификации исходных файлов
- *nav2_ws* - рабочее пространство для пакетов Nav2. Аналогично пакетам из *tb3_ws*, были скачаны и собраны с [23] для дальнейшего внесения изменений
- *supervisory_sys_ws* - рабочее пространство для реализации системы контроля робота

- *dyn_obs_avoid_sys_ws* - рабочее пространство для реализации системы учета динамических объектов

3.5 Система контроля

3.5.1 Архитектура системы

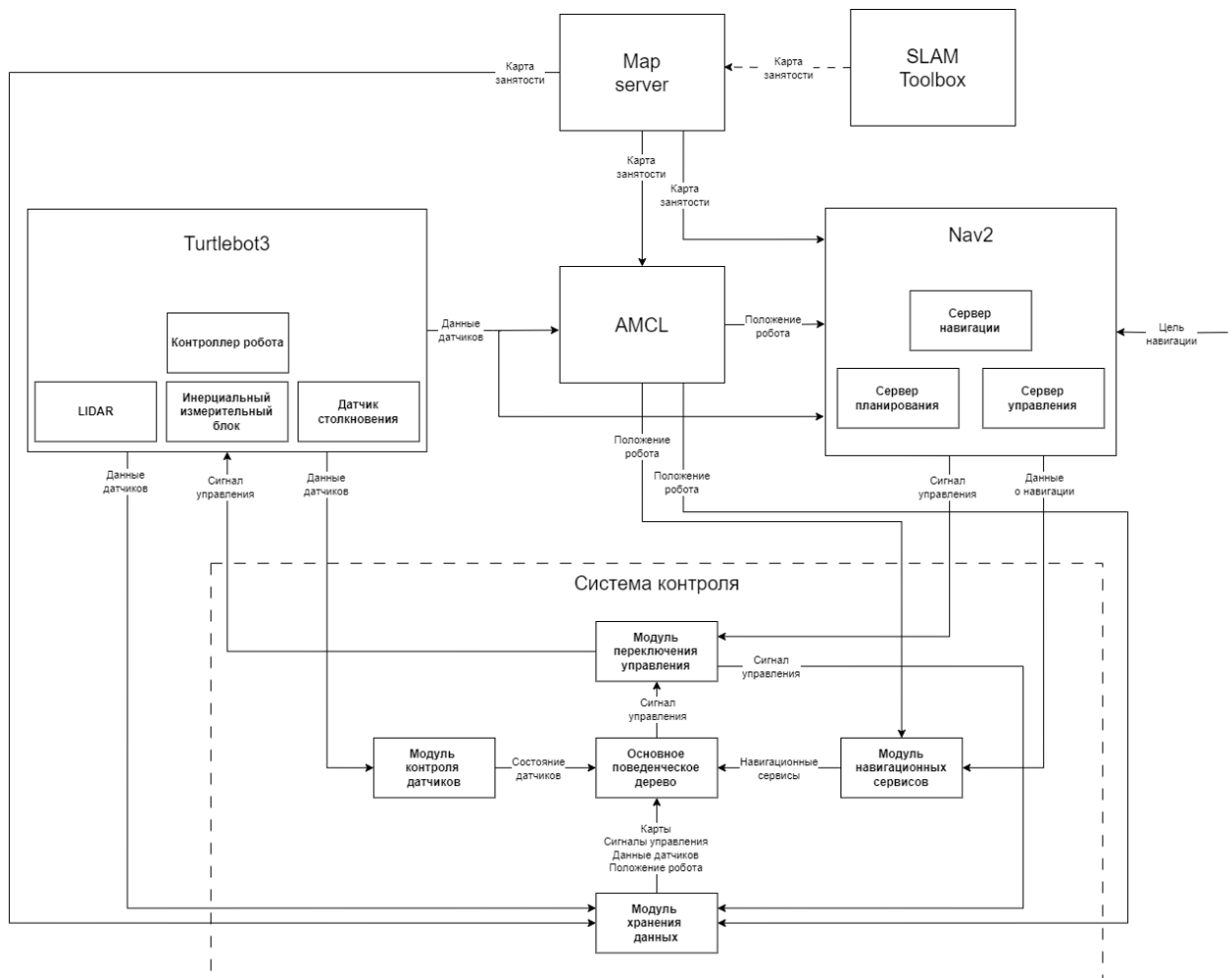


Рисунок 13 — Архитектура системы контроля

На Рисунке 13 изображена архитектура стандартной системы навигации с внедрением разработанной система контроля.

Система контроля включает 5 ключевых модулей, каждый из которых реализован в качестве отдельного ROS пакета. Описание каждой компоненты приведено ниже в соответствующих пунктах.

3.5.2 Модуль контроля датчиков

Данный модуль подписывается на темы (topics) `/odom`, `/imu`, `/scan`, из которых получает данные с сенсоров. Если сообщения перестают поступать, то модуль определяет это и обновляет свое внутреннее состояние, записывая информацию о том, что определенный узел (node) имеет неполадки. Информацию о неполадках модуль предоставляет в виде реализованных сервисов. Через эти сервисы основное поведенческое дерево получает данные о работе датчиков.

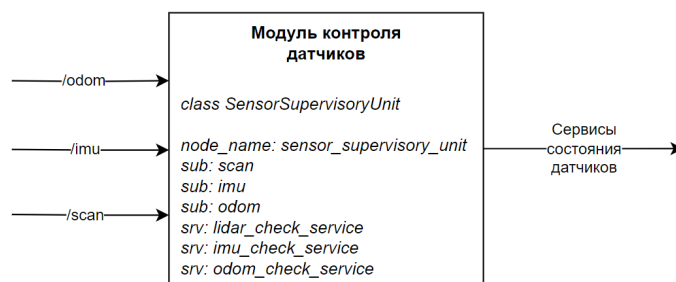


Рисунок 14 — Модуль контроля датчиков

3.5.3 Модуль навигационных сервисов

Данный модуль предоставляет сервисы, которые необходимы для проверки системы при навигации робота.

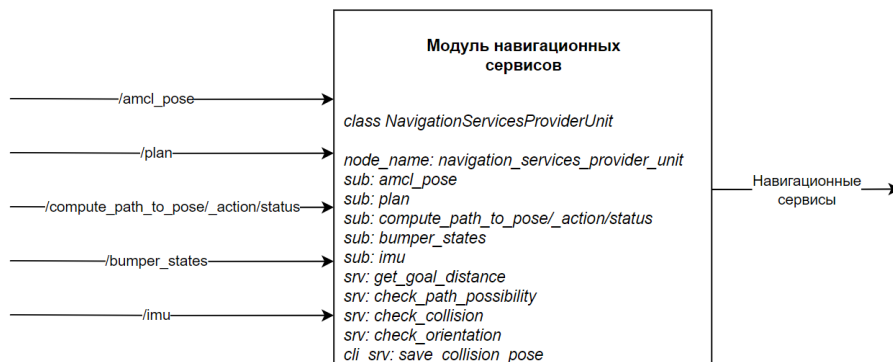


Рисунок 15 — Модуль навигационных сервисов

Имя	Тип данных	Объем данных
LIDAR	sensor_msgs/msg/Laserscan	3 секунды
IMU	sensor_msgs/msg/Imu	3 секунды
Одометрия	nav_msgs/Odometry	3 секунды
Положение робота	geometry_msgs/Pose	2 секунды
Положение при столкновении	geometry_msgs/Pose	Только последнее
Карта занятости	nav_msgs/OccupancyGrid	Только последняя
Сигналы управления	geometry_msgs/Twist	3 секунды

Таблица 3 — Типы и объем данных для хранения

3.5.4 Модуль хранения данных

Модуль хранения данных реализует функционал сбора данных от всех компонентов системы и предоставления их основному поведенческому дереву через соответствующие сервисы. Этот модуль подписывается на все потоки данных датчиков и сохраняет сообщения в очереди. Это означает, что самые последние данные помещаются в массив, а если массив превышает заданную длину, самые старые данные из него удаляются. Таким образом, вычислительная и сетевая нагрузки значительно снижаются.

Типы и количество хранимых данных указаны в Таблице 3.

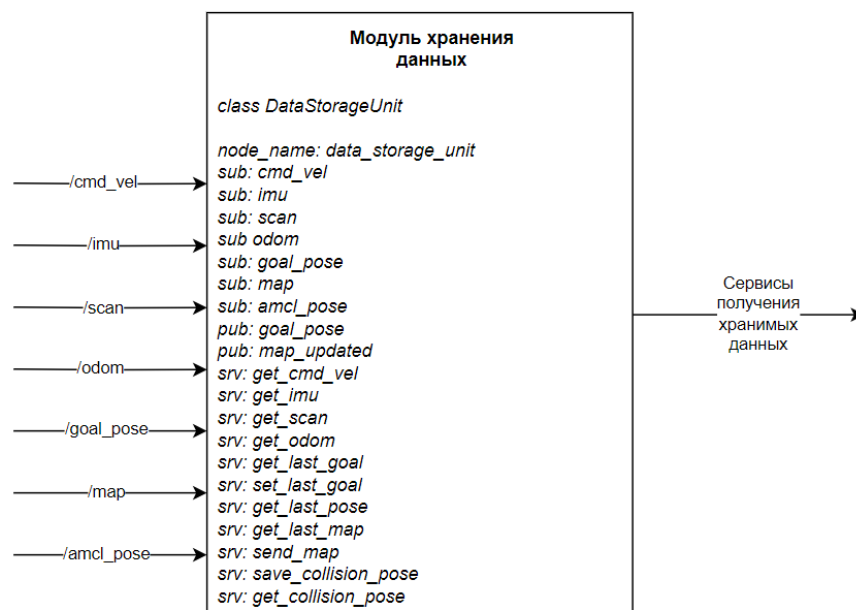


Рисунок 16 — Модуль хранения данных

3.5.5 Модуль переключения управления

Данный модуль анализирует сообщения от системы Nav2 и основного поведенческого дерева. Сообщения с основного поведенческого дерева имеют больший приоритет, соответственно если данный модуль обнаруживает сообщения в теме *cmd_vel_bt*, то сообщения с темы *cmd_vel_nav* не будут пропускаться.

Также данный модуль реализует сервисы по полной остановке и изменению скорости робота (уменьшение и увеличение). Эти сервисы вызываются когда система обнаруживает сбой некоторых компонент и, для безопасности, требуется ограничить скорость робота или же совсем его остановить.

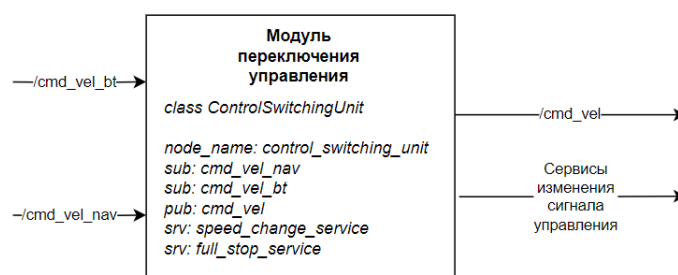


Рисунок 17 — Модуль переключения управления

3.5.6 Основное поведенческое дерево

Данный модуль реализует алгоритм принятия решений всей системы контроля. В основе лежит структура поведенческого дерева, реализованная с помощью библиотеки *BehaviorTree.cpp*.

Структура дерева представляется в виде XML-файла, а поведение отдельных действий объявляется в соответствующих *.hpp* файлах, которые затем регистрируются как узлы деревьев.

Ниже представлена высокоуровневая схема дерева.

Листья представляют собой поддеревья, которые будут рассмотрены в пунктах далее. В качестве главного узла управления выбран последовательный узел.

Соответственно, все потомки последовательного узла расположены в порядке увеличения сложности функционирования. То есть вначале идут низкоуровневые компоненты, без работы которых функционирование системы

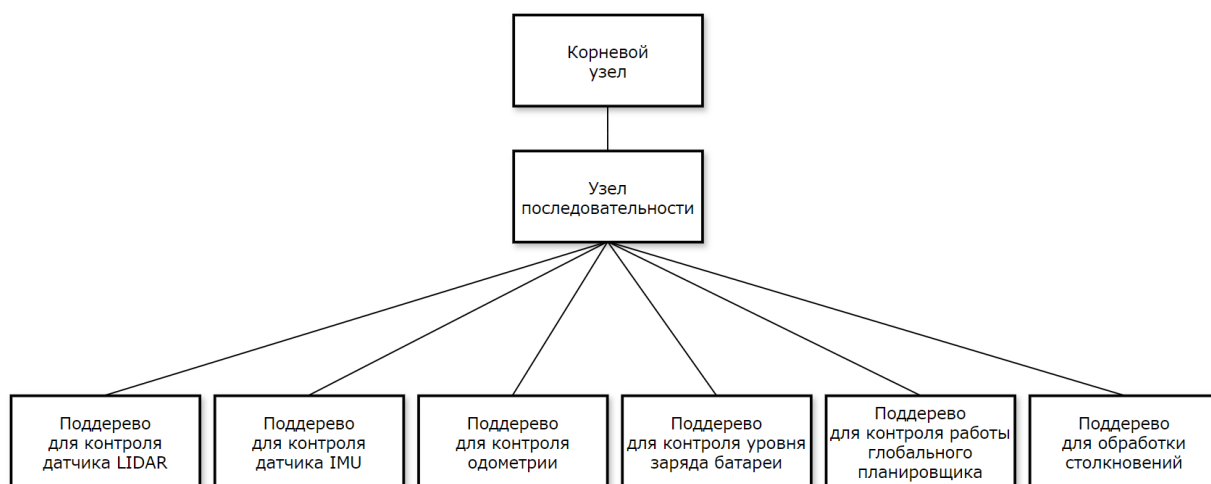


Рисунок 18 — Высокоуровневый граф основного поведенческого дерева

невозможно. А справа более высокоуровневые, которые требуют выполнения всех компонентов левее от себя.

Поддеревья для контроля датчиков

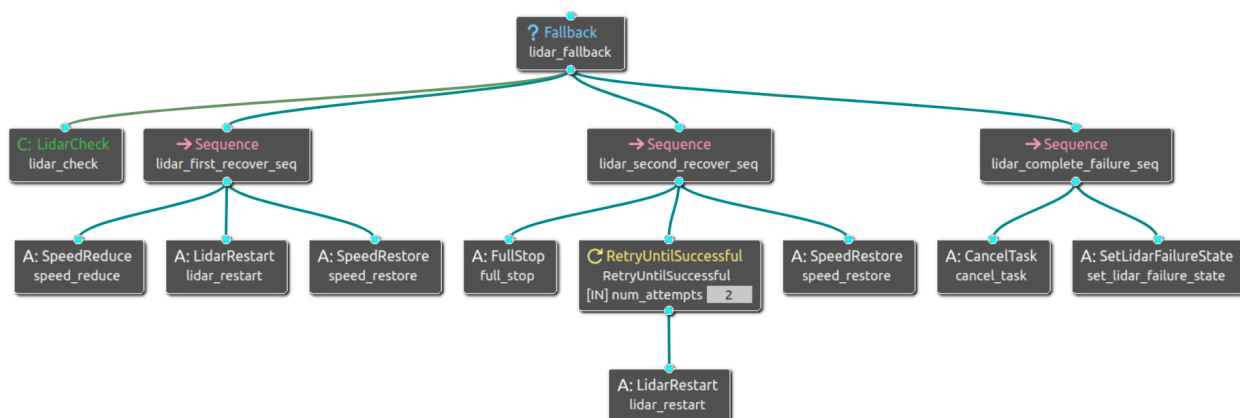


Рисунок 19 — Поддереву для контроля датчика LIDAR

На Рисунке 19 находится схема поддерева для проверки работоспособности датчика LIDAR (поддеревья для датчиков IMU и проверки одометрии аналогичны).

Модуль основного поведенческого дерева вызывает сервисы проверки датчиков в узлах-условиях (condition nodes), реализованные в модуле контроля датчиков. Если узел условия *collision_check* (вызывающий сервис проверки) возвращает "Успех" то датчик LIDAR в норме и проход данного поддерева завершается. Иначе, сигнал прохода идет в узел последовательности

collision_handle_seq.

Обработка данной ситуации выполняет следующие действия:

1. Снижение скорости до минимально допустимой, попытка перезагрузить модуль сенсора и, в случае успешной перезагрузки, восстановление скорости
2. Если же после первой перезагрузки модуля сенсора, сенсор продолжает не работать, следует полная остановка робота, далее производятся 2 попытки перезагрузки, и в случае если сенсор заработал - восстановление скорости
3. Если же 2 предыдущие последовательности не смогли вернуть функционирование сенсору - происходит отмена всех целей навигации робота и введение системы в состояние ошибки

Время которое должно пройти с последнего сообщения от датчика для того чтобы модуль контроля датчиков объявил о неисправности датчика равно 1 секунде. По истечению данного времени робот будет недолго продолжать движение к цели. Однако если не удастся восстановить функционирование датчика - робот больше не сможет безопасно функционировать, а значит следует его полная остановка.

Такое поведение реализовано для датчиков LIDAR и IMU, а также для системы одометрии. Так как данные компоненты являются критически важными для безопасного функционирования, то в случае любых неисправностей навигация прекращается.

Подерево для контроля уровня заряда батареи

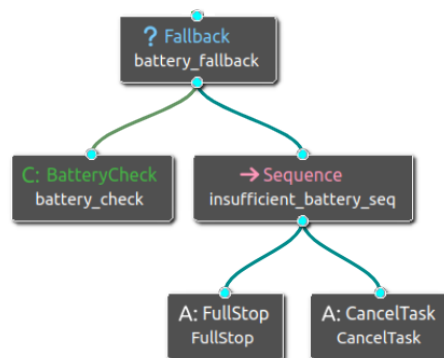


Рисунок 20 — Подерево для контроля уровня заряда батареи

Данное поддерево реализует проверку заряда батареи для успешного выполнения задачи навигации. Проверка основывается на значениях текущего заряда батареи, среднего потребления батареи и длины пути до цели. Задача навигации будет отменена, если будет установлено, что уровня заряда не хватит до достижения цели.

Для реализации данного поведения был создан ROS пакет для симуляции компонента батареи у мобильного робота (модуль симуляции батареи). У данного модуля есть сервис через который модуль основного поведенческого дерева может получить уровень заряда и, проведя операции, определить достаточен ли уровень.

Поддерево для контроля работы глобального планировщика

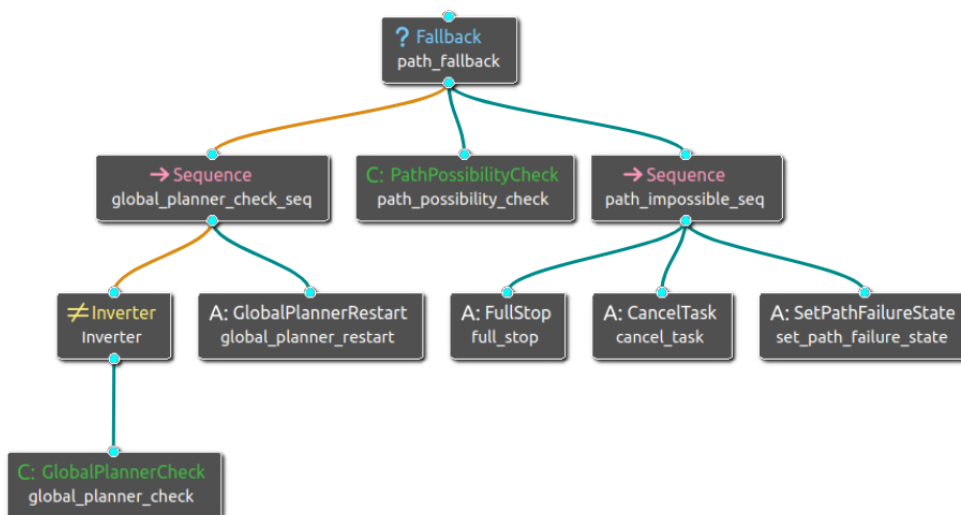


Рисунок 21 — Поддерево для контроля работы глобального планировщика

Данное поддерево реализует поведение для проверки работоспособности планировщика траектории. Так как узел *GlobalPlanner* модуля Nav2 является узлом типа *LifecycleNode*, то информацию о его состоянии можно получить с помощью вызова соответствующего сервиса.

В основе *LifecycleNode* лежит следующий конечный автомат состояний: Следовательно обращаясь непосредственно к соответствующему сервису узла глобального планировщика мы получаем информацию в каком состоянии он находится.

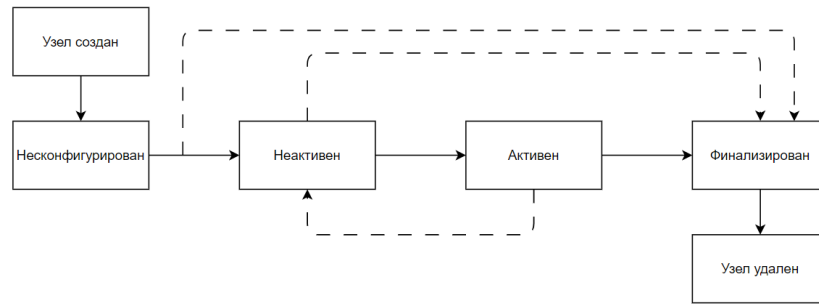


Рисунок 22 — Конечный автомат узла жизненного цикла

Также при проходе данного поддерева происходит проверка на достижимость поставленной точки на карте (цели навигации) с помощью просмотра состояния вызванного действия (там тоже используется конечный автомат) *NavigateToPose* узла *bt_navigator* модуля Nav2.

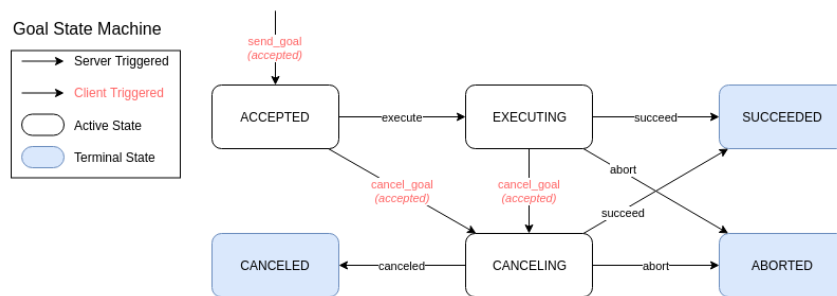


Рисунок 23 — Конечный автомат для действия в ROS2

Таким образом реализуется поведение, которое проверяет работу планировщика пути и может устранить автономно некоторые неполадки или же уведомить об ошибке.

Поддерево для обработки столкновений

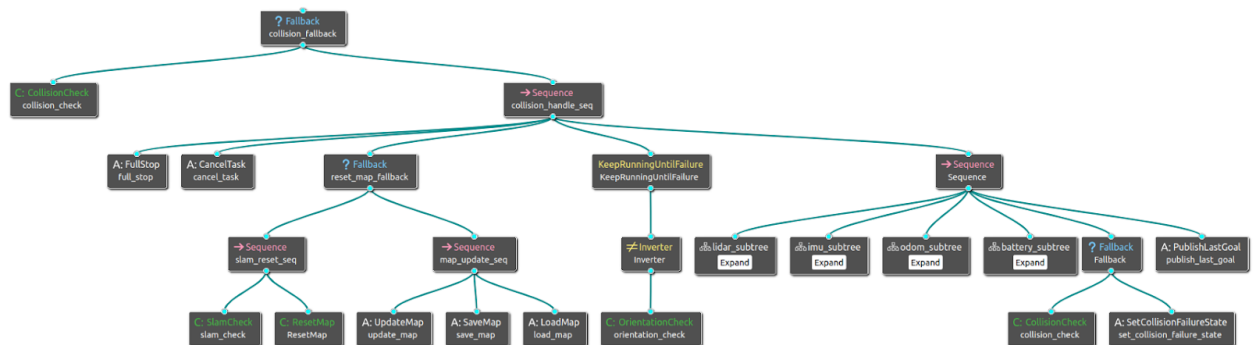


Рисунок 24 — Поддерево для обработки столкновений

Данное поддерево реализует поведение при столкновениях робота с объектами.

Необходимо обнаруживать столкновения, обновлять карту местности (добавление препятствия) и перестраивать путь до цели с учетом новой карты.

Для обнаружения препятствий к передней части робота был добавлен датчик столкновений, который способен детектировать столкновения перед собой и публиковать информацию в тему */bumper_states*. Далее на это подписывается модуль навигационных сервисов и предоставляет сервис через который основное поведенческое дерево узнает был ли робот в состоянии столкновения с объектом в последние 0.5 секунд или нет. При столкновении модуль навигационных сервисов запоминает положение и ориентацию робота в последний момент перед столкновением для того, чтобы далее передать эту информацию для добавления препятствия на карту местности.

После того как было обнаружено столкновение, происходит полная остановка робота и отмена всех целей навигации. Затем проверяется функционирует ли робот в режиме SLAM или же карта местности была заранее загружена в память с помощью Map server. В зависимости от этого происходит либо полный сброс карты, либо же обновление карты. Действие *UpdateMap* соответственно добавляет препятствие в виде квадрата 0.25x0.25 м перед роботом.

Далее робот переводится в режим ожидания до тех пор, пока не будет восстановлена его ориентация в пространстве.

Проверка ориентации в пространстве происходит через анализ данных, поступающих с инерциального измерительного блока. Датчик IMU посылает сигналы типа данных *sensor_msgs/Imu*. В объектах этого класса присутствуют следующие поля:

- *Header header*
- *geometry_msgs/Quaternion orientation*
- *float64[9] orientation_covariance*
- *geometry_msgs/Vector3 angular_velocity*
- *float64[9] angular_velocity_covariance*
- *geometry_msgs/Vector3 linear_acceleration*
- *float64[9] linear_acceleration_covariance*

Для проверки ориентации мы используем поле *orientation*, записанное в виде кватерниона. Преобразуем его в матрицу поворота с помощью библиотеки *tf2*, далее из этой матрицы поворота извлекаем углы крена, тангажа и рыскания и проверяем чтобы они лежали в диапазоне от -6 до 6 градусов. Данную проверку осуществляет модуль навигационных сервисов при вызове соответствующего сервера.

После восстановления ориентации происходит проверка всех важных компонент системы (LIDAR, IMU, одометрия, батарея) и если хотя бы один из модулей работает некорректно, робот переходит в состояние неисправности. Такой случай означает то, что в результате столкновения был поврежден какой-то критически важный компонент.

Если же все критически важные компоненты работают исправно, то происходит еще одна проверка на состояние столкновения. В данной работе для гарантий безопасности робот не будет продолжать движение, если датчик столкновения фиксирует контакт с другим объектом. Движение может продолжиться только в случае отсутствия контакта.

Соответственно, при отсутствии контакта, восстанавливается прошлая цель навигации и планировщик пути строит новую траекторию с учетом отмеченного на карте объекта столкновения.

3.5.7 Дополнительные модули

Модуль симуляции батареи

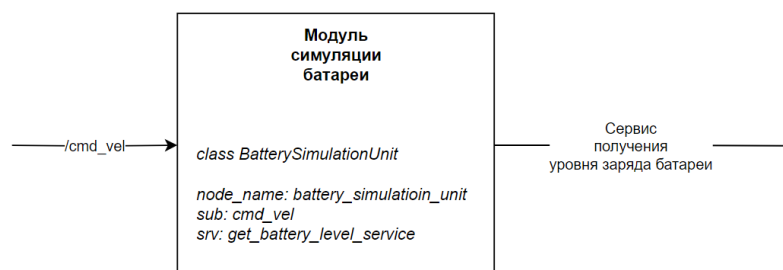


Рисунок 25 — Модуль симуляции батареи

Для тестирования поведения системы при недостаточном уровне заряда был реализован модуль симуляции батареи робота.

Модуль принимает информацию о текущей скорости робота и в соответствии

с заданными параметрами рассчитывает уровень заряда.

Параметры:

- расход батареи при простое;
- коэффициент расхода батареи при движении - линейно зависит от скорости робота;
- стартовый уровень заряда - в процентах.

В модуле также реализован интерфейс по которому другие модули системы могут узнать текущий уровень заряда *get_battery_level_service*.

Модуль симуляции неисправностей

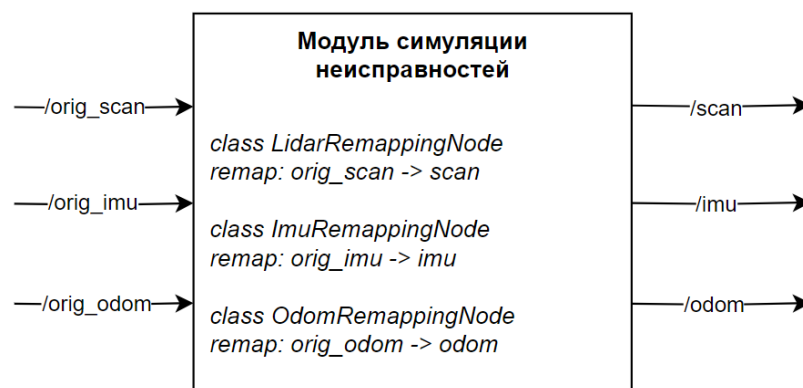


Рисунок 26 — Модуль симуляции неисправностей

Для тестирования поведения системы при неисправностях датчиков LIDAR и IMU, а также одометрии был реализован модуль симуляции неисправностей.

Все данные с этих компонент проходят через этот модуль, для того чтобы имелась возможность приостановить потоки данных. Это реализовано в виде перенаправления тем (topic remappings).

В итоге, приостановка одного из узлов данного модуля симулирует неисправность соответствующей компоненты.

Модуль пользовательских интерфейсов системы

Так как основной способ передачи данных между модулями являются сервисы, то для их корректной работы потребовалось создать соответствующие им типы данных - интерфейсы.

Имя сервиса	Тип запроса	Тип ответа
GetBatteryLevel.srv	std_msgs/Empty empty_request	float64 battery_level float64 idle_consumption float64 drive_consumption
GetGoal.srv	std_msgs/Empty empty_request	geometry_msgs/Pose goal_pose
GetGoalDistance.srv	std_msgs/Empty empty_request	float64 goal_distance
GetLastMap.srv	std_msgs/Empty empty_request	nav_msgs/OccupancyGrid map
GetPose.srv	std_msgs/Empty empty_request	geometry_msgs/Pose pose
PubResCmdVel.srv	geometry_msgs/Twist cmd_vel	bool success
SendMap.srv	nav_msgs/OccupancyGrid map	std_msgs/Empty empty_response

Таблица 4 — Описание реализованных интерфейсов для системы контроля

3.6 Система учета динамических объектов

3.6.1 Принципы обхода препятствий в Nav2

В системе Nav2 планирование траектории движения разделено на 2 уровня:

1. Сервер планирования (глобальный планировщик) - строит полный оптимальный путь от начального положения робота до цели навигации. Он принимает на вход статическую карту местности (либо заранее созданную, либо динамически обновляемую с помощью SLAM), а также может принимать некоторые другие слои карты (слой препятствий, слой инфляции, слой вокселей). Все слои накладываются друг на друга и на выходе дают объект класса *2DCostmap*. В итоге получается карта стоимости, представляющая собой сетчатую карту, в каждой ячейке которой записана стоимость нахождения робота в ней. Сервер планирования работает с этим результирующим объектом (глобальная карта стоимости).
2. Сервер управления (локальный планировщик) - вычисляет сигнал управления для слежения за траекторией построенной глобальным планировщиком. По аналогии с сервером планирования, у сервера управления также имеется в наличии локальная карта стоимости. Однако она имеет меньший размер, иную конфигурацию слоев и большую частоту обновления.

Соответственно отсюда мы видим, что обработка новых, заранее не отмеченных на статической карте объектов (в том числе динамических) происходит на уровне локального планировщика. Следовательно, если мы хотим как-то учитывать динамические препятствия, то необходимо модифицировать работу локального планировщика.

Работа основных серверов системы Nav2 (сервер навигации, сервер планирования, сервер управления, сервер восстановления) основывается на использовании плагинов: как уже существующих и предоставляемых разработчиками Nav2, так и пользовательских (для введения новых особенностей).

Для вычисления траектории сервер планирования по умолчанию использует плагин *NavfnPlanner*. Данный алгоритм планирования основан на применении алгоритма Дейкстры. Модифицировать работу глобального планировщика нет нужды, так как обработка динамических объектов должна происходить на уровне сервера управления.

Сервер управления, в свою очередь, может использовать для вычисления траектории в динамическом окружении два плагина (предоставляемых Nav2):

1. DWB контроллер (*nav2_dwb_controller* [24])
2. ТЕВ контроллер (*teb_local_planner* [25])

DWB контроллер - это плагин, реализующий подход динамического окна. Он учитывает все скорости, которые робот может достичь за промежуток времени между двумя последовательными сигналами управления. Скорости, которые могут привести к столкновениям, исключаются. Среди оставшихся скоростей выбирается наиболее перспективные сигналы управления с помощью функции затрат, представленной в виде карты стоимости (локальной), которая определяет стоимость прохождения через каждую ячейку карты [26].

Контроллер ТЕВ - это реализация подхода эластичной ленты. Он использует полный, свободный от препятствий путь от начальной точки до точки цели навигации. Этот путь разбивается на участки, которые затем представляются под воздействием вновь обнаруженных препятствий с помощью модели сил. Сила, действующая между участками, удерживает их вместе, а сила отталкивания толкает путь в сторону от препятствий. ТЕВ расширяет данный метод, добавляя дополнительную информацию о времени участкам

пути. Используя эту информацию, функция стоимости может учитывать как время движения, так и расстояние до препятствий [27].

3.6.2 Описание работы предлагаемой системы

Контроллер DWB, используемый по умолчанию сервером управления для построения траектории на основе локального представления (*local_costmap*), обращается с новыми (заранее не отмеченными препятствиями на статической карте) статическими и динамическими объектами одинаково.

В простейшем случае - данные с 2D LIDAR обрабатываются и отмечаются в слое *obstacle_layer*, далее накладывается *inflation_layer* для добавления “запаса безопасности” - экспоненциально убывающая функция стоимости от расстояния соседних клеток до препятствия.

В процессе движения динамического препятствия карта обновляется в соответствии с указанными параметрами.

В таком случае локальный планировщик не имеет информации о скорости и направлении движения объекта, а значит он не может спрогнозировать ситуацию, когда динамический объект будет пересекать построенную траекторию движения робота. При пересечении объектом построенной траектории движения планировщиком будут вызываться попытки перестроить траекторию, однако из-за отсутствия информации о величине и направлении скорости препятствия, будут происходить случаи, когда объект будет пересекать минимально допустимое расстояние до робота. Это вызовет поведение восстановления *wait* сервером восстановления и мобильный робот будет ждать до тех пор, пока динамический объект не покинет зону близости.

Соответственно для уменьшения таких ситуаций было предложено следующее решение, основанное на работе [12]. Предлагаемая система учета динамических объектов состоит из следующих модулей:

- модуль обнаружения динамических объектов;
- модуль слежения за обнаруженными объектами и оценки их скоростей;
- модуль отображения информации об объектах на карте стоимости.

Итого, данная система реализует учет динамических объектов в среде с помощью их отслеживания и представления информации о скорости движения в качестве плагина для добавления слоя в локальную карту стоимости.

3.6.3 Архитектура системы

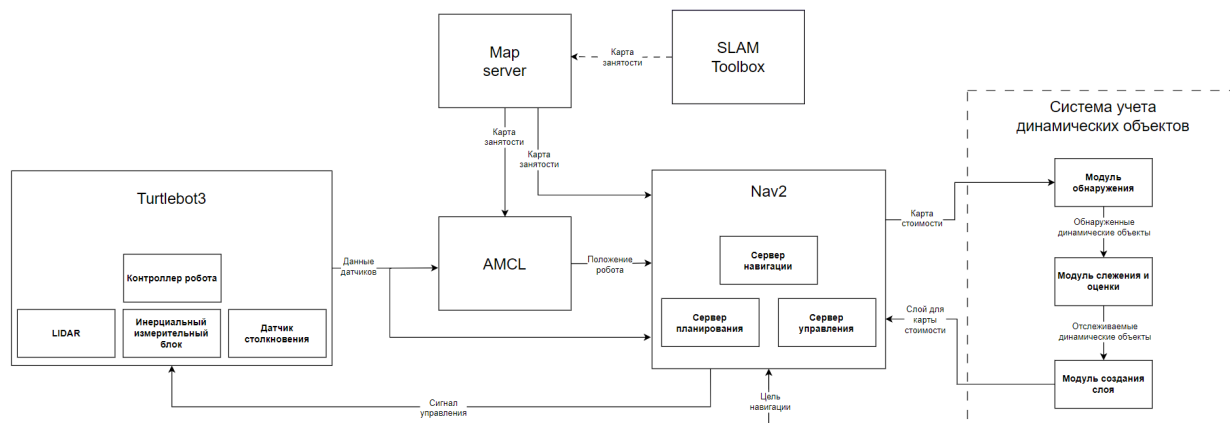


Рисунок 27 — Архитектура системы учета динамических объектов

На Рисунке 27 представлена архитектура системы учета динамических препятствий. Она состоит из трех модулей: модуль обнаружения динамических препятствий, модуль слежения за препятствиями и оценки их скорости и модуль добавления динамических объектов на новый слой для локальной карты стоимости.

3.6.4 Модуль обнаружения

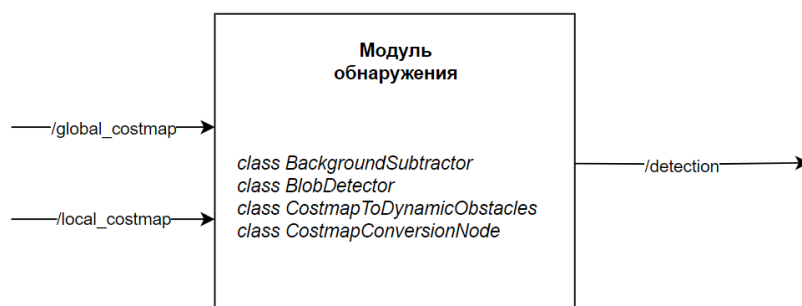


Рисунок 28 — Модуль обнаружения динамических объектов

Объект класса *CostmapConversionNode* является главным и управляет ходом процесса выделения динамических объектов из карты стоимости.

При инициализации происходит создание подписчика (subscriber) на темы */global_costmap* и */local_costmap* и издателя (publisher) темы */detection*. В тему */detection* с определенным интервалом времени будет передаваться массив *ObstacleArray.msg* из детектированных динамических объектов (имеющих пользовательский тип данных *Obstacle.msg*).

Процесс обнаружения динамических объектов также будет выполняться с частотой публикации данных в тему */detection*.

Функционал преобразования карты местности в массив динамических объектов определен в классе *CostmapToDynamicObstacles*. Этот процесс состоит из следующих шагов:

1. *CostmapToDynamicObstacle::setCostmap2D()* - преобразование карты стоимости из типа *nav2_costmap_2d::Costmap2D* в тип данных *cv::Mat*, для того чтобы можно было применять методы библиотеки OpenCV
2. *BackgroundSubtractor::apply()* - применение фильтров, в результате динамические объекты будут отражены в бинарной матрице *fg_mask*
3. *BlobDetector::detect()* - применение кластеризации и вычисление центров динамических объектов. Реализованный класс *BlobDetector* наследован от класса *SimpleBlobDetector* библиотеки OpenCV [28]
4. Запись всех обнаруженных объектов в *ObstacleArray.msg*

На Рисунке 29 представлен результаты работы данного модуля в ситуации с одним динамическим объектом, движущимся по короткой стороне карты.

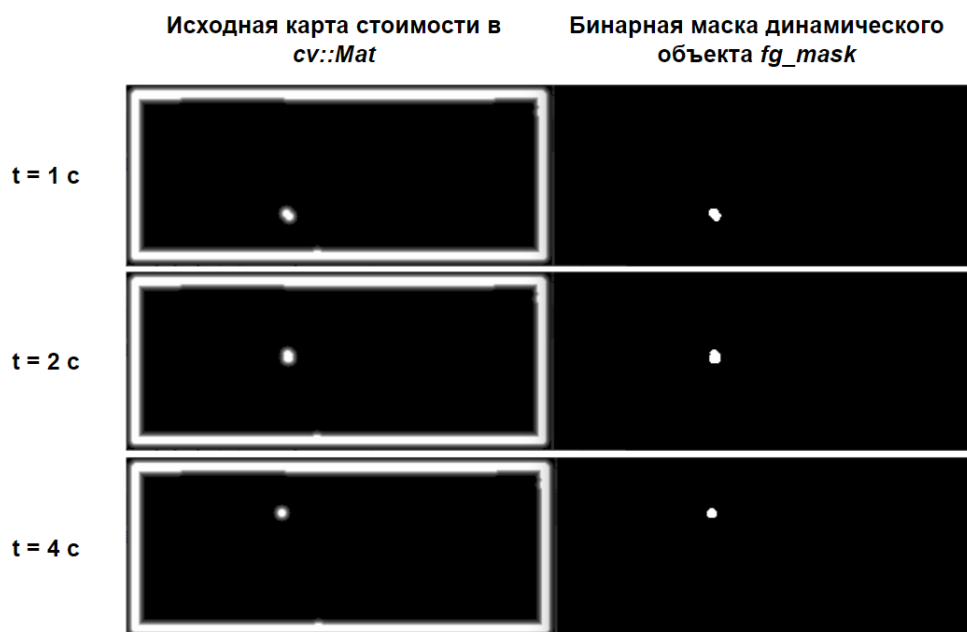


Рисунок 29 — Результат работы модуля обнаружения динамических объектов

Как видно, выделение динамических объектов было проведено успешно. Удалось отфильтровать статические объекты и выделить только динамические.

3.6.5 Модуль слежения и оценки

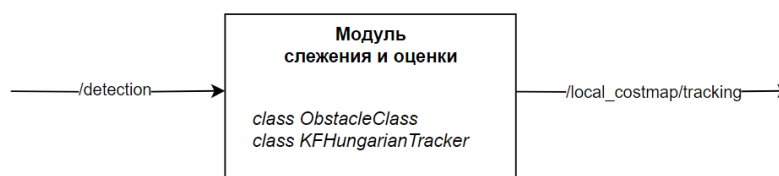


Рисунок 30 — Модуль слежения и оценки динамических объектов

Класс *ObstacleClass* представляет собой один динамический объект, его поля хранят необходимую о динамическом объекте информацию:

- положение;
- скорость;
- фильтр Калмана типа *cv2.KalmanFiler*;
- количество фреймов (снимков), в которых данный объект отсутствовал (при достижении порогового значения - объект удаляется).

Отметим, что *cv2.KalmanFiler* - это класс библиотеки OpenCV [29], который реализует шаги предсказания и обновления фильтра Калмана через методы *cv2.KalmanFilter.predict()* и *cv2.KalmanFilter.update()*. При создании объекта этого класса матрицы H, F, R, P, Q инициализируются, как указано в подразделе 1.3.2.

Класс *KFHungarianTracker* реализует процесс слежения за динамическими объектами. Он подписывается на тему */detection* (с которого поступают данные об обнаруженных динамических объектах), и публикует данные об отслеживаемых объектах в тему */local_costmap/tracking*.

При каждом новом поступлении данных выполняются следующие действия:

1. Вычисляется промежуток времени между сообщениями для того чтобы передать его в метод *predict()* класса *ObstacleClass*, тем самым предсказав положение и скорость детектированных объектов
2. Далее решается задача сопоставления объектов (assignment problem) с помощью венгерского алгоритма. Для этого используется функция *linear_sum_assignment()* библиотеки SciPy [30]
3. Имея информацию о парах объектов (объект в момент t и тот же объект в момент $t + 1$), мы обновляем фильтр Калмана для соответствующих объектов (метод *correct()* класса *ObstacleClass*)
4. В конце массив отслеживаемых объектов публикуется в тему */local_costmap/tracking*

На Рисунке 31 представлен пример выделения двух динамических объектов и оценки их скорости.

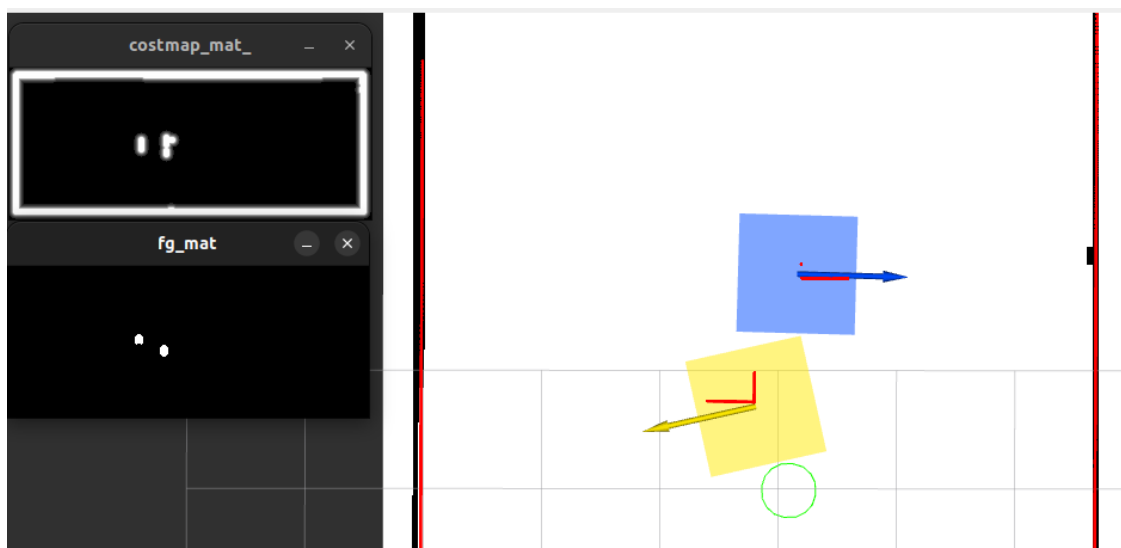


Рисунок 31 — Пример работы модуля слежения и оценки динамических объектов

Два динамических объекта были обнаружены и производится слежение за ними, а также оценка их скоростей. Динамические объекты сегментированы в виде прямоугольников, а также показана оценка величины и направления скорости (в виде векторов, идущих из центров прямоугольников).

3.6.6 Модуль добавления слоя

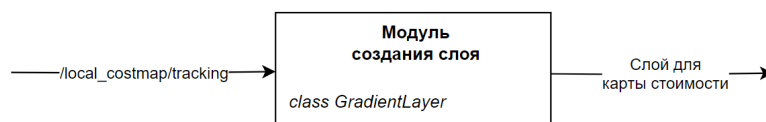


Рисунок 32 — Модуль добавления слоя

Данный модуль создавался по шаблону предложенному разработчиками Nav2 для создания пользовательского плагина для реализации слоя для карты стоимости [31].

Основной класс *GradientLayer* вычисляет и добавляет область стоимости около каждого динамического объекта через метод *updateCosts()*, который выполняет следующие действия:

1. Получает текущие отслеживаемые динамические объекты из темы */local_costmap/tracking*

2. Меняет систему координат относительно которой записаны динамические объекты (*map* -> *odom*). Так как *local_costmap* использует систему координат */odom*, а мы как раз хотим применить наш слой к *local_costmap*
3. Добавление к каждому динамическому объекту области стоимости в соответствующей формы (описано в подразделе 1.3.3), в зависимости от направления и величины скорости объекта. Данный функционал реализован в функции *markDynamicObstacle()*

На Рисунке 33 представлена локальная карта местности с применением данного *GradientLayer*:

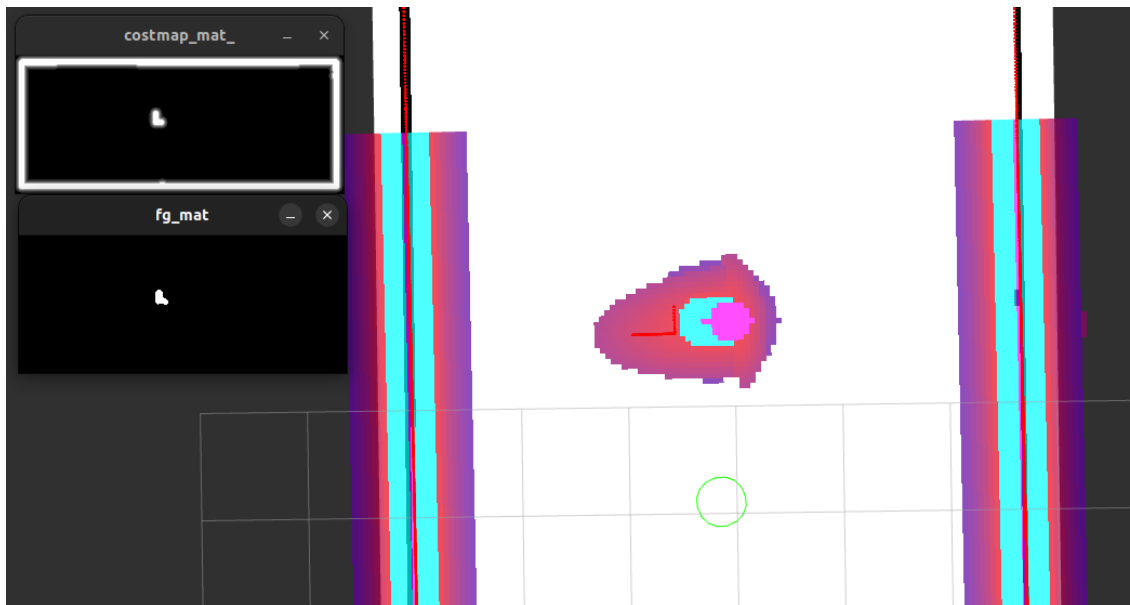


Рисунок 33 — Пример работы модуля добавления слоя

Опишем работу данного модуля на основе приведенного примера. Динамический объект совершает движение налево. Система динамического учета обнаруживает, отслеживает и оценивает величину и направление скорости динамического объекта. В соответствии с этим на локальную карту местности добавилась гауссовская область затрат, распределенная таким образом, что затраты по направлению движения больше, чем за объектом. Таким образом локальный планировщик будет учитывать информацию о скорости динамического препятствия.

3.6.7 Модуль пользовательских интерфейсов системы

Для представления динамических объектов был создан класс *DynamicObstacle*. Для передачи объектов между модулями необходимо было реализовать пользовательский интерфейс.

Он реализован в качестве двух типов сообщений:

1. *Obstacle.msg*

Obstacle.msg
<pre># Заголовок для временной информации std_msgs/Header header # Площадь объекта (obstacle footprint) geometry_msgs/Polygon polygon # Идентификатор объекта unique_identifier_msgs/UUID id # Ориентация центра объекта geometry_msgs/Quaternion orientation # Информация о границе объекта geometry_msgs/Vector3 size # Скорость объекта geometry_msgs/Vector3 velocity # Положение центра объекта geometry_msgs/Point position # Уверенность в существовании объекта int32 score</pre>

Рисунок 34 — *Obstacle.msg*

2. *ObstacleArray.msg*

ObstacleArray.msg
<pre># Заголовок для временной информации std_msgs/Header header # Массив для хранения объектов dms_interfaces/Obstacle[] obstacles</pre>

Рисунок 35 — *ObstacleArray.msg*

Оба типа сообщений находятся в общем пакете *dms_interfaces*, вместе с интерфейсами для системы контроля.

3.7 Общая архитектура системы принятия решений

Итоговая архитектура разработанной системы:

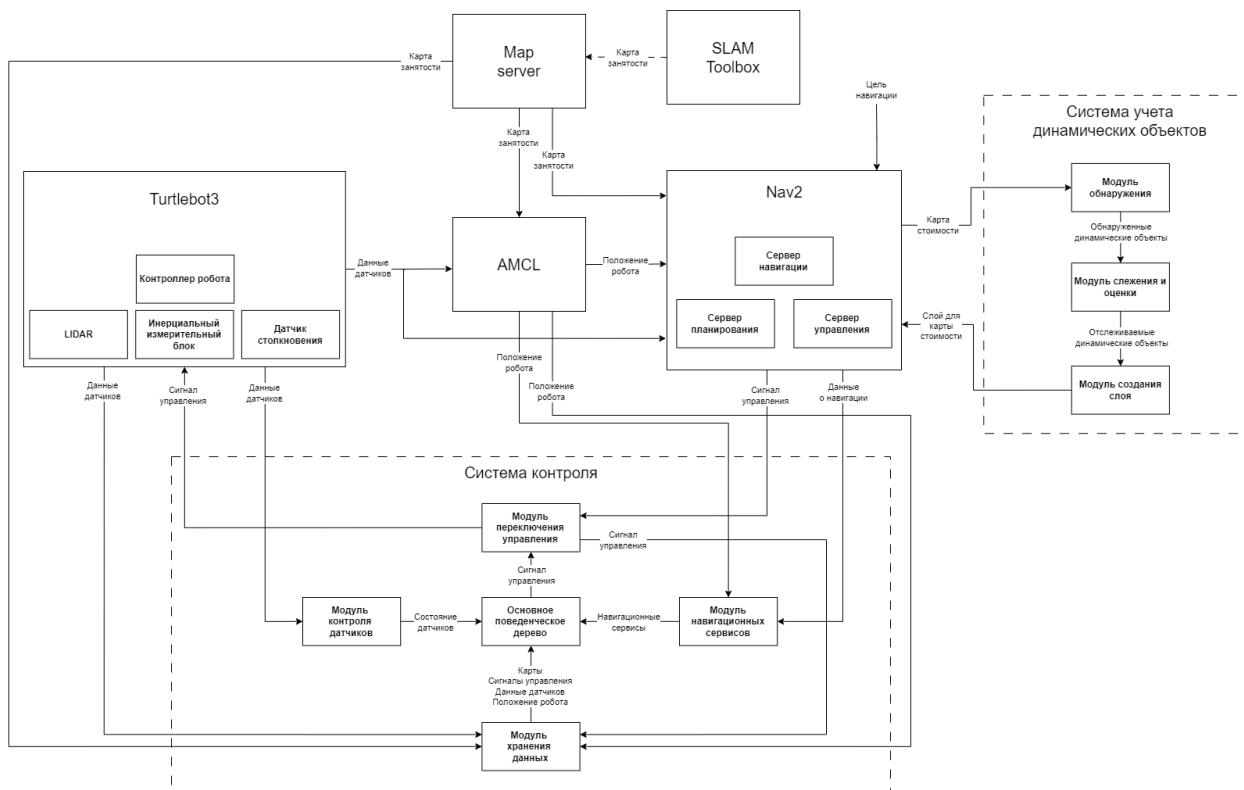


Рисунок 36 — Архитектура системы принятия решений

4 МОДЕЛИРОВАНИЕ И ОЦЕНИВАНИЕ

Для оценки эффективности разработанной системы проведем эксперименты при различных сценариях с двумя системами: стандартная система для Nav2 и предложенная система с подсистемой контроля и подсистемой учета динамических объектов.

Эксперименты разделены на 2 части: первая часть нацелена на проверку поведений при различных неполадках (проверка системы контроля), а вторая будет проводиться в условиях динамического окружения (проверка системы учета динамических объектов).

4.1 Модификация модели мобильного робота

Как уже отмечалось в главе 2.3 в качестве мобильного робота был выбран робот ROBOTIS Turtlebot3.

В рамках данной работы в конфигурации робота были внесены следующие изменения:

1. Убран модель камеры
2. Добавлен датчик, реагирующий на столкновения (bumper sensor). Он имеет размеры $10 \times 265 \times 25$ мм и прикреплен к передней части робота

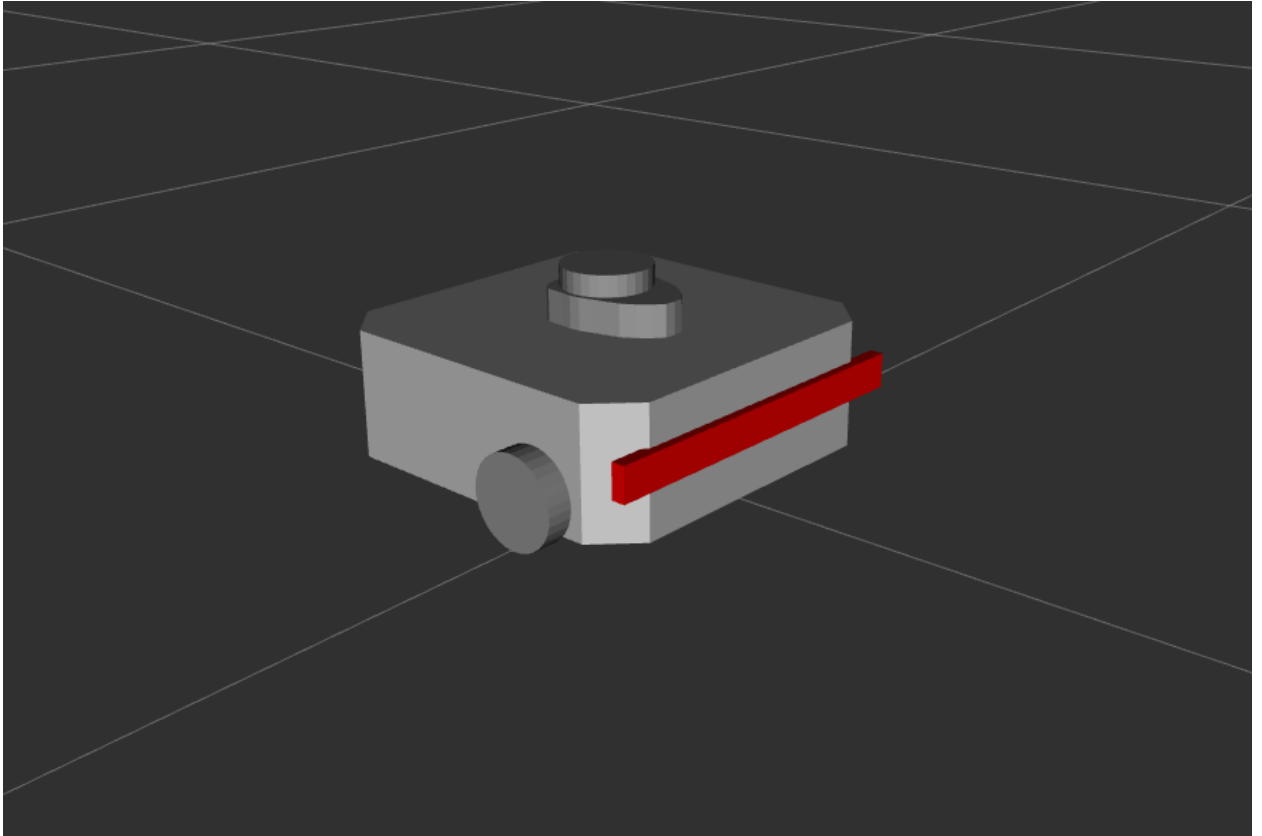


Рисунок 37 — Модель мобильного робота в Rviz (красным цветом отмечен датчик столкновения)

4.2 Симуляционные миры

Эксперименты проводятся в двух окружениях:

1. *TurtleBot3 House* - статическое окружение, описывающее модель дома. Эта карта поставляется вместе с пакетом *turtlebot3*. Эксперименты на данной карте будут проводиться для тестирования системы контроля



Рисунок 38 — Симуляционный мир *TurtleBot3 House*

2. *Dynamic World* - динамическое окружение, описывающее замкнутое пространство, окруженное стенами высотой 1.5 м и периметром 5×10 м. Внутри пространства находятся 3 пешехода (actors), которые имеют траектории движения параллельно короткой стороне прямоугольника. При тестировании различных сценариев значения скоростей пешеходов будет изменяться, также начальная точка движения пешеходов будет различна при каждом сценарии. Высота пешеходов равна 2 м, а максимальные размеры на плоскости XY не превышают 30×30 см. Эксперименты на данной карте будут проводиться для тестирования системы учета динамических объектов

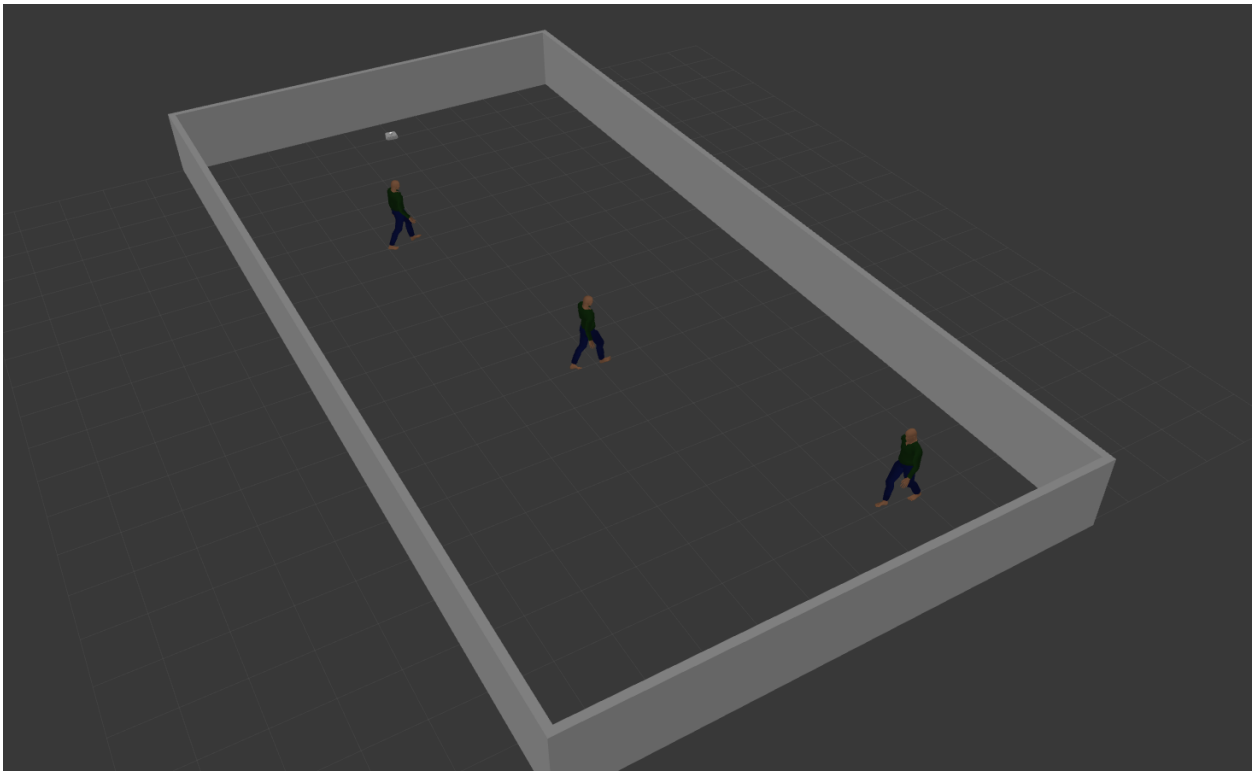


Рисунок 39 — Симуляционный мир *Dynamic World*

4.3 Картографирование местности

Для построения статической карты местности и дальнейшей передачи на вход системе навигации использовался SLAM. Таким образом, построенная заранее карта будет использоваться для всех последующих на ней тестов. Это убирает необходимость ”изучения” мобильным роботом окружения при каждом новом запуске.

В частности, в качестве метода SLAM по умолчанию использовался картограф - проект Google с открытым исходным кодом, реализованный в пакете *turtlebot3_cartographer*. Для выполнения SLAM для обоих вышеупомянутых окружений были созданы две статические карты занятости.

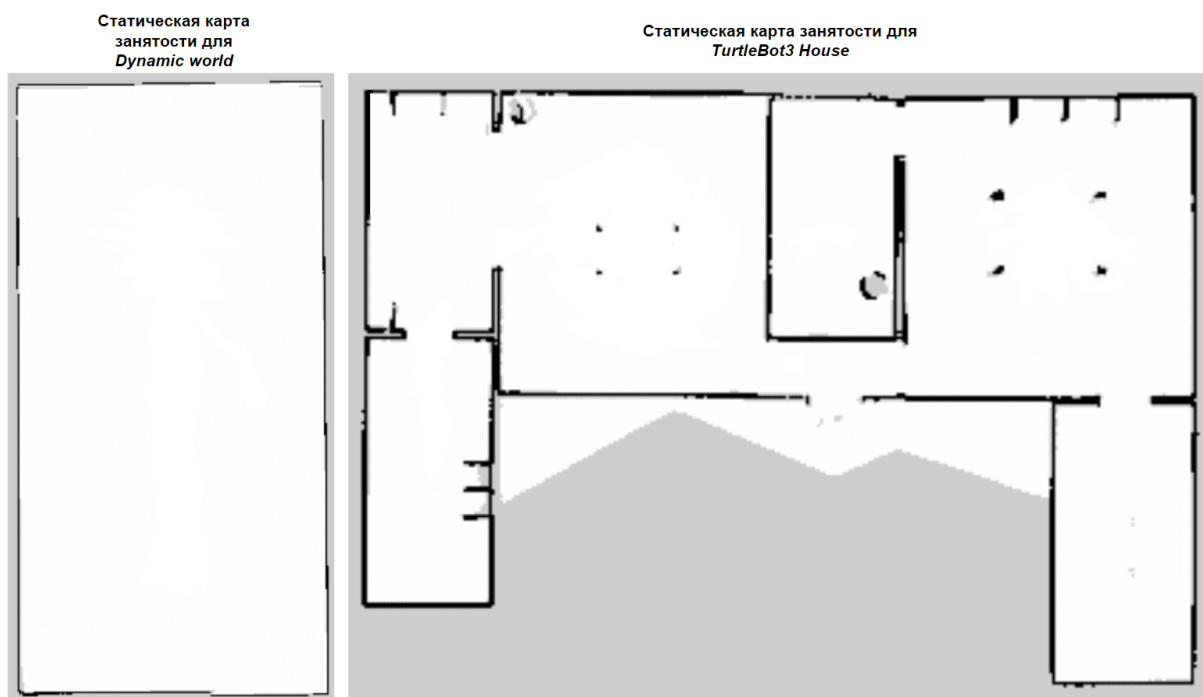


Рисунок 40 — Статические карты занятости для используемых окружений (масштабы карт изменены)

4.4 Тестирование

Целью тестирования является получение оценки работоспособности и эффективности разработанной системы и её сравнение с существующими решениями.

Предлагаемые ниже тесты предназначены для сравнения поведения робота в условиях внезапных неполадок с различными компонентами системы и в условиях динамического окружения.

В тестировании участвуют две системы:

1. Стандартная система Nav2. Архитектура представлена на Рисунке 11. Для удобства дадим ей обозначение *STD*
2. Система Nav2 с внедрением разработанной системы принятия решений. Архитектура представлена на Рисунке 36. Обозначим её *DMS*

4.4.1 Тестирование системы контроля

Сценарии

В Таблицах 5 и 6 представлены описания сценариев тестирования вместе с критериями успешной отработки данного сценария. Для того чтобы тест был признан успешным, должны быть выполнены все критерии успешной отработки для данного сценария.

В тестах с LIDAR, IMU и одометрией драйвер датчика отключался при достижении желаемой скорости, указанной в описании сценария. Каждый из данных компонент был протестирован несколько раз в трех различных ситуациях навигации.

В сценарии *collision_1* столкновение регистрируется датчиком столкновения. Данный сценарий проверяет способность системы контроля проверять состояние робота после столкновения, обновлять карту местности и возобновлять навигацию.

collision_2 тестирует то же поведение, однако теперь робот находится в процессе навигации. На пути робота возникает недетектируемое препятствие, оно находится слишком низко, чтобы его мог обнаружить датчик LIDAR. Робот должен способен восстановиться после столкновения и продолжить движение к цели по обновленному пути.

Сценарий *battery_1* проверяет поведение робота при низком заряде батареи. Симулируемая батарея разряжается до низкого уровня, и задается новая цель навигации. Тест завершается неудачей, если во время навигации заряд батареи опускается ниже нуля процентов.

gl_pl_1 - сценарий, в котором состояние жизненного цикла глобального планировщика переходит в "неактивное" и не может больше выполнять действия по планированию при срабатывании. В этом случае модуль основного поведенческого дерева должен обработать данное событие и восстановить функционирование навигации.

Результаты

Результаты, приведенные в Таблице 7, свидетельствуют о существенном увеличении процента успешного выполнения тестовых сценариев разработанной системы. Система контроля достаточно надежна в предложенных

Тестируемый компонент	Имя теста	Описание сценария	Критерии успешной отработки
LIDAR	<i>lidar_1</i>	1. Робот стоит 2. Неполадка с LIDAR	1. Перезапуск LIDAR
	<i>lidar_2</i>	1. Робот движется с линейной скоростью 0.25 м/с прямо к цели 2. Неполадка с LIDAR	1. Перезапуск LIDAR 2. Достижение цели
	<i>lidar_3</i>	1. Робот движется с линейной скоростью 0.25 м/с и угловой скоростью 0.5 рад/с 2. Неполадка с LIDAR	1. Перезапуск LIDAR 2. Достижение цели
IMU	<i>imu_1</i>	1. Робот стоит 2. Неполадка с LIDAR	1. Перезапуск IMU
	<i>imu_2</i>	1. Робот движется с линейной скоростью 0.25 м/с прямо к цели 2. Неполадка с IMU	1. Перезапуск IMU 2. Достижение цели
	<i>imu_3</i>	1. Робот движется с линейной скоростью 0.25 м/с и угловой скоростью 0.5 рад/с 2. Неполадка с IMU	1. Перезапуск IMU 2. Достижение цели
Одометрия	<i>odom_1</i>	1. Робот стоит 2. Неполадка с одометрией	1. Перезапуск одометрии
	<i>odom_2</i>	1. Робот движется с линейной скоростью 0.25 м/с прямо к цели 2. Неполадка с одометрией	1. Перезапуск одометрии 2. Достижение цели
	<i>odom_3</i>	1. Робот движется с линейной скоростью 0.25 м/с и угловой скоростью 0.5 рад/с 2. Неполадка с одометрией	1. Перезапуск одометрии 2. Достижение цели

Таблица 5 — Таблица сценариев для тестирования датчиков LIDAR и IMU, а также одометрии

Тестируемый компонент	Имя теста	Описание сценария	Критерии успешной отработки
Столкновение	collision_1	1. Робот стоит 2. Произошло столкновение	1. Отмена навигации 2. Ожидание выхода из состояния столкновения 3. Добавление объекта столкновения на карту 4. Проверка всех компонент системы
	collisioin_2	1. Робот движется с линейной скоростью 0.25 м/с 2. Произошло столкновение с недетектируемым объектом	1. Отмена навигации 2. Ожидание выхода из состояния столкновения 3. Добавление объекта столкновения на карту 4. Проверка всех компонент системы 5. Продолжение навигации
Батарея	battery_1	1. Уровень заряда батареи недостаточен для достижения цели	1. Робот не будет двигаться к цели, которая находится за пределами
Глобальный планировщик	gl_pl_1	1. Робот стоит 2. Путь до цели не может быть вычислен	1. Перезапустить глобальный планировщик 2. Перестроить путь до цели 3. Достигнуть цель

Таблица 6 — Таблица сценариев для тестирования глобального планировщика, столкновений и недостаточного уровня заряда батареи

сценариях и позволяет системе восстанавливаться после неожиданной неисправности какой-то компоненты.

Сценарий недостаточного уровня заряда батареи был успешным каждый раз, когда проводилось тестирование.

Однако поведение планирования пути для перезапуска планировщика успешно выполнялось не всегда (90%).

Поведение при столкновениях работает достаточно хорошо и одним из важным компонентом этого является обновление карты местности для добавления объекта столкновения на нее. Тем не менее, в 30-40% случаев роботу не хватает автономности продолжить выполнение задач после столкновения.

В результате проведенного тестирования можно выделить следующие функциональные особенности, которые превнесла реализованная система контроля в уже существующую архитектуру:

1. Система обнаруживает сбой компоненты. В таких ситуациях система может перезапускать датчики и снижать скорость в течение некоторого времени, когда компонент предоставляет ограниченную информацию
2. Система обнаруживает аварийные ситуации. Система может обнаружить, когда продолжение движения по рассчитанному пути уже небезопасно (отказ датчиков, нехватка заряда батареи, блокирование объектом)
3. Система может инициировать аварийную остановку. Система может отменить все команды и остановиться в случае обнаружения аварийной ситуации
4. Система может отменить команды с Nav2. Команды системы всегда могут отменить команды, поступающие от Nav2
5. Система предотвращает случаи, когда во время движения к цели у робота разрядится аккумулятор
6. (Выполняется не всегда) Система может восстанавливаться после столкновений

Также помимо приведенных выше функциональных преимуществ, разработанная система обладает некоторыми архитектурными особенностями. Отметим наиболее важные из них:

1. Архитектурный дизайн системы контроля устраняет единую точку отказа системы. Даже если система Nav2 станет недоступна во время движения, робот сможет корректно обработать данный сценарий и безопасно приостановить навигацию
2. Производительность системы при проверке условий основного поведенческого дерева достаточна для успешного выполнения задач навигации в динамическом окружении. Этого удалось достичь за счет применения принципа модульности при реализации (вся система работает в нескольких потоках)

Имя теста	Количество симуляций	Процент успеха, %	
		STD	DMS
<i>lidar_1</i>	5	0	100
<i>lidar_2</i>	5	0	100
<i>lidar_3</i>	5	0	100
<i>imu_1</i>	5	0	100
<i>imu_2</i>	5	0	100
<i>imu_3</i>	5	0	100
<i>odom_1</i>	5	0	100
<i>odom_2</i>	5	0	100
<i>odom_3</i>	5	0	100
<i>collision_1</i>	10	0	80
<i>collision_2</i>	10	0	60
<i>battery_1</i>	10	0	100
<i>gl_pl_1</i>	10	0	90

Таблица 7 — Результаты тестирования системы контроля

4.4.2 Тестирование системы учета динамических объектов

Для тестирования работы системы в условиях динамического окружения будем использовать мир *Dynamic World* с различными конфигурациями движущихся объектов.

Стоит отметить, что каждый динамический объект в окружении (пешеход) симуляции был создан с использованием плагина *actor_collisions* [32]. Добавление этого плагина позволяет пешеходу иметь свойства столкновения (*<collision>*), что позволяет динамическим пешеходам или препятствиям быть охваченными лидаром в сцене моделирования Gazebo. То есть использование данного плагина увеличивает степень реалистичности моделирования.

Сценарии

Для тестирования системы в динамическом окружении были придуманы 3 сценария, отличающихся только значениями скоростей пешеходов, они представлены в Таблице 8.

Все сценарии происходят в окружении *Dynamic World*. Роботу задается цель навигации, расположенная на противоположной стороне прямоугольной зоны. Общее расстояние пути равно 8 метрам. Упрощенная схема тестирования изображена на Рисунке 41.

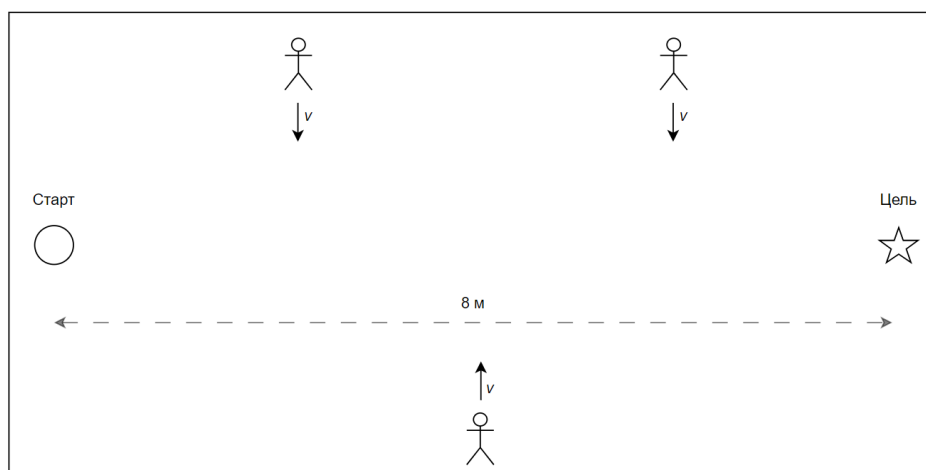


Рисунок 41 — Упрощенная схема тестирования в условиях динамических объектов

Для оценивания рассчитываются три величины:

1. Успешная навигация - процент от общего числа тестов, при котором робот достигает цель навигации
2. Поведение ожидания - это процент от общего числа тестов, во время которых срабатывало поведение ожидания (*wait*), но при этом цель навигации была достигнута. Поведение ожидания вызывается сервером восстановления, когда препятствия внезапно оказываются слишком близко и сервер управления не может найти приемлемый путь в течение некоторого времени
3. Столкновение - процент от общего числа тестов, в которых не была достигнута цель навигации из-за столкновения робота с пешеходом

Номер теста	Скорость пешеходов, м/с
1	0.5
2	0.75
3	1.0

Таблица 8 — Таблица с данными о скоростях пешеходов при тестировании

Результаты

Для каждого из трех сценариев тестирование проводилось 25 раз. Результаты каждого эксперимента для двух систем представлены в Таблице 9.

Номер теста	Успешная навигация, %		Поведение ожидания, %		Столкновение, %	
	STD	DMS	STD	DMS	STD	DMS
1	96	100	0	8	4	0
2	64	72	0	28	36	28
3	28	48	0	44	72	52

Таблица 9 — Результаты тестирования системы в динамическом окружении

Анализируя результаты, можно увидеть, что разработанная система учета динамических объектов показала более высокий процент успешной навигации во всех 3-х сценариях. Наиболее существенное улучшение происходит на третьем тесте (скорость пешеходов 1 м/с).

Также разработанная система меньше попадает в столкновения с объектами. Это напрямую связано с увеличением процента поведения ожидания. Ведь с добавлением слоя для локального планировщика, в котором отмечена зона вокруг динамических объектов в форме гауссовского распределения - это позволяет учитывать величину и направление движения объектов мобильным роботом. А значит что сервер восстановления срабатывает своевременно и вызывает поведение ожидания, а значит удаётся предотвратить столкновение.

5 ЗАКЛЮЧЕНИЕ

В данной работе продемонстрирована реализация системы принятия решений для задачи автономной навигации в условиях динамического окружения. Целью разработки системы являлась увеличение автономности, надежности и безопасности. Этого удалось достичь за счет внедрения системы контроля и системы учета динамических объектов.

Реализация системы контроля существенно снизила количество сценариев, в которых требуется участие человека-оператора, что говорит о более устойчивой и автономной работе. Реализованное поведение, реагирующее на сбои различных компонент системы навигации, ситуации столкновения а также контролирующее заряд батареи, улучшило качество навигации. В основе данной системы лежит поведенческое дерево, которое позволяет безопасно выполнять более сложные модели поведения, поскольку возможные сценарии, мешающие правильному представлению среды, проверялись и обрабатывались до того, как сложные модели поведения выполнялись на основе этого представления.

Система учета динамических объектов обеспечивает более безопасную навигацию в условиях динамических препятствий. Она позволяет планировщику траектории учитывать информацию о скорости динамических объектов, таким образом система навигации успешно корректирует траекторию, позволяя роботу избежать столкновения.

Кроме того, архитектура данной системы построена по принципам модульности и может быть усовершенствована, не сталкиваясь с проблемами или ограничениями на более поздних этапах разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Murphy R.R.* Introduction to AI robotics. — MIT press, 2019.
2. *Юревич Е.* Сенсорные системы в робототехнике: учеб. пособие. — Издательство Политехнического ун-та, 2013.
3. *Barfoot T.D.* State estimation for robotics. — Cambridge University Press, 2024.
4. *LaValle S.M.* Planning algorithms. — Cambridge university press, 2006.
5. *Поляк Б.Т. Хлебников М.В. Р.Л.* Математическая теория автоматического управления: учебное пособие. — М.: ЛЕНАНД, 2019.
6. *Wagner F., Schmuki R., Wagner T., Wolstenholme P.* Modeling software with finite state machines: a practical approach. — Auerbach Publications, 2006.
7. *Flórez-Puga G., Gomez-Martin M.A., Gomez-Martin P.P., Diaz-Agudo B., Gonzalez-Calero P.A.* Query-enabled behavior trees // IEEE Transactions on Computational Intelligence and AI in Games. — 2009. — Vol. 1, no. 4. — P. 298–308.
8. *Colledanchise M., Ögren P.* Behavior trees in robotics and AI: An introduction. — CRC Press, 2018.
9. *Van Otterlo M., Wiering M.* Reinforcement learning and markov decision processes // Reinforcement learning: State-of-the-art. — Springer, 2012. — P. 3–42.
10. *Krishnamurthy V.* Partially observed Markov decision processes. — Cambridge university press, 2016.
11. *Laugier C., Chatila R.* Autonomous navigation in dynamic environments. Vol. 35. — Springer, 2007.
12. *Albers F., Rösmann C., Hoffmann F., Bertram T.* Online trajectory optimization and navigation in dynamic environments in ROS // Robot Operating System (ROS) The Complete Reference (Volume 3). — 2019. — P. 241–274.
13. Blob detection using OpenCV (Python, C++). — Электронный ресурс доступен по адресу <https://learnopencv.com/blob-detection-using-opencv-python-c/>.

14. *Kuhn H.W.* The Hungarian method for the assignment problem // Naval research logistics quarterly. — 1955. — Vol. 2, no. 1/2. — P. 83–97.
15. *Степанов О.* Фильтр Калмана: история и современность // Гироскопия и навигация. — 2010. — Т. 69, № 2. — С. 107–121.
16. *Saho K.* Kalman filter for moving object tracking: Performance analysis and filter design // Kalman Filters-Theory for Advanced Applications. — 2017. — P. 233–252.
17. *Yang F., Peters C.* Social-aware navigation in crowds with static and dynamic groups // 2019 11th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games). — IEEE. 2019. — P. 1–4.
18. *Macenski S., Foote T., Gerkey B., Lalancette C., Woodall W.* Robot Operating System 2: Design, architecture, and uses in the wild // Science robotics. — 2022. — Vol. 7, no. 66. — eabm6074.
19. *Macenski S., Martín F., White R., Clavero J.G.* The marathon 2: A navigation system // 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — IEEE. 2020. — P. 2718–2725.
20. ПО для реализации алгоритма SLAM. — Электронный ресурс доступен по адресу https://wiki.ros.org/slam_toolbox.
21. Описание задачи для реализации динамической навигации Nav2. — Электронный ресурс доступен по адресу <https://navigation.ros.org/2021summerOfCode/projects/dynamic.html>.
22. Исходный код пакетов для Turtlebot3. — Электронный ресурс доступен по адресу <https://github.com/ROBOTIS-GIT/turtlebot3/tree/humble-devel>.
23. Исходный код пакетов для Nav2. — Электронный ресурс доступен по адресу <https://github.com/open-navigation/navigation2/tree/humble>.
24. Исходный код для DWB controller. — Электронный ресурс доступен по адресу https://github.com/ros-navigation/navigation2/tree/main/nav2_dwb_controller.

25. Исходный код для TEB controller. — Электронный ресурс доступен по адресу
https://github.com/rst-tu-dortmund/teb_local_planner.
26. *Fox D., Burgard W., Thrun S.* The dynamic window approach to collision avoidance // IEEE Robotics & Automation Magazine. — 1997. — Vol. 4, no. 1. — P. 23–33.
27. *Magyar B., Tsiogkas N., Deray J., Pfeiffer S., Lane D.* Timed-elastic bands for manipulation motion planning // IEEE Robotics and Automation Letters. — 2019. — Vol. 4, no. 4. — P. 3513–3520.
28. Описание класса SimpleBlobDetector библиотеки OpenCV. — Электронный ресурс доступен по адресу
https://docs.opencv.org/4.x/d0/d7a/classcv_1_1SimpleBlobDetector.html.
29. Описание класса KalmanFilter библиотеки OpenCV. — Электронный ресурс доступен по адресу
https://docs.opencv.org/4.x/dd/d6a/classcv_1_1KalmanFilter.html.
30. Описание функции linear_sum_assignment библиотеки SciPy. — Электронный ресурс доступен по адресу
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html.
31. Шаблон для написания пользовательского плагина для реализации слоя для карты местности. — Электронный ресурс доступен по адресу
https://navigation.ros.org/plugin_tutorials/docs/writing_new_costmap2d_plugin.html.
32. Плагин добавления свойств столкновения для пешехода в Gazebo. — Электронный ресурс доступен по адресу
https://github.com/JiangweiNEU/actor_collisions.