МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»)

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

Разработка программного модуля для автономной разгрузки роботизированного мини-погрузчика

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301. 2025.406 ПЗ ВКР

Руководитель работы,
к.пед.н., доцент каф. ЭВМ
_____ Ю. Г. Плаксина
«___»_____2025 г.

Автор работы,
студент группы КЭ-406
_____ И. А. Зуев
«___»_____2025 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2025 г.

Челябинск-2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2025 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Зуеву Ивану Александровичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка программного модуля для автономной разгрузки роботизированного мини-погрузчика» утверждена приказом по университету от «21» апреля 2025 г. № 648-13/12**.**

2. **Срок сдачи студентом законченной работы:** «01» июня 2025 г.

3. **Исходные данные к работе:**

3.1. Стереокамера ZED 2i.

3.2. Бортовой компьютер на базе ноутбука Machenike L15:

3.2.1. Симулятор на Unity.

3.2.2. Операционная система Ubuntu 22.04.

3.2.3. Пакет ROS 2 Humble.

3.2.4. Пакет локализации robot_localization.

3.2.5. Модуль ArUco-детекции.

4. **Перечень подлежащих разработке вопросов:**

4.1. Анализ предметной области и технологий автономной локализации, навигации и управления.

4.2. Проектирование программного модуля.

4.3. Разработка и интеграция в ROS 2.

4.4. Симуляция и тестирование системы.

5. **Дата выдачи задания:** «02» декабря 2024 г.

Руководитель работы _____/*Ю. Г. Плаксина /*

Студент _____/*И. А. Зуев /*

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Анализ предметной области и технологий автономной локализации, навигации и управления	01.03.2025	
Проектирование программного модуля	01.04.2025	
Разработка и интеграция в ROS 2	01.05.2025	
Симуляция и тестирование системы	15.05.2025	
Компоновка текста работы и сдача на нормоконтроль	24.05.2025	
Подготовка презентации и доклада	30.05.2025	

Руководитель работы _____/ *Ю. Г. Плаксина* /

Студент _____/ *И. А. Зуев* /

Аннотация

И.А. Зуев. Разработка программного модуля для автономной разгрузки роботизированного мини-погрузчика. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 61 с., 22 ил., библиогр. список – 30 наим.

В данной выпускной квалификационной работе рассматривается проектирование и реализация программного модуля, обеспечивающего автономную разгрузку роботизированного мини-погрузчика.

В первой части работы проведён комплексный анализ предметной области. Описаны существующие промышленные решения компаний БЕЛАЗ, Volvo и Sandvik. Рассмотрены ключевые технологии, применяемые в автономных робототехнических системах: визуальную одометрию; лидарная локализация; инерциальные измерительные устройства; методы интеграции данных с использованием фильтра Калмана. Проведен сравнительный анализ архитектур навигации: конечных автоматов, обучения с подкреплением и деревьев поведения, в результате чего обоснован выбор последнего для реализации системы.

На основании полученных данных сформулированы функциональные и нефункциональные требования к системе. Программный модуль реализован с применением фреймворка ROS 2 и языка C++. Навигация выполнена с использованием стека Nav2, а поведение погрузчика организовано через архитектуру дерева поведения. Отдельные узлы, включая управление рабочими органами и визуальную локализацию, разработаны в виде специализированных ROS-компонентов.

Работа системы протестирована в виртуальной среде Unity. Были успешно реализованы и проверены ключевые сценарии: поиск разгрузочной зоны, движение к цели, управление ковшом, возврат в исходную точку.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современные тенденции в области автоматизации производства и механических процессов обусловлены стремлением к повышению эффективности, снижению затрат и минимизации участия человека в рутинных или тяжёлых операциях. Автоматизация позволяет значительно повысить точность выполнения задач, обеспечить стабильность работы независимо от внешних условий, таких как время суток или погодные факторы, а также увеличить общую производительность.

Одним из актуальных направлений автоматизации является внедрение автономных решений в сфере погрузочно-разгрузочных работ. Особенно это актуально для карьерных и строительных площадок, где традиционная модель с участием оператора требует высокой квалификации и значительных физических усилий. Использование автономных погрузчиков с возможностью автоматической разгрузки позволяет не только облегчить труд человека, но и оптимизировать весь процесс за счёт точного позиционирования и высокой повторяемости операций.

В рамках данного проекта разрабатывается программный модуль, обеспечивающий автономную разгрузку погрузчика с использованием Aruco-маркеров для локализации в пространстве.

Объектом исследования является процесс автоматизации разгрузочных операций мобильных автономных погрузчиков.

Предмет исследования – методы и алгоритмы автономной навигации и разгрузки с использованием визуальных маркеров.

Представленная работа направлена на практическое применение технологий автономной мобильной робототехники и может быть использована как основа для дальнейших исследований в области автоматизации производственных процессов.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ТЕХНОЛОГИЙ АВТОНОМНОЙ ЛОКАЛИЗАЦИИ, НАВИГАЦИИ И УПРАВЛЕНИЯ

Автоматизация погрузочно-разгрузочных процессов является одним из приоритетных направлений в развитии современных промышленных, строительных и горнодобывающих предприятий. В этих отраслях широко применяются фронтальные и мини-погрузчики для перемещения сыпучих материалов, грунта, строительных отходов и других видов грузов. Несмотря на значительный прогресс в механизации и техническом оснащении, большинство подобных операций по-прежнему требует постоянного участия оператора. Это накладывает ограничения на производительность, увеличивает эксплуатационные затраты и снижает повторяемость результатов, что в конечном итоге отражается на общей эффективности производственного процесса.

В контексте перехода к концепции «умных» производств (Smart Factory) особую важность приобретают системы, способные автономно выполнять широкий спектр типовых действий. К таким действиям относятся определение точного положения груза и техники, построение оптимального маршрута перемещения, управление рабочими органами машины, а также обеспечение безопасного возврата к исходной точке после завершения операций. При этом ключевым требованием к таким системам является их адаптивность, устойчивость к ошибкам локализации и способность работать в условиях частичной структурированности среды – например, в карьерах, на складах и строительных площадках, где использование традиционных средств позиционирования, таких как GPS, либо затруднено, либо невозможно из-за недостатка сигнала или сложного рельефа.

Эффективная реализация автономного управления в подобных условиях требует комплексной интеграции различных типов сенсорных данных. Ключевую роль играют визуальные сенсоры, инерциальные измерительные устройства и лидары, позволяющие получать многомерную информацию о

состоянии окружающей среды и положении техники в пространстве. Помимо этого, успешная работа автономной системы зависит от продуманного проектирования модульной архитектуры поведения, которая способна в режиме реального времени адаптироваться к изменениям в рабочей среде и корректировать действия техники в соответствии с текущими условиями.

Таким образом, анализируемая предметная область включает в себя следующие ключевые аспекты:

- автоматизацию специализированной техники в ограниченных или частично структурированных пространствах;
- применение современных подходов к локализации;
- разработку архитектур поведения на базе гибких сценариев управления.

В рамках данной работы рассматривается частный случай такой автоматизации – задача автономной разгрузки мини-погрузчика. Она требует согласованного применения технологий локализации, навигации и программного управления поведением в условиях ограниченного пространства.

Для более детального изучения специфики автоматизации процессов разгрузки представляется целесообразным проанализировать существующие промышленные решения. В последующем разделе рассмотрены современные разработки, отражающие текущее состояние внедрения автономных технологий в данной сфере.

1.1. ОБЗОР АНАЛОГОВ

1.1.1. БЕЛАЗ

Одним из наиболее близких аналогов является автономный погрузчик компании БЕЛАЗ (Рисунок 1), который был протестирован с возможностью удаленного управления на расстоянии 2500км. Оператор находился в Екатеринбурге, в то время как техника работала на полигоне в Белорусии [1].

Рисунок 1 – Беспилотный погрузчик БЕЛАЗ-78250 [1]

Связь между оператором и погрузчиком проходила через каналы спутниковой связи. Это стало значимым шагом в области автоматизации карьерной техники, позволившим продемонстрировать перспективность дистанционного контроля.

Сам погрузчик оснащен современными сенсорами, обеспечивающими контроль над перемещением техники и ее позиционированием в пространстве. Однако, несмотря на внедрение таких технологий, полностью автономным погрузчиком назвать эту модель нельзя. Машина способна двигаться по заранее заданным маршрутам, но для выполнения сложных маневров, требуется уже участие оператора. Управление ковшом осуществляется вручную, что не позволяет исключить человеческий фактор полностью. Вмешательство оператора также необходимо в случае возникновения неожиданных ситуаций на рабочем участке, таких как появление препятствий.

Еще одной проблемой является зависимость работы машины от стабильности соединения. Если сигнал связи будет прерван или возникнут проблемы с передачей данных, оператор может потерять контроль над машиной, что повышает риск инцидентов на рабочем участке. В условиях

реальных карьерных работ, где радиопомехи или неблагоприятные погодные условия могут влиять на качество связи, это становится существенным ограничением.

1.1.2. VOLVO

Volvo LX03 (Рисунок 2) – первый в мире самообучающийся электрический погрузчик с искусственным интеллектом (ИИ). Разработанная инженерами Volvo машина способна самостоятельно адаптироваться к различным рабочим сценариям, принимать решения и взаимодействовать с другой техникой и людьми. Основная особенность данной модели заключается в том, что техника способна обучаться на основе полученных данных, тем самым повышается эффективность работы с течением времени [2].

Рисунок 2 – Колесный погрузчик Volvo LX03 [2]

Программисты Volvo используют алгоритмы машинного обучения, которые позволяют системе анализировать окружающую среду, распознавать объекты и адаптировать поведение машины в реальном времени. Эти

алгоритмы обучаются на больших объёмах данных и способны предсказывать изменения в рабочей среде, что обеспечивает гибкость в выполнении задач.

Система управления основана на данных с множества датчиков таких как: лидар, камеры и ультразвуковые, что позволяет погрузчику строить детальную картину окружения. На основе этих данных ИИ принимает решения, корректирует маршрут движения и манипуляции ковшом. Также концепция этой машины достаточно уникальная. Она использует ножничную раму, позволяющую использовать потенциальную энергию.

Но у любой системы также есть недостатки. Ключевым недостатком является полная зависимость от ИИ. Автономная система принимает решения на основе алгоритмов машинного обучения, что потенциально снижает уровень надёжности в критических ситуациях в нестандартных ситуациях. В критических условиях, например, рядом с человеком система может не всегда реагировать так, как ожидается, что создаёт опасность для окружающих. Кроме того, на данный момент это лишь концептуальная разработка и не применяется в реальных промышленных условиях, поэтому его надёжность невозможно объективно оценить.

1.1.3. SANDVIK

В 2018 году, Sandvik показали, что погрузочно-доставочные машины и самосвалы их производства наработали в общей сложности уже более 2 млн часов без единого происшествия с участием работников предприятий.

Основные требования: автономная горная техника должна уметь быстро и беспрепятственно передвигаться в сложных условиях подземных выработок, что и продемонстрировал новый самоходный погрузчик LH 514 (Рисунок 3) грузоподъемностью 38 т.

Погрузчику удалось преодолеть весь лабиринт подземных горных выработок, избежав касания элементов инфраструктуры, как при включенном, так и при выключенном освещении. Свой путь машина определяет на основе информации, поступающей одновременно с лазерных датчиков, сенсоров,

гироскопов и позволяет погрузчику функционировать в условиях, исключающих использование спутниковой навигации [3].

Рисунок 3 – Погрузчик Sandvik LH 514 [3]

Для наглядного представления различий между рассмотренными автономными погрузчиками предоставлена сравнительная таблица.

Таблица 1 – Сравнительная таблица погрузчиков

Характеристика	**Volvo LX03**	**БЕЛАЗ-78250**	**Sandvik LH514**
Тип погрузчика	Колесный фронтальный	Колесный фронтальный	Самоходный подземный
Грузоподъёмность	5 тонн	22 тонны	38 тонн
Привод	Электрический	Дизельный	Дизельный
Навигация	GPS, AI, LIDAR	Дистанционное управление	LIDAR, гироскопы, лазерные датчики
Автономность	Полностью автономен	Дистанционно управляемый	Полностью автономен
Область применения	Карьеры, стройплощадки	Карьеры, открытые работы	Подземные шахты и рудники
Энергопотребление	Аккумулятор до 8 часов	Топливный двигатель	Топливный двигатель
Требования	Зарядные станции	Радиосвязь с оператором	Централизованная система управления AutoMine

Продолжение таблицы 1

Характеристика	Volvo LX03	БЕЛАЗ-78250	Sandvik LH514
Преимущества	Автономность, экологичность	Высокая нагрузка	Безопасность работы в шахтах, работа без GPS
Недостатки	Концептуальный статус	Требуется оператор	Не предназначен для открытых работ

В рамках данного исследования целью обзора аналогов было изучение не конкретных моделей погрузчиков, а технологий автоматизации, применяемых в различных системах автономного управления. Анализируемые машины были выбраны, поскольку они реализуют различные принципы навигации, автономности и взаимодействия с окружающей средой, что позволяет выделить ключевые методы для разработки собственного решения.

На основе данных, представленных в таблице 1, для разработки системы будут учтены следующие особенности существующих решений:

- Volvo LX03 – принципы автономного управления, но без полной зависимости от ИИ, что обеспечит предсказуемость работы;
- БЕЛАЗ-78250 – механизмы дистанционного управления, которые будут адаптированы под автономную навигацию;
- Sandvik LH514 – использование лидаров и лазерных датчиков для локализации, но с учетом возможности работы в условиях GPS.

Таким образом, разработка направлена не на создание нового погрузчика, а системы, которая обеспечит автономное движение и разгрузку мини-погрузчика в самосвал в различных условиях эксплуатации.

На основе проведенного анализа можно выделить ключевые технологии, применяемые в существующих решениях, и адаптировать их для решения поставленной задачи. В следующем разделе будут рассмотрены основные технологические компоненты, которые планируется использовать в разработке.

1.2. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧКСКИХ РЕШЕНИЙ

Обзор аналогов показал, что на рынке существуют различные модели погрузчиков для решения реализации автономного управления. Однако для решения задачи автономной разгрузки мини-погрузчика в самосвал требуется комбинация нескольких технологий, которые обеспечат точное позиционирование, стабильное управление и предсказуемость системы.

1.2.1. Методы локализации

Локализация роботов предполагает определение их положения и ориентации в пространстве с учетом текущего окружения. Эта задача решается с использованием различных подходов (Рисунок 4): от одометрии до сложных методов SLAM (Simultaneous Localization and Mapping). Каждый метод имеет свои ограничения, такие как шумы, погрешности сенсоров или необходимость в вычислительных ресурсах [4].

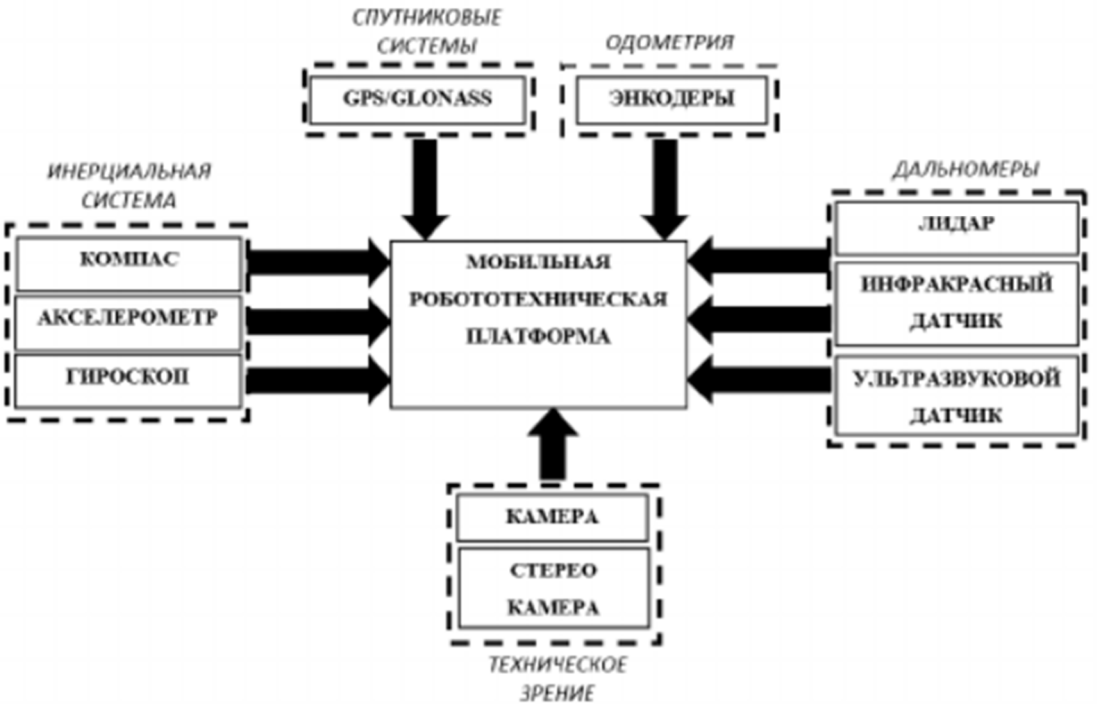

Рисунок 4 – Структурная схема мобильной робототехнической платформы для решения задач позиционирования в пространстве [4]

Визуальная одометрия – это современный и широко используемый метод оценки положения и ориентации мобильного робота или автономной платформы, основанный на анализе последовательностей изображений, получаемых с одной или нескольких камер, установленных на роботе. Суть метода заключается в использовании визуальной информации окружающей среды – таких особенностей, как текстуры, углы, контуры объектов, линии и другие уникальные элементы – для определения относительного перемещения робота относительно его начального положения.

Визуальная одометрия позволяет оценивать смещения и повороты робота, анализируя, как изменяются ключевые точки на последовательных изображениях. При этом особое внимание уделяется выявлению стабильных и хорошо различимых особенностей (ключевых точек) на изображениях, которые могут быть надёжно сопоставлены между собой при движении робота. Это обеспечивает вычисление траектории движения без необходимости использования внешних навигационных систем.

Алгоритм визуальной одометрии с использованием стереоизображений основывается на сравнении изображений одного и того же объекта, снятых двумя камерами, расположенными на известном фиксированном расстоянии друг от друга. Благодаря этому можно вычислить глубину и трехмерные координаты точек окружающего пространства. Идея алгоритма состоит в том, чтобы на стереоизображениях выделить набор ключевых точек, обладающих определёнными свойствами – например, углы зданий, текстурные пятна на поверхности или уникальные элементы ландшафта. После этого отслеживается изменение положения этих точек при движении робота, что позволяет вычислить его смещение и угол поворота в пространстве.

Визуальная одометрия показывает наилучшие результаты в средах с богатой текстурой и большим количеством уникальных визуальных признаков, например, в природных ландшафтах, городских условиях с разнообразной архитектурой, а также на производственных площадках с различным оборудованием. Одним из важных преимуществ метода является

его относительная независимость от внешних систем позиционирования, таких как GPS, что особенно важно в условиях, где спутниковый сигнал недоступен или ненадёжен – например, в помещениях, под землёй или в условиях плотной городской застройки.

Для практической реализации визуальной одометрии часто используют современные программные платформы и библиотеки, например, ORB-SLAM, VINS-Mono и другие, которые интегрируются с популярными робототехническими системами, такими как ROS (Robot Operating System). Эти инструменты обеспечивают удобный и эффективный доступ к алгоритмам обнаружения ключевых точек, построения карты и оценки положения робота в реальном времени.

Кроме того, визуальная одометрия может быть значительно улучшена за счёт интеграции данных с инерциальных измерительных устройств (IMU) и GPS. IMU обеспечивает измерения ускорений и угловых скоростей, что позволяет компенсировать кратковременные ошибки и проскальзывания, характерные для чисто визуальных методов. GPS же может применяться в открытых пространствах для коррекции накопленных ошибок локализации, обеспечивая тем самым более высокую точность и надёжность навигации.

Однако у визуальной одометрии есть и свои ограничения. Метод теряет эффективность в условиях плохого освещения, например, при работе ночью или в помещениях с недостаточным светом, так как качество и количество визуальной информации значительно снижается. Также метод менее точен в однообразных и мало текстурированных средах, таких как пустыни, заснеженные равнины или гладкие бетонные поверхности, где уникальных ключевых точек мало или они плохо различимы.

В статье [7] подробно рассматриваются алгоритмы визуальной одометрии, включая методологию выделения ключевых точек, обработку стереопар изображений и вычисление перемещений робота. Важной особенностью данной работы является сочетание теоретического обоснования алгоритмов с результатами экспериментальных исследований, которые

подтверждают высокую точность и надёжность предложенных методов в различных условиях эксплуатации. Возможно, визуальная одометрия подходит для нашей задачи, потому что:

- устойчивость к проскальзыванию гусениц;
- возможность работы в сложных условиях, таких как пересечённая местность;
- легкость интеграции с платформой ROS и готовыми библиотеками;
- возможность комбинирования с другими сенсорами для достижения высокой точности.

Лидар (LiDAR, Light Detection and Ranging) устройство, предназначенное для измерения расстояния до объектов с использованием лазерных импульсов. Он фиксирует время, за которое сигнал отражается от препятствия и возвращается, что позволяет создать карту окружающей среды (Рисунок 5). Формула (1) для расчета расстояния:

$$d = \frac{ct}{2}, \quad (1)$$

Где: d – расстояние до объекта, c – скорость света ($3*10^8$ м/с), t – время, необходимое для прохождения сигнала туда и обратно.

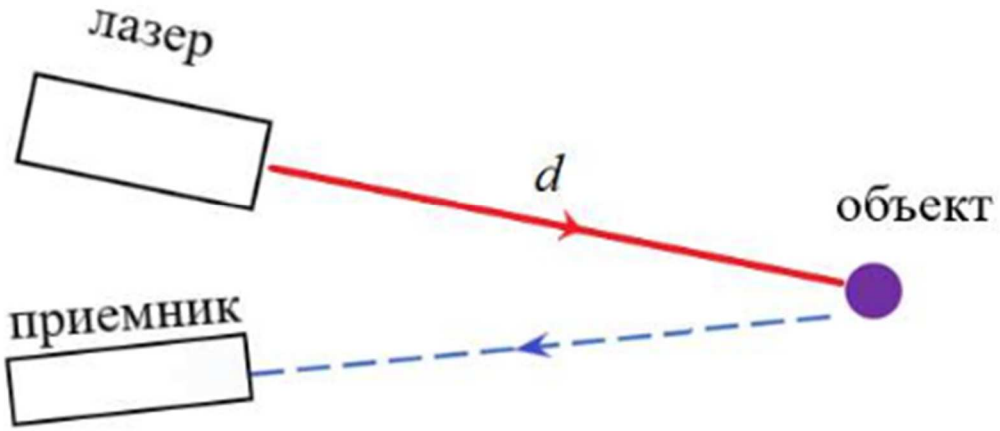

Рисунок 5 – Принцип действия лидара [7]

Полученный световой сигнал преобразуется на приемнике в аналоговый, и далее преобразуется в цифровой при помощи аналого-цифрового преобразователя (АЦП), формируя цифровой «отпечаток» точки. Благодаря совокупности этих точек проецируется модель объекта. а затем отображается на дисплее в реальном времени, либо сохраняется на каком-либо накопителе для дальнейшей обработки. Частоты излучения импульсов современных лидаров могут достигать 150 ГГц [8].

Преимущества лидаров:

- высокая точность и детализация измерений (до миллиметров) даже в сложных условиях;
- возможность создания 2D и 3D карт для задач локализации и построения маршрутов.

Недостатки лидаров:

- высокая стоимость лидаров высокого качества;
- снижение точности при неблагоприятных погодных условиях (дождь, туман).

Применение лидаров в задачах локализации подробно описано в работе Мезенцевой А.Д. (2018). Автор акцентирует внимание на использовании лидара для обнаружения препятствий и построения карт в реальном времени, подчёркивая их значимость для автономных систем [9].

Лидарная одометрия – это метод определения движения робота в пространстве на основе анализа облаков точек, полученных от лидара. Вместо отслеживания углов вращения колёс или анализа изображений, этот подход основывается на трёхмерной геометрической информации. Одометрия только с использованием лидара может быть классифицирована на три типа в зависимости от того, как выполняется сопоставление со сканированием: прямое сопоставление, сопоставление на основе характеристик и сопоставление на основе глубокого обучения. В источнике [10] рассматривается лидарная одометрия как метод локализации,

основывающийся исключительно на данных, предоставляемых лидаром. Авторы классифицируют методы лидарной одометрии на три основные категории в зависимости от подхода к сопоставлению сканирований. Статья подчёркивает перспективы использования методов глубокого обучения для лидарной одометрии, особенно в условиях динамичной среды.

Несмотря на высокую точность измерений, лидар имеет и ряд недостатков. Одна из главным проблем это большие данные, лидарная система генерирует объёмное 3Д-обако точек, содержащее обширные данные об окружающей среде и объектах. Он дает значительное преимущество в получении 3D-информации об окружающей среде, однако существуют проблемы, связанные с его размером. Размер этого облака зависит от поле зрения самого лидара и от его разрешения. Например, лидар OS1-128 может производить сканирование, содержащее значительное количество точек, которое может достигать в 100 тысяч точек на кадр, работая с максимальной частотой 20 Гц. Кроме того, каждая точка в облаке содержит информацию о себе (дальность действия, интенсивность, отражательная способность, условия окружающей среды и время обнаружения точки) что вносит вклад в объём данных. Обработка таких обширных данных в режиме реального времени требует значительных вычислительных мощностей. Также присутствует проблема искажения движения. Когда робот движется с высокой скоростью относительно частоты получения данных датчиком, может возникнуть значительный пространственный разрыв между местами, где были получены данные, в начале и в конце одного сканирования лидаром и это разрыв может повлиять за собой к значительным искажениям к лидарному сканированию. Еще одна проблема, что лидар способен измерять большие расстояния, имеет присущие ему ограничения. Одним из заметных недостатков является его узкий обзор, что особенно проблематично для задач восприятия. Данные, полученные с лидара, характеризуются меньшей плотностью по сравнению с изображениями RGB-камер даже несмотря на то, что поле зрения у лидара по горизонтали обычно шире. Кроме того, при

использовании механического лидара они обычно устанавливаются на открытие площадки для того, чтобы была 360-градусная видимость и это создает проблемы с защитой датчика от внешних воздействий, а если защитить датчик и установить его в более защищённом месте может привести к потере полей зрения.

IMU (инерциальные измерительные устройства) – это комплексные сенсорные системы, которые предназначены для измерения линейных ускорений, угловых скоростей и, в некоторых случаях, магнитного поля окружающей среды. В робототехнике и автономных системах IMU являются одним из основных инструментов для оценки ориентации, скорости и перемещения мобильных платформ, особенно в условиях, когда традиционные методы позиционирования, такие как GPS или визуальная одометрия, оказываются недостаточно надёжными или вовсе недоступными.

Стандартный IMU состоит из нескольких ключевых компонентов: акселерометров, гироскопов и, при необходимости, магнитометров. Каждый из этих сенсоров играет важную роль в формировании полной картины движения и положения объекта в пространстве.

Акселерометры измеряют величину линейных ускорений, воздействующих на систему, по трем взаимно перпендикулярным осям. Эти данные позволяют определить, с какой силой и в каком направлении движется платформа. Путём интегрирования ускорения по времени можно получить скорость движения объекта, а повторное интегрирование позволяет вычислить пройденное расстояние или перемещение. Однако этот процесс интегрирования сопровождается накоплением ошибок, вызванных шумами сенсоров и внешними помехами, что снижает точность расчётов на длительных интервалах времени. Для минимизации таких ошибок широко применяются различные алгоритмы фильтрации, например, фильтр Калмана [11], фильтр «альфа-бета» и другие методы, которые позволяют улучшить качество данных и снизить влияние случайных возмущений.

Гироскопы измеряют угловую скорость вращения объекта вокруг трёх осей: продольной (X), поперечной (Y) и вертикальной (Z). Информация о скоростях вращения позволяет рассчитывать угол поворота и ориентацию платформы в пространстве. После интегрирования угловой скорости по времени можно получить углы вращения относительно каждой из осей. Так, вращение вокруг продольной оси X называется углом крена – он характеризует наклон платформы влево или вправо; вращение вокруг поперечной оси Y обозначается как угол тангажа и отражает наклон вперёд или назад; вращение вокруг вертикальной оси Z называют углом курса (или рыскания), который показывает поворот платформы в горизонтальной плоскости. Первые два угла (крен и тангаж) непосредственно связаны с положением объекта относительно земной поверхности и являются критически важными для поддержания устойчивости и правильного положения мобильных роботов [12].

Магнитометры, если они включены в состав IMU, служат для измерения магнитного поля Земли и помогают корректировать ошибки в определении ориентации, возникающие из-за дрейфа гироскопов и внешних магнитных помех. Это позволяет улучшить точность оценки азимута и направления движения.

В целом, IMU является незаменимым элементом систем автономной навигации и локализации, обеспечивая высокую частоту обновления данных о движении и ориентации, что особенно важно для динамичных и быстро меняющихся условий работы мобильных роботов. Совместное использование IMU с другими сенсорными системами – например, с визуальными или радиолокационными – позволяет создавать гибридные системы навигации, которые компенсируют ограничения каждой из технологий и обеспечивают стабильную и точную работу автономных платформ.

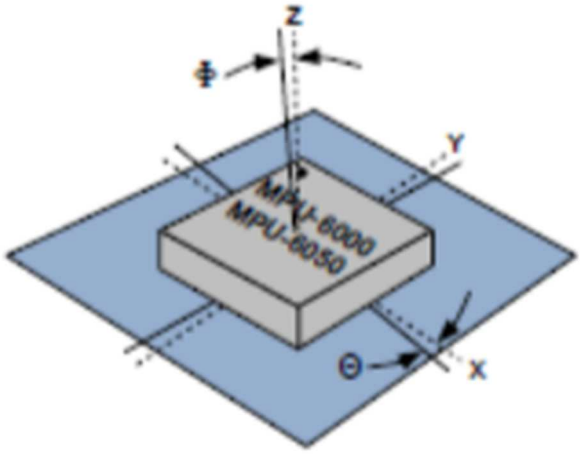

Рисунок 6 – Положение осей относительно платы [12]

Глобальная система позиционирования (GPS, Global Positioning System) основана на использовании спутниковых сигналов для определения координат объекта. Спутники передают радиосигналы, которые принимаются GPS-приемником, и по разнице во времени прихода сигналов вычисляются координаты устройства [13].

Преимущества GPS:

- стабильная работа в открытом пространстве;
- высокая точность при использовании RTK (Real-Time Kinematics).

Недостатки:

- погрешность может достигать до нескольких метров;
- плохо работает в закрытых и подземных помещениях;
- задержка при обновлении данных.

В данной работе GPS может быть использован в качестве вспомогательного средства для локализации, но не является основным инструментом навигации, так как для маневрирования мини-погрузчика требуется сантиметровая точность недостижима при использовании только GPS-приёмников.

ArUco-маркеры представляют собой черно-белые квадратные маркеры с уникальными кодами, которые используются для определения положения объектов (самосвала) в пространстве с помощью камер. Маркеры широко

применяются в задачах робототехнике т.к. позволяют роботу точно вычислять координаты относительно известного объекта [14].

Принцип работы маркера:

1. Камера погрузчика фиксирует маркер, прикрепленный к самосвалу.
2. Алгоритм распознавания маркера, определяет угол и расстояние.
3. На основе параметров камеры и размеров маркера, вычисляются трехмерные координаты самосвала.

Благодаря использованию маркеров можно получить высокую точность (в пределах сантиметров), а также быструю обработку. Но важно понимать, что ArUco-маркер используется для уточнения финального положения погрузчика перед разгрузкой, но не заменяет глобальные методы навигации.

Для обеспечения точной и надежной локализации погрузчика перед разработкой модуля разгрузки необходимо комбинировать данные датчики и сглаживать шумы. В данной работе используется пакет robot_localization [15], который позволяет объединять данные от IMU, GPS и лидара, используя фильтрацию методом Калмана. Важное преимущество этого пакета – гибкость в настройке, позволяет учитывать различную конфигурацию сенсоров, а также благодаря объединение данных через EKF позволяет создать стабильную картину местоположения погрузчика, обеспечивая точную навигацию.

1.2.2. Современные методы навигации для автономных роботов

Современные автономные роботы используют различные методы навигации и управления поведением, которые обеспечивают адаптивность, надежность и эффективность работы в сложных условиях. Среди самых популярных подходов – конечные автоматы, обучение с подкреплением и деревья поведения. Каждый из этих методов имеет свои особенности, преимущества и ограничения, которые важно учитывать при разработке систем навигации.

Конечные автоматы (Finite State Machines, FSM) – это математическая модель, описывающая систему с конечным числом состояний и переходов между ними. Используется для представления и управления потоком выполнения каких-либо команд [16].

Они широко применяются в задачах робототехники, управления движением и взаимодействия с окружающей средой, предлагая предсказуемую и детерминированную логику поведения.

FSM базируется на представлении поведения системы в виде множества состояний и переходов между ними, где каждое состояние описывает конкретную задачу, а переходы определяются заранее заданными условиями (Рисунок 7).

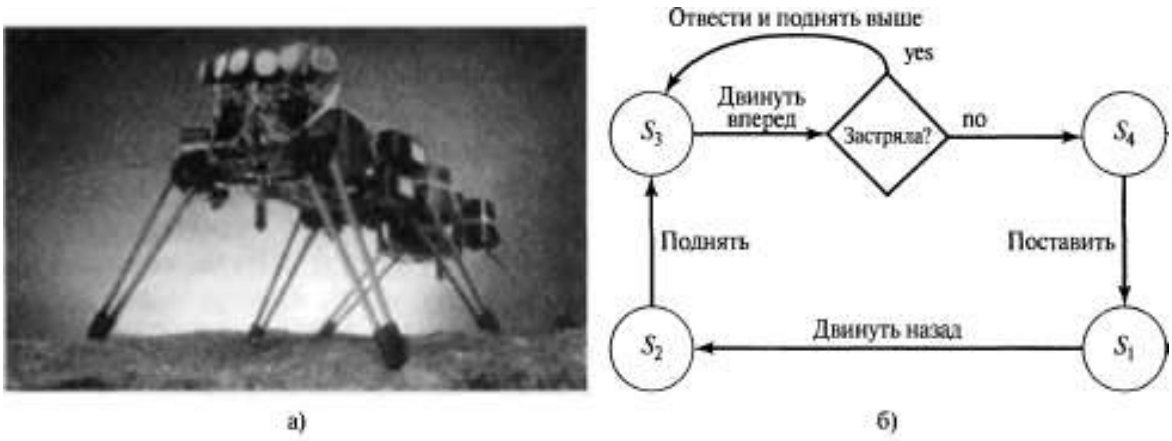

Рисунок 7 – Пример работы FSM [16]

Рисунок 7 иллюстрирует, как FSM позволяет разбить задачу управления движением на отдельные этапы и установить логические условия перехода между ними. Это необходимо для адаптивного передвижения робота, особенно в случае непредвиденных препятствий.

В работе [16] рассматривается использование конечных автоматов в децентрализованном эволюционном управлении группой мобильных роботов. В данном подходе FSM используются для адаптации поведения агентов в реальном времени на основе эволюционных алгоритмов.

Несмотря на все, конечные автоматы ряд ограничений, в основном это проблемы с обработкой неопределенности, FSM требует явного задания всех состояний и переходов, что делает его не очень гибким в динамически изменяющихся условиях.

Обучение с подкреплением (Reinforcement Learning, RL) - подразумевает, что робот самостоятельно обучается навигации через пробу и ошибки (Рисунок 8). Агент взаимодействует с окружением: получает наблюдение, выполняет какие-то действия и получает вознаграждения (за достижения цели).

Рисунок 8 – Принцип работы RL [17]

Алгоритмы RL могут превосходить жестко запрограммированные методы в динамичных или неизвестных условиях. Например, в окружении с движущимися препятствиями или неполной информации агент может выработать стратегию обхода в реальном времени, тогда как консервативные алгоритмы могут запутаться. Исследования показывают, что DRL (deep RL) может обеспечить движение без столкновений там, где традиционная навигация буксует из-за непредвиденных помех [17]. Со временем обучение с подкреплением улучшает свои навыки за счет накопленных знаний. Например, автономный погрузчик, оснащенный RL, мог бы постепенно оптимизировать свои маршруты разгрузки, анализируя прошлые поездки. Ну

и главный плюс, что вместо ручной настройки правил (как в FSM), RL позволяет получить политику навигации автоматически путем оптимизации, это особенно удобно, когда тяжело заранее предусмотреть все правила для робота.

Основным недостатком является длительное обучение: для освоения базовых навыков RL требуется сотни тысяч шагов [17]. Также необходимо тщательно проектировать функцию вознаграждения – неправильная формулировка может привести к нежелательному поведению агента. В условиях промышленной техники обучение на реальном оборудовании рискованно, поэтому требуется достоверная симуляция (например, цифровой двойник), перенос модели из которой в реальный мир затруднён. При изменении условий (планировка, препятствия) требуется дополнительное обучение или повторная тренировка модели, что усложняет применение метода.

В статье [18] доступно принцип работы метода машинного обучения RL на примере физической системы. Автор демонстрирует реализацию алгоритма Q-Learning на языке Python, что позволяет понять основы применения RL в практических задачах.

Дерево поведения (Behavior Tree) – это иерархическая модель поведения, которая разветвляется на узлы и выполняется итеративно (Рисунок 9). В основе BT лежит механизм тиков (tick), с заданной чистотой корневой узел запускает обход дерева, проверяя условия и выполняя действия. Каждый узел при активации возвращает один из трех статусов: Succes (успех), Failure (провал), Running (выполнение) [19]. Этот цикл повторяется непрерывно, обеспечивая реактивность – способность реагировать на изменения среды или условий в реальном времени.

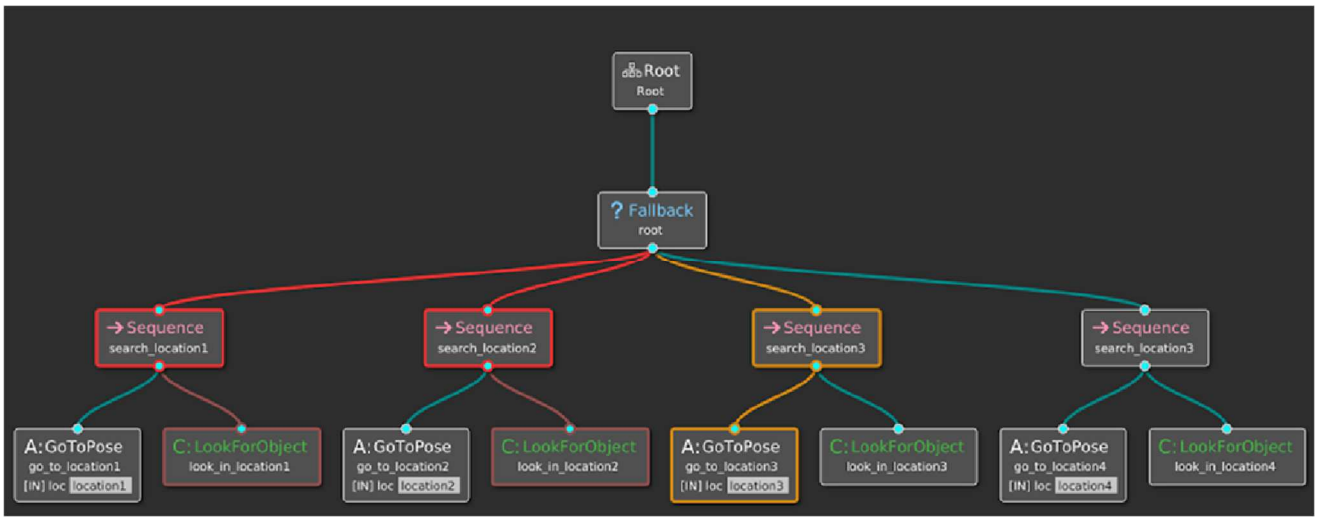

Рисунок 9 – Принцип действия BT

Структура дерево состоит из разных узлов, каждый из которых выполняет свою роль в управлении поведением:

- корневой узел (root) – точка входа, с нее начинается тик обхода дерева. У корня обычно 1 ребенок, этот ребенок является поддеревом поведения;
- внутренние узлы – определяет логику дочерних узлов. Исходя и рисунка 9 это: Fallback (запасной вариант), Sequence (последовательность);
- листовые узлы – конечные узлы дерева, которые не имеют потомков. Их две категории: Action (действие) и Condition (условие). Action-узлы выполняют определенное действия робота, а Condition-узлы для проверки состояния и обычно сразу возвращают успех или провал.

Рассмотрим, как дерево поведения может быть применена для реализации сценария автономной разгрузки погрузчика. Глядя на дерево, можно визуально проследить логику: сначала делай А, если не получилось делай Б и В, потом Г и т.д.

Это облегчает отладку и сопровождение. Например, при необходимости добавления нового манёвра его можно легко интегрировать, не переписывая сложную логику переходов конечных автоматов (FSM).

Таблица 2 – Сравнительная таблица систем навигации

Критерий	**Behavior Trees (BT)**	**Finite State Machines (FSM)**	**Reinforcement Learning (RL)**
Гибкость и масштабируемость	Высокая, нет проблем добавлять новые ветви без изменения других частей	Низкая, сложность растет экспоненциально с увеличением состояний	Очень высокая, может обучаться новым стратегиям
Предсказуемость и надежность	Высокая, поведение строго определяется структурой дерева	Высокая, но сложность управлениями состояниями затрудняет сопровождение	Средняя, зависит от обучения, может быть непредсказуемым
Реактивность	Очень высокая	Низкая	Высокая
Сложность	Средняя	Средняя	Очень высокая
Вычислительные ресурсы	Низкие	Низкие	Высокие
Подходит для нашей задачи с разгрузкой	Да, идеально подходит для автономного выполнения задач с адаптацией к окружению	Может быть использована, но требует ручного программирования всех переходов	Нет, нет гарантий безопасности

В данной выпускной квалификационной работе (ВКР) Behavior Trees рассматриваются как наиболее эффективный и удобный инструмент управления поведением. В рамках ВКР планируется внедрение BT в мини-погрузчик для обеспечения автономного последовательного выполнения действий. В дальнейшем система будет развиваться и модернизироваться. На начальном этапе для локализации предполагается использование ArUco-

маркеров, однако в перспективе система сможет функционировать без них. Благодаря гибкости Behavior Trees, при необходимости изменений не требуется перестраивать всё дерево, достаточно корректировать отдельные листовые узлы. Таким образом, дерево поведения является не только эффективным решением для автономной разгрузки, но и перспективной основой для дальнейшего развития системы.

1.3. ВЫВОД ПО ГЛАВЕ ОДИН

В результате анализа предметной области выявлено, что современные автономные погрузчики применяют различные технологии локализации и навигации, включая GPS, лидары, визуальную одометрию и методы машинного обучения. Рассмотренные аналоги продемонстрировали различные подходы к обеспечению автономности и безопасности работы в сложных условиях.

В качестве компромиссного решения выбрана архитектура управления на основе дерева поведения, которая сочетает иерархичность, модульность и предсказуемость работы системы в реальном времени.

2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ

При разработке программного модуля для автономной разгрузки важно чётко сформулировать требования, обеспечивающие достижение цели и объективную оценку результата.

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Рисунок 10 – Диаграмма последовательностей автономного модуля разгрузки мини-погрузчика (UML, ГОСТ Р ИСО/МЭК 19501-2006)

На рисунке 10 представлена диаграмма последовательности, отражающая процесс взаимодействия оператора с автономным мини-погрузчиком во время задачи автономной разгрузки. Шаги выполняются в следующем порядке:

1. Оператор отправляет погрузчику команду на начало автономной разгрузки.

2. «После получения команды погрузчик начинает автономный процесс:

- сначала выполняется определение местоположения самосвала с использованием ArUco-маркеров;
- затем выбирается оптимальная точка разгрузки, исходя из текущих условий окружающей среды;
- мини-погрузчик автономно позиционируется в выбранной точке относительно самосвала;
- далее происходит непосредственная разгрузка материала в самосвал;
- после успешной разгрузки погрузчик возвращается в исходную точку;»

3. По завершении всех действий погрузчик отправляет оператору отчет о выполнении задачи.

Использование такой схемы взаимодействия позволяет минимизировать участие человека в процессе разгрузки и обеспечить высокую точность и безопасность выполняемых операций.

На основе приведённого сценария были сформулированы и детализированы следующие функциональные требования к системе (таблица 3).

Таблица 3 – Классификация функциональных требований к системе.

Категория	**Описание требования**
Приём команды оператора	Система должна принимать команду оператора через интерфейс ROS2 на начало автономной разгрузки
Автоматическое распознавание ArUco-маркера	Система должна самостоятельно обнаруживать ArUco-маркер с помощью стереокамеры ZED 2i
Определение позиции разгрузочной площадки	Вычисление координат точки разгрузки относительно обнаруженного ArUco-маркера с точностью до 10 см

Продолжение таблицы 3

Категория	Описание требования
Автоматизированная разгрузка материала	Автоматическое поднятие и опускание ковша с разгрузкой материала в течение не более 20 секунд
Автономный возврат в исходную точку	Планирование и реализация обратного маршрута с точностью до 5 см
Точное позиционирование	Автономное позиционирование погрузчика перед разгрузкой с точностью до 10 см
Планирование безопасной траектории	Автономное позиционирование погрузчика перед разгрузкой с точностью до 10 см

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Основным требованием к системе является обеспечение безопасности, так как она выполняет автономные операции, представляющие потенциальную опасность для людей и техники. Для этого должна быть реализована система предотвращения столкновений с использованием датчиков, а также предусмотрена возможность экстренного останова оператором в любой непредвиденной ситуации, что снижает риски травм и повреждений оборудования.

Кроме того, система должна обладать высокой надёжностью и отказоустойчивостью, обеспечивая стабильную работу на протяжении всего процесса и быструю реакцию на возможные сбои с восстановлением функциональности.

Разработка программного модуля осуществляется с учётом совместимости с платформой ROS. При этом необходимо соблюдать стандарты оформления кода, передачи данных, типов сообщений и организации узлов системы.

Также необходимо обеспечить удобство эксплуатации системы оператором. Интерфейс должен предоставлять понятные и информативные уведомления, способствующие эффективному управлению и мониторингу.

Таблица 4 – Классификация нефункциональных требований к системе.

Категория	**Описание требования**
Надежность	Система должна функционировать без сбоев на протяжении всего цикла разгрузки. Возможны попытки восстановления при частичных отказах
Отказоустойчивость	При возникновении неисправности (например, потеря связи с датчиком) система должна сохранять работоспособность или корректно завершать текущую задачу
Безопасность	Система должна предотвращать столкновения с объектами. Предусмотреть механизм аварийной остановки, активируемый оператором
Совместимость	Программный модуль должен быть совместим с архитектурой ROS 2
Расширяемость	Возможность добавлять новый функционал в модуль без существенной переработки логики
Производительность	Время отклика должно быть минимальным
Обслуживаемость	Код должен быть структурирован в соответствии с требованиями ROS
Стандартизация реализации	Основной язык программирования – C++, соответствующий требованиям ROS и используемый в ключевых библиотеках проекта (в частности, BehaviorTree.CPP и Nav2)

2.3. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Для реализации автономного модуля разгрузки была разработана модульная архитектура, основанная на платформе ROS 2 (Robot Operating System), представляющей собой популярный фреймворк с открытым исходным кодом для разработки сложных и распределённых робототехнических систем.

ROS [19] – это широко используемый фреймворк для создания сложных, распределенных робототехнических систем. ROS представляет собой мета-операционную систему с открытым исходным кодом, структура которой

представлена в виде узлов (ROS-node), которые обмениваются сообщениями между собой [20]. Узлы – это программы, написанные на языках Python или C++, созданные пользователями. Каждый узел отвечает за конкретную задачу, например, управление двигателями или построение карты местности. Узел, как и любая другая программа, поддерживает различные библиотеки. Общение узлов происходит путем передачи сообщений в различные шины данных, именуемых топиками (ROS-topic). Сообщение представляет собой структуру данных, содержащую типизированные элементы. ROS содержит множество стандартных сообщений различных типов, но при необходимости можно создать свой тип сообщений. Каждый узел может получать и отправлять сообщения в различные топики. Принцип работы показан на рисунке 11.

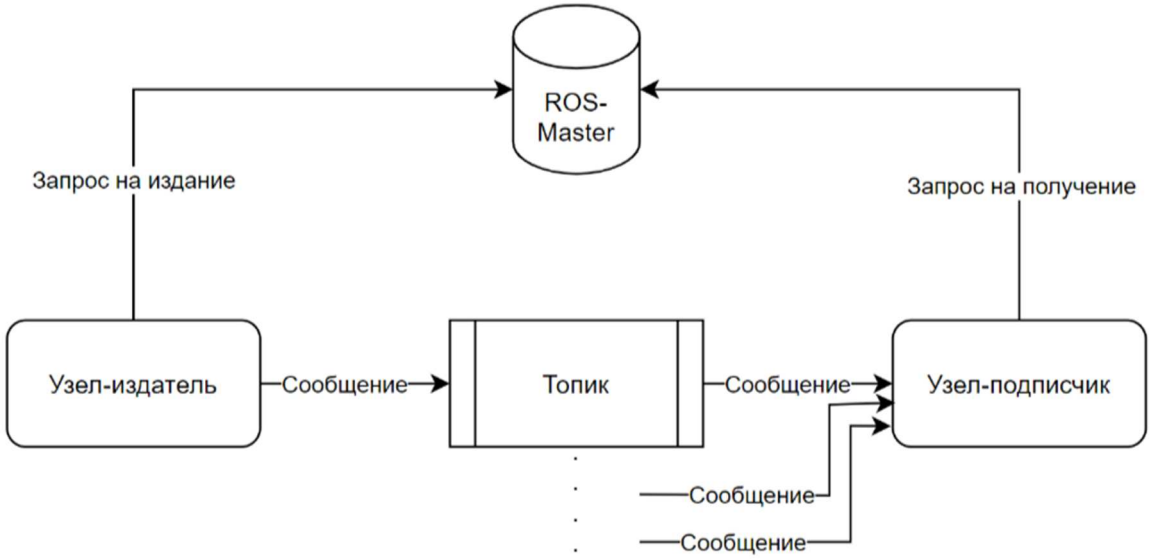

Рисунок 11 – Принцип работы ROS (ГОСТ 19.701-90. ЕСПД.)

ROS выбран в качестве платформы для реализации программного модуля автономного мини-погрузчика. Этот выбор обусловлен следующими ключевыми преимуществами:

1. Модульность – фреймворк поддерживает раздельную архитектуру, где каждая часть системы может быть реализована как отдельный

узел. Позволяет легко заменять или улучшать компоненты без переделки всей системы.

2. Поддержка многопоточности и работы в реальном времени – в отличии от ROS 1, новая версия (ROS2) оптимизирована для работы с реальными роботами.
3. Интеграция с Behavior Trees (BT) – ROS 2 поддерживает BehaviorTree.CPP [21], что делает его идеальной платформой для реализации автономных сценариев, таких как автономная разгрузка.

В качестве основного инструмента для решения задачи автономного передвижения робота выбран Navigation Stack 2 (Nav2) [22]. Данный стек навигации является стандартным решением для ROS 2 и предназначен для реализации локализации, планирования траектории и автономного перемещения (Рисунок 12).

Рисунок 12 – Архитектура навигационного стэка (Nav2)

Центральным элементом системы является BT Навигатор сервер, координирующий работу остальных модулей через дерево поведения.

Архитектура включает следующие основные компоненты:

- сервер восстановления (Recovery Server) – активируется при возникновении нештатных ситуаций. Он может выполнять повторные попытки или альтернативные действия, используя локальную карту препятствий;
- сервер контроллеров (Controller Server) – реализует управление движением робота в соответствии с построенной траекторией;
- сервер планировщика (Planner Server) – осуществляет построение безопасного маршрута до цели, используя карту среды;
- сервер сглаживания (Smoother Server) – улучшает траекторию, сделанную планировщиком, для более плавного движения;
- сервер маршрутов (Waypoint Follower) – обеспечивает выполнение маршрутов, состоящих из нескольких точек, при необходимости. Поддерживает повторение заранее заданных траекторий;
- модули ввода (TF, Map, Sensor Data) – предоставляют данные о карте, трансформациях и сенсорах, которые используются для построения и обновления локальной и глобальной карты;
- узлы «Следование за точками» и «Система автономности» – пользовательские или надстройки, которые инициируют запуск дерева поведения и управляют высокоуровневой логикой робота;
- менеджер жизненного цикла (Lifecycle Manager) – обеспечивает последовательную активацию, деактивацию и перезапуск узлов в соответствии с состоянием системы.

Навигационная архитектура реализуется на базе Nav2 и использует готовые компоненты ROS 2. Это позволяет избежать повторной реализации базовых алгоритмов планирования и управления движением. На практике верхний уровень управления передаёт в систему цель (координаты и

ориентацию), а остальная логика выполняется Nav2 автоматически. Это освобождает разработчика от необходимости реализовывать низкоуровневые функции и позволяет сосредоточиться на проектировании сценариев поведения погрузчика, которые реализуются средствами Behavior Tree.

Ключевым элементом архитектуры управления является Behavior Tree (BT), обеспечивающее формализацию и последовательность выполнения действий автономной системы. В рамках данной работы дерево поведения разработано как основной механизм координации модулей обнаружения цели, приближения к точке разгрузки, выполнения разгрузки и возвращения в начальное положение. На текущем этапе полностью описана структура BT: определены управляющие узлы и прикладные листовые узлы, отвечающие за выполнение конкретных действий (например, навигация к цели, управление ковшом и т.д.) (Рисунок 13).

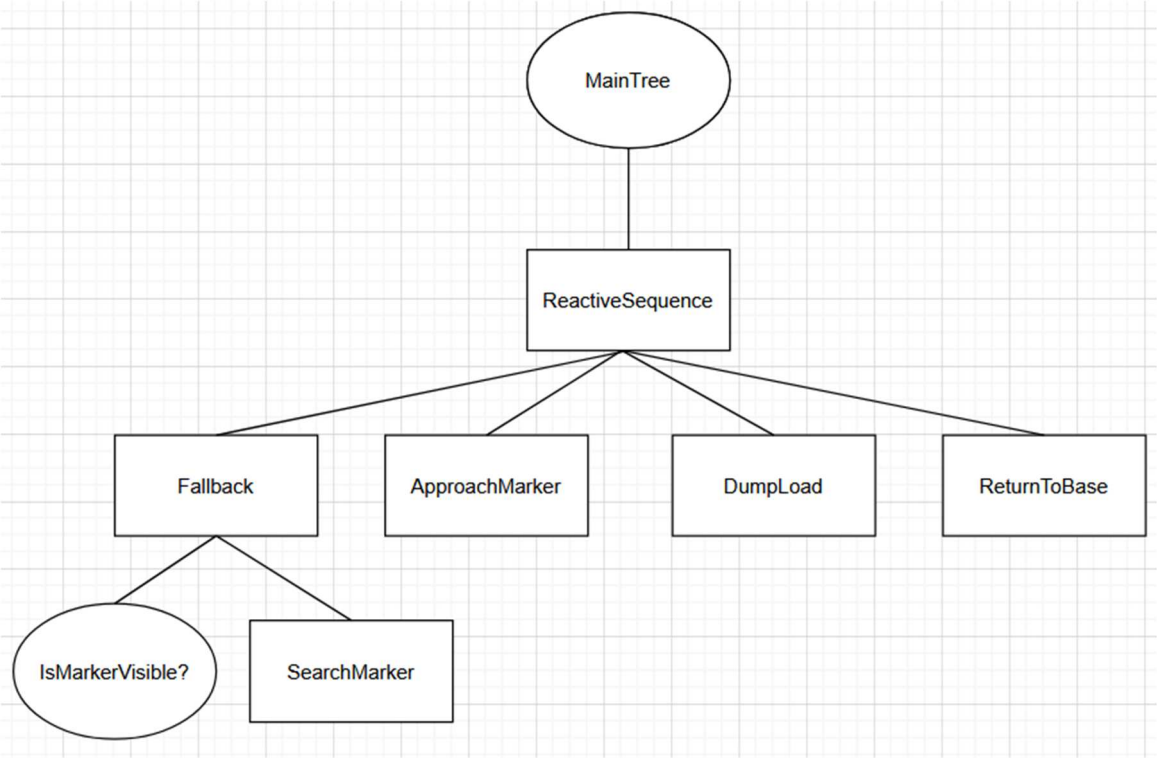

Рисунок 13 – дерево поведения для автономной разгрузки (условные обозначения по ГОСТ 19.701-90, ЕСПД)

Разрабатываемая структура дерева (Рисунок 13) включает в себя несколько поддеревьев, каждое из которых отвечает за конкретное действие (Action-плагин [23]). В основе BT находится управляющий узел «ReactiveSequence» обеспечивающий реактивную проверку условий и выполнение действий в строго определенной последовательности. Под реактивностью понимается способность дерева поведения на каждом тике (обновлении) переоценивать условия и автоматически отменять или переключать действия, если текущая ситуация изменилась. Например, при неожиданном появлении препятствия.

На первом этапе осуществляется проверка наличия ArUco-маркера, обозначающего положения самосвала. Это логика реализуется с помощью узла Fallback, который содержит два дочерних элемента: условный узел (decorator) «IsMarkerVisible?», проверяющий наличие маркера в текущем кадре камеры и поведенческий узел «SearchMarker», активирующий поиск в случае отрицательного результата условного узла.

После успешного определения положения маркера активируется узел «ApproachMarker», обеспечивающий движение погрузчика в сторону точки разгрузки. Навигация при этом осуществляется средствами Nav2, как было описано выше.

Следующий узел «DumpLoad» отвечает за выполнение операции разгрузки. На данном этапе его реализация планируется в виде пользовательского плагина, интегрируемого в дерево поведения. Этот узел будет отправлять управляющие команды для подъёма стрелы и наклона ковша, при этом предполагается наличие механизма обратной связи для контроля выполнения действий.

2.4. ВЫВОД ПО ГЛАВЕ ДВА

В данной главе было проведено комплексное проектирование архитектуры программного модуля системы автономной разгрузки. На основе анализа требований и технических характеристик задачи обоснован и выбран оптимальный архитектурный подход, обеспечивающий гибкость, масштабируемость и надежность работы системы. Рассмотрены ключевые технологические решения, которые позволяют эффективно реализовать функционал модуля с учетом ограничений аппаратной платформы и требований к быстродействию.

Особое внимание уделено построению структуры дерева поведения, что обеспечивает четкую иерархию действий и состояний модуля, упрощая его управление и сопровождение. Представленная структура способствует модульности и повторному использованию компонентов, а также облегчает процесс отладки и тестирования.

В результате проделанной работы заложена прочная основа для дальнейшей реализации и интеграции программного модуля в общую систему автономной разгрузки, что повысит эффективность и надежность её функционирования.

3. РАЗРАБОТКА И ИНТЕГРАЦИЯ В ROS 2

Разработка программного модуля производилась на бортовом компьютере с предустановленной операционной системой Ubuntu 22.04 и использованием фреймворка ROS 2 Humble Hawksbill. В качестве основного языка программирования применялся C++. Вся система собиралась с помощью CMake [24] и управлялась через инструмент сборки colcon, который является стандартом в ROS 2. Для запуска и отладки использовались конфигурационные файлы.

3.1. СТРУКТУРА ПРОГРАММНОГО РЕШЕНИЯ

Программное решение, реализующее автономную разгрузку мини-погрузчика, построено с использованием архитектуры ROS 2 и библиотеки поведения BehaviorTree.CPP. Основная логика работы реализована в виде дерева поведения, описанного в XML-формате и исполняемого на уровне одного из ROS-узлов.

Дерево включает реактивные ветвления, повторяющиеся действия, восстановительные механизмы и последовательности задач, обеспечивая при этом гибкость, модульность и устойчивость к ошибкам при выполнении автономных операций.

На рисунке 14 представлена общая структура дерева поведения, определяющая последовательность действий, выполняемых погрузчиком при поиске ArUco-маркера, расчёте положения разгрузки, выполнении разгрузки и возврате в начальную точку.

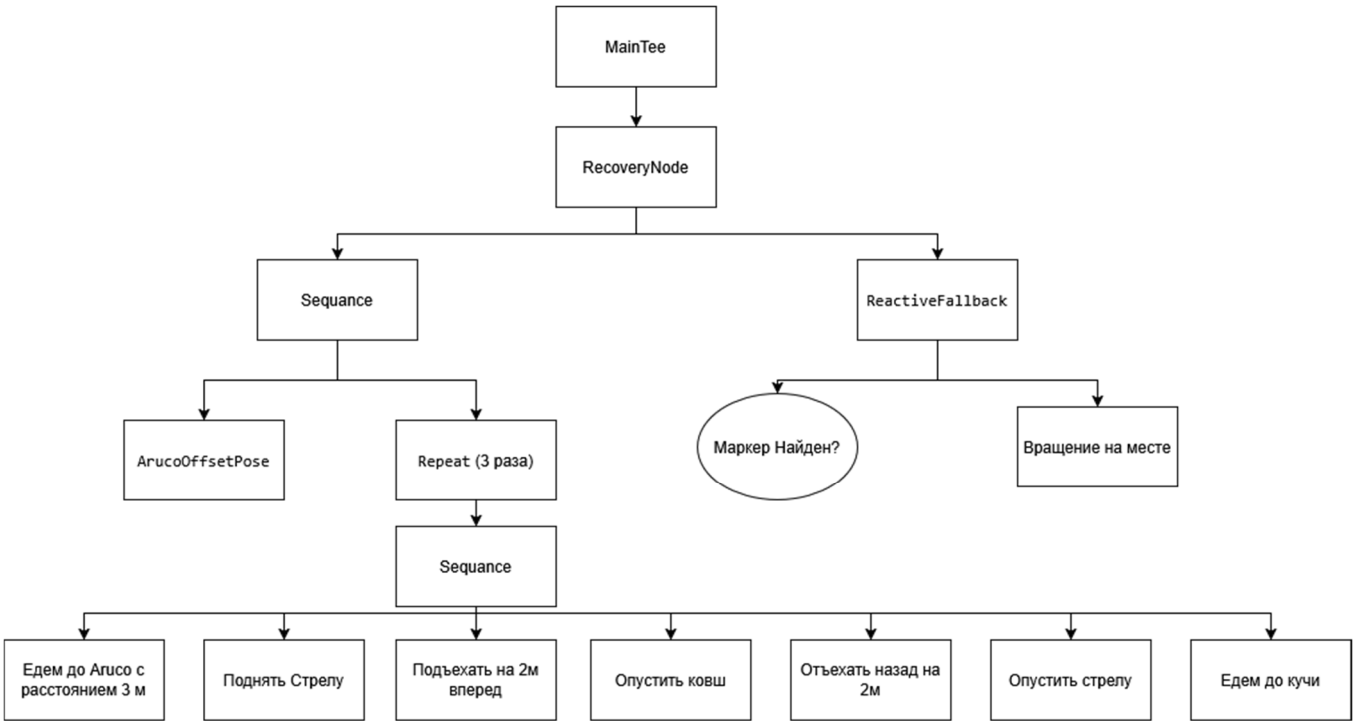

Рисунок 14 – Структура дерева поведения (условные обозначения по ГОСТ 19.701-90, ЕСПД)

Для визуализации деревьев поведения в промышленной практике применяется программа Groot [25] – редактор, разработанный автором BehaviorTree.CPP. Groot поддерживает экспорт в формат XML, совместимый с исполняемыми структурами в ROS 2. В данной работе Groot не использовался из-за отсутствия поддержки русскоязычного интерфейса и локализации, что затрудняет включение схем, оформленных в этой среде, в текст пояснительной записки.

Корнем дерева является узел «MainTree» внутри которого исполняется один основной управляющий блок – «RecoveryNode» [26]. Этот узел обеспечивает устойчивость к ошибкам: если один из дочерних элементов завершается неудачно, он может передать управление альтернативной логике. «RecoveryNode» содержит два дочерних узла:

- «sequence» – основная логика автономной разгрузки;
- «reactivefallback» – поведение при потере маркера или невозможности продолжить выполнение;

«RecoveryNode» завершает выполнение если первая ветка (Sequence) завершается успешно. Если же она возвращает FAILURE, активируется вторая ветка – «ReactiveFallback», в которой выполняются компенсирующие действия (например, вращение погрузчика для повторного поиска маркера).

Узел Sequence внутри «RecoveryNode» содержит 2 элемента: вычисление координат разгрузки относительно ArUco-маркера и основная последовательность разгрузки. Внутри узла «Sequence» находится вся логика реализации разгрузки.

Узел «ReactiveFallback», входящий в состав «RecoveryNode», представляет собой реактивное поведение, активируемое в случае неудачного выполнение основной логики. Его задача – обеспечить устойчивость системы при невозможности обнаружения ArUco-маркера или нарушений условий точной локализации.

Данный управляющий узел реализует логику проверки условия и при необходимости, переключения на альтернативное поведение. Он включает в себя 2 дочерних элемента:

- «FindAruco» – условное поведение, в котором осуществляется попытка обнаружения ArUco-маркера в текущем кадре с камеры;
- «Spin» – компенсирующее действие, заключающееся во вращении погрузчика на месте для расширения обзора и повышения вероятности обнаружения маркера.

Работа «ReactiveFallback» основана на следующей логике: сначала вызывается узел «FindAruco», который проверяет наличие маркера в текущем видеопотоке. Если маркер успешно обнаружен (SUCCESS), выполнение ветки завершается на этом этапе, и основная логика может быть перезапущена. Если же маркер не найден (FAILURE), активируется вторая ветка – Spin, выполняющая вращение техники с заданным углом, что позволяет изменить угол обзора камеры и повысить шансы на успешную детекцию при следующей попытке.

Такой подход обеспечивает реактивность системы и адаптацию к изменяющимся условиям среды, без необходимости ручного вмешательства со стороны оператора.

3.2. ИНТЕГРАЦИЯ с NAV2 И ИСПОЛЬЗОВАНИЕ СТАНДАРТНЫХ УЗЛОВ

Для реализации автономной навигации в проекте использован готовый навигационный стек Nav2. Дерево поведения, описанное в формате XML, передаёт управление различным узлам навигации. Эти узлы – часть Nav2 и работают в фоновом режиме как отдельные ROS-сервисы, к которым можно "обратиться" по имени. Вызов каждого действия в дереве поведения происходит через Action-сервер – это такой специальный механизм в ROS 2, который позволяет "запустить задачу" и дождаться результата.

1.2.1. Nav2: Планирование маршрута и следование по нему

«ComputePathToPose» [27] – планирование пути до цели. Этот узел отвечает за построение маршрута от текущего положения погрузчика до заданной точки (координаты разгрузки, базы и т.д.). Внутри он использует глобальный планировщик, выбранный через параметр planner_id. В данной работе используется «GridBased». Данный узел выполняет следующие функции:

- формирует маршрут на основе карты проходимости, представленной в виде двумерной сетки, где ячейки обозначают препятствия или проходимые зоны;
- применяет классические алгоритмы поиска пути, такие как Dijkstra или A*, в зависимости от настроек;
- возвращает список координат, через которые необходимо пройти для достижения цели с обходом препятствий.

Это аналог GPS-навигатора, но на уровне карты робота.

```
<ComputePathToPose goal="{goal_pose}" path="{path}" planner_id="GridBased"/>
```

Рисунок 15 – Пример вызова «ComputePathToPose» [27]

Обозначения:

- goal – координаты цели;
- path – переменная, куда сохраняется маршрут.

Следом за построением маршрута запускается узел «F ollowPath» [28], который берёт маршрут и управляет гусеницами, чтобы поехать по нему. Используется контроллер «FollowPath», который отслеживает текущую позицию робота по одометрии и корректирует движение, чтобы точно следовать по построенному маршруту.

```
<FollowPath path="{path}" controller_id="FollowPath"/>
```

Рисунок 16 – Пример вызова «FollowPath» [28]

1.2.2. Nav2: Прямолинейные движения

Иногда погрузчику необходимо выполнить фиксированное движение вперёд или назад, а не следовать заранее построенному маршруту. Например:

- аккуратно подъехать ближе к точке разгрузки;
- откатиться после сброса груза;
- вращаться на месте чтобы найти маркер.

Для этого используются стандартные команды, которые тоже реализованы в Nav2 как Action-узлы.

Узел «DriveOnHeading» [29] даёт команду ехать вперёд по текущему направлению на заданную дистанцию. Узел «BackUp» делает всё то же самое, только в обратную сторону – отъезжает назад на заданное расстояние. Используется, чтобы освободить зону разгрузки или безопасно отойти от цели. «Spin» – стандартный компонент, реализующий вращение робота на месте на

заданный угол. Это действие используется в рамках узла «ReactiveFallback», когда «FindAruco» не может обнаружить маркер в текущем изображении.

Таким образом, в рамках данной работы были задействованы ключевые стандартные action-узлы Nav2, обеспечивающие планирование маршрутов («ComputePathToPose»), управление движением («FollowPath»), а также выполнение простых прямолинейных манёвров («DriveOnHeading», «BackUp») и реактивных действий («Spin»). Благодаря гибкой архитектуре Nav2, все эти компоненты были встроены в поведенческое дерево и использованы без необходимости модификации исходного кода.

Однако, стандартных узлов оказалось недостаточно для решения прикладной задачи автономной разгрузки. Возникла необходимость в создании собственных компонентов, реализующих специализированную логику: вычисление позиции разгрузки, управление рабочим органом (ковшом) и генерация управляющих команд. Все эти действия реализованы в виде пользовательских action-узлов, которые также встроены в структуру дерева поведения.

3.3. РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКИХ УЗЛОВ

3.3.1. Узел «ArucoOffsetPose»

Узел «ArucoOffsetPose» реализует важную часть логики автономной разгрузки: он вычисляет точку разгрузки, смещённую от ArUco-маркера, на основании текущего положения погрузчика. Эта операция критически важна – если направить погрузчик точно на центр маркера, существует риск столкновения ковша с препятствием или окажется слишком близко к цели. Смещённая точка (output_pose) позволяет разгружаться аккуратно, при этом сохраняя точность позиционирования.

Назначение узла:

- определить текущую позицию погрузчика (current_pose);
- получить координаты ArUco-маркера (input_pose);

- вычислить точку разгрузки на расстоянии backoff_distance позади маркера;
- передать в BT выходные значения: новую цель (output_pose) и координаты возврата (goal_pose);

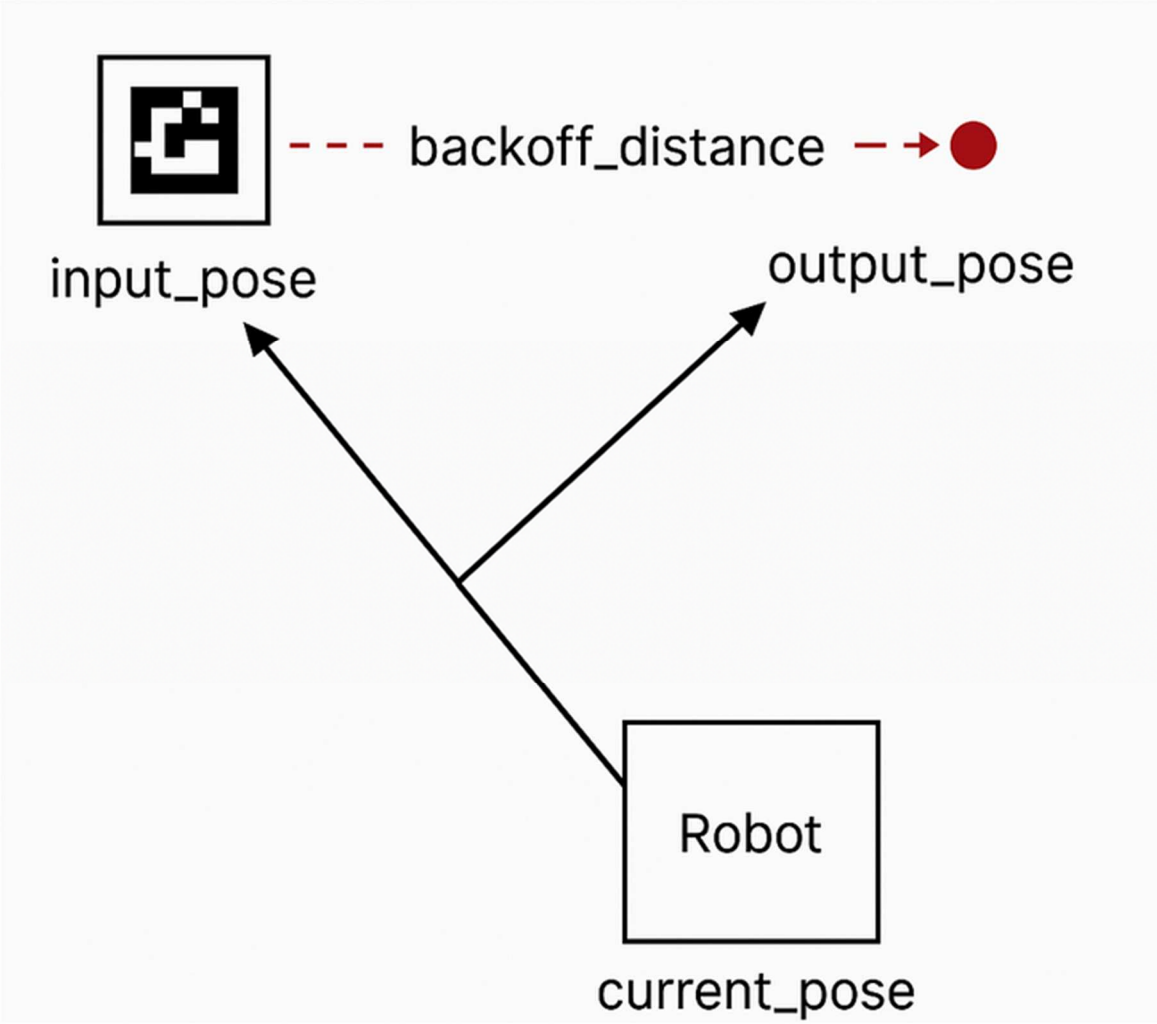

Рисунок 17 – Схема расчёта координаты разгрузки в узле «ArucoOffsetPose»

На рисунке 17 представлена визуализация работы узла. Робот находится в позиции current_pose и направлен в сторону маркера (input_pose). Узел ArucoOffsetPose рассчитывает точку output_pose, которая отстоит от маркера на заданное расстояние backoff_distance, и именно туда планируется дальнейшее перемещение.

3.3.2. Узел GenerateWorkingBodyCommand – генерация команд движения

«GenerateWorkingBodyCommand» формирует команды для привода ковша и стрелы. Этот узел собирает необходимые параметры положения «рабочего органа» и генерирует сообщение, содержащее последовательность действий для достижения требуемой ориентации ковша и стрелы. Генерируется две команды: начальная и завершающая. Такое разделение обеспечивает явное определение начала и конца рабочего цикла механизма, что облегчает управление сложными движениями и синхронизацию последовательности действий. В контексте ковша и стрелы это означает отдельное указание на начало подъема стрелы и ковша, а затем отдельную команду на остановку движения и фиксацию положения по достижении требуемого угла. Узел «GenerateWorkingBodyCommand» работает как планировщик команд: на основе текущего состояния и целевого задания он выдает структурированный набор действий для дальнейшего исполнения.

3.3.3. Узел MakeCommandAction – отправка команд через Action

Сформированные команды не выполняются напрямую самим Behavior Tree, вместо этого их исполнение делегируется системе управления приводами через механизм действий (Action) ROS 2. Узел «MakeCommandAction» отвечает за передачу сгенерированных команд исполнительным механизмам, используя Action-сервер. Action-сервер – это механизм в ROS 2, который позволяет отправить задачу на выполнение и дождаться результата.

При активации данный узел получает на вход сформированное сообщение (начальную или конечную команду в зависимости от этапа выполнения) и инициирует соответствующий Action-запрос. Выполнение команды происходит асинхронно на стороне Action-сервера.

Благодаря использованию шаблона BtActionNode, реализация узла «MakeCommandAction» сосредоточена преимущественно на формировании содержимого команды (указание типа действия и передаваемых данных), в то время как логика взаимодействия с Action-инфраструктурой (создание клиента, ожидание результата, обработка успешного или неуспешного завершения) обеспечивается встроенными средствами Behavior Tree.

Такой подход существенно упрощает интеграцию новых действий: для подключения дополнительного функционала достаточно зарегистрировать узел и определить структуру соответствующего Action-сообщения. Это обеспечивает масштабируемость и повторное использование компонентов при проектировании поведения системы.

3.3.4. Плагин SetWorkedBodyOrientation – управление ориентацией с PID-регулятором.

Низкоуровневое исполнение полученных команд осуществляется отдельным компонентом – классом «SetWorkedBodyOrientation». Он запускается на стороне Action-сервера при получении команды «make_command» от узла Behavior Tree. Основная задача SetWorkedBodyOrientation – управлять приводами ковша и стрелы таким образом, чтобы достичь требуемой ориентации, указанной в команде «InvokeActionArray», и затем стабильно её удерживать либо завершить движение.

Для реализации этого класса используется принцип обратной связи: «SetWorkedBodyOrientation» подписывается на топики датчиков, сообщающих текущее положение рабочего органа – например, углы наклона стрелы (arm_pitch) и ковша (bucket_pitch). Эти текущие значения постоянно сравниваются с заданными целевыми углами из команды. Разница между целевым и текущим значением (ошибка регулирования) служит входом для PID-регуляторов, встроенных в данный плагин.

PID-регуляторы вычисляют управляющие воздействия для плавного и точного достижения требуемого положения механизма. Управляющий сигнал ПИД-регулятора формируется как сумма трёх компонентов:

- пропорционального – зависит от текущей величины ошибки (разницы между заданным и фактическим положением);
- интегрального – учитывает накопленную ошибку за время, что позволяет устранять систематические смещения;
- дифференциального – реагирует на скорость изменения ошибки, сглаживая процесс и предотвращая перерегулирование.

В контексте узла «SetWorkedBodyOrientation» пропорциональная часть отвечает за реакцию на текущее отклонение угла ковша или стрелы от заданного значения. Интегральная часть компенсирует постоянные ошибки, возникающие при недостаточной коррекции пропорциональной составляющей. Дифференциальная часть помогает предсказать и сгладить изменения, предотвращая резкие колебания.

На основании суммы этих трёх компонентов PID-регулятор формирует управляющий сигнал, который подаётся на приводы для корректировки положения механизма.

Для непосредственного воздействия на приводы ковша и стрелы класс SetWorkedBodyOrientation публикует команды на специальный топик привода. Эти команды представляют собой управляющие скорости (линейные или угловые), влияющие на двигатели, поднимающие или опускающие стрелу и ковш. Таким образом, плагин постоянно в цикле получает текущие показания углов, рассчитывает необходимые поправки через PID и отправляет команду скорости приводам. Такой замкнутый цикл продолжается до тех пор, пока ковш и стрела не достигнут требуемой ориентации в пределах допустимой погрешности либо не истечёт заданный таймаут на выполнение. Когда целевое положение достигнуто (или если время вышло, предотвращая бесконечное ожидание), «SetWorkedBodyOrientation» прекращает подачу

команд – фактически останавливая двигатели. Завершающая команда, полученная на входе, может также предусматривать действия по удержанию положения или плавной остановке, что гарантирует безопасное завершение манёвра.

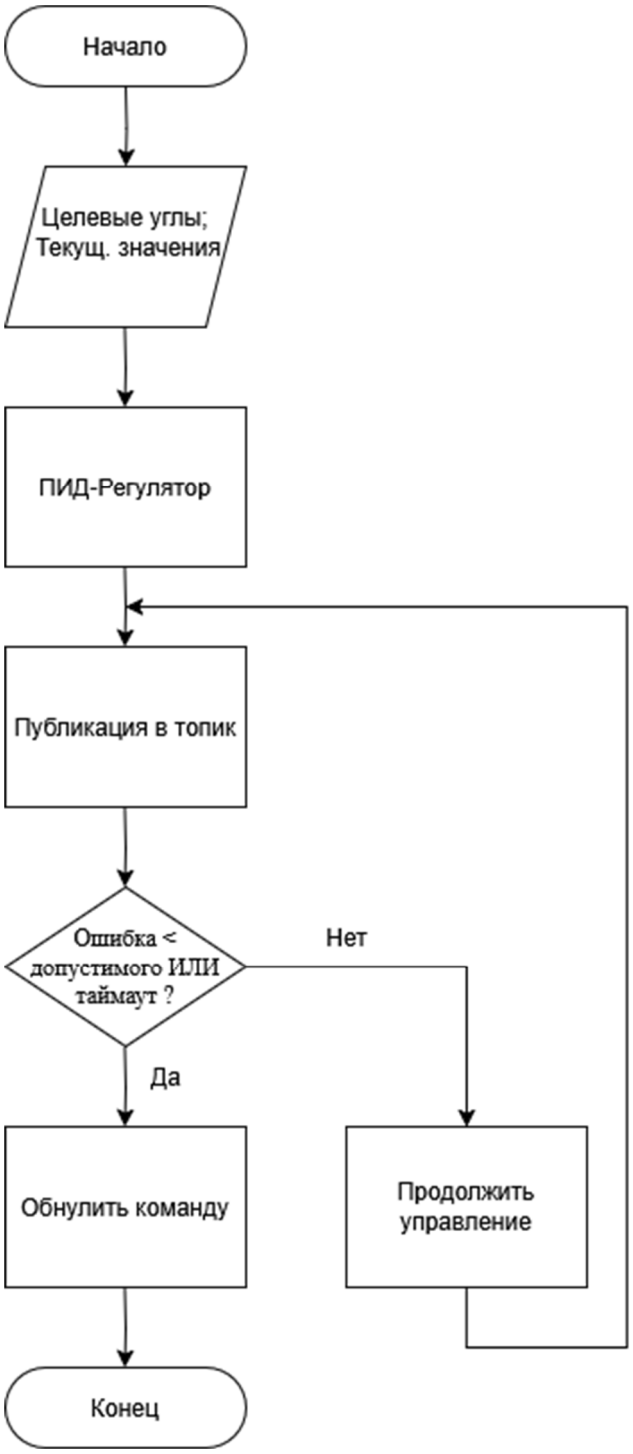

Рисунок 18 – Схема работы плагина SetWorkedBodyOrientation с использованием PID-регуляторов (ГОСТ 19.701-90. ЕСПД.)

На схеме (Рисунок 18) представлена логика работы пользовательского плагина «SetWorkedBodyOrientation», реализующего управление ориентацией рабочего органа (стрелы и ковша) с помощью ПИД-регулирования. Входные данные включают целевые и текущие значения углов наклона, получаемые через топики ROS. На основе ошибки (разницы между текущим и целевым значением) два отдельных PID-регулятора вычисляют управляющие сигналы, которые публикуются в топик для привода исполнительных механизмов. Завершение управления осуществляется при достижении заданной точности или истечении времени выполнения.

3.4. ВЫВОД ПО ГЛАВЕ ТРИ

В рамках главы была реализована программная часть системы автономной разгрузки мини-погрузчика. Основная логика управления построена с использованием платформы ROS 2 и библиотеки BehaviorTree.CPP, что обеспечило модульность, расширяемость и возможность реактивного поведения.

Для управления движением использован стандартный навигационный стек Nav2, включающий компоненты глобального планирования маршрутов и локального следования по пути. Поведенческое дерево позволяет координировать работу этих компонентов, а также переключаться на альтернативные сценарии при возникновении ошибок или отклонений от маршрута.

Важной частью реализации стала разработка пользовательских узлов, включая вычисление точки разгрузки относительно ArUco-маркера, генерацию управляющих команд для стрелы и ковша, а также их передачу и исполнение через Action-механизмы. Для точного позиционирования рабочих органов реализован плагин с ПИД-регуляцией, обеспечивающий плавное и устойчивое выполнение команд.

4. СИМУЛЯЦИЯ И ТЕСТИРОВАНИЕ СИСТЕМЫ

Целью тестирования являлась проверка работоспособности разработанного программного модуля автономной разгрузки мини-погрузчика, а также оценка корректности работы всех подсистем: локализации, навигации, управления разгрузкой и возвратом в исходную позицию.

Тестирование проводилось с виртуальной симуляционной среде на базе Unity с использованием физического движка «Algoryx» [30]. В качестве платформы использовался мини-погрузчик MT1 (Рисунок 19) производства ДСТ-Урал.

Рисунок 19 – MT1 от компании ДСТ-Урал

Бортовой компьютер: ноутбук Machenike L15, ОС Ubuntu 22.04, фреймворк ROS 2 Humble.

Проверялись следующие сценарии:

- обнаружение ArUco-маркера в сцене;
- построение и следование траектории к точке разгрузки;
- управление рабочими органами (ковшом и стрелой) для выполнения разгрузки;
- возврат на исходную точку после разгрузки;
- обработка ошибок и аварийная остановка;
- обработка потери маркера.

Перед запуском автономной разгрузки проводилась оценка корректности работы модуля локализации. Для этого использовался инструмент визуализации Rviz (Рисунок 20), в котором отображались глобальные и локальные координаты системы, траектория, рассчитанная системой навигации, ориентация и позиция погрузчика в реальном времени.

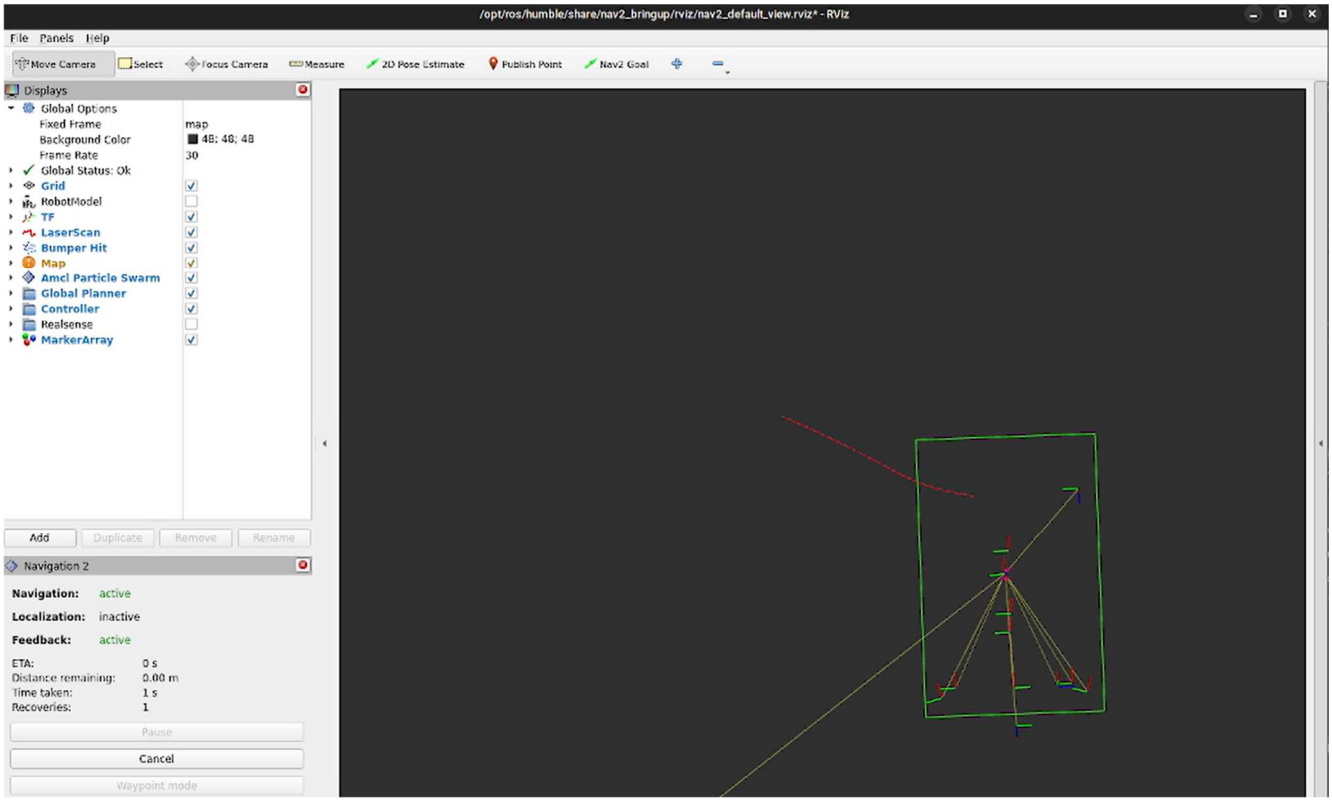

Рисунок 20 – Отображение результатов локализации и навигации в RViz

«На рисунке 20 видно: зелёная рамка – это положение и ориентация мини-погрузчика в пространстве, красная линия – это траектория, рассчитанная навигационным стеком Nav2. Работа фильтра robot_localization

показала стабильные и точные данные по позиции. Ось направления совпадала с реальной ориентацией погрузчика в сцене, а траектория не содержала разрывов.

После инициализации локализации погрузчик активировал подсистему визуального позиционирования для поиска маркера. Система проверяла, виден ли маркер в текущем кадре (камеры). Если маркер находился – вычислялись координаты цели, и погрузчик переходил к действию. В случае отсутствия маркера в поле зрения активировался режим реактивного сканирования – робот начинал медленно вращаться на месте, чтобы изменить угол обзора и повторить попытку обнаружения.

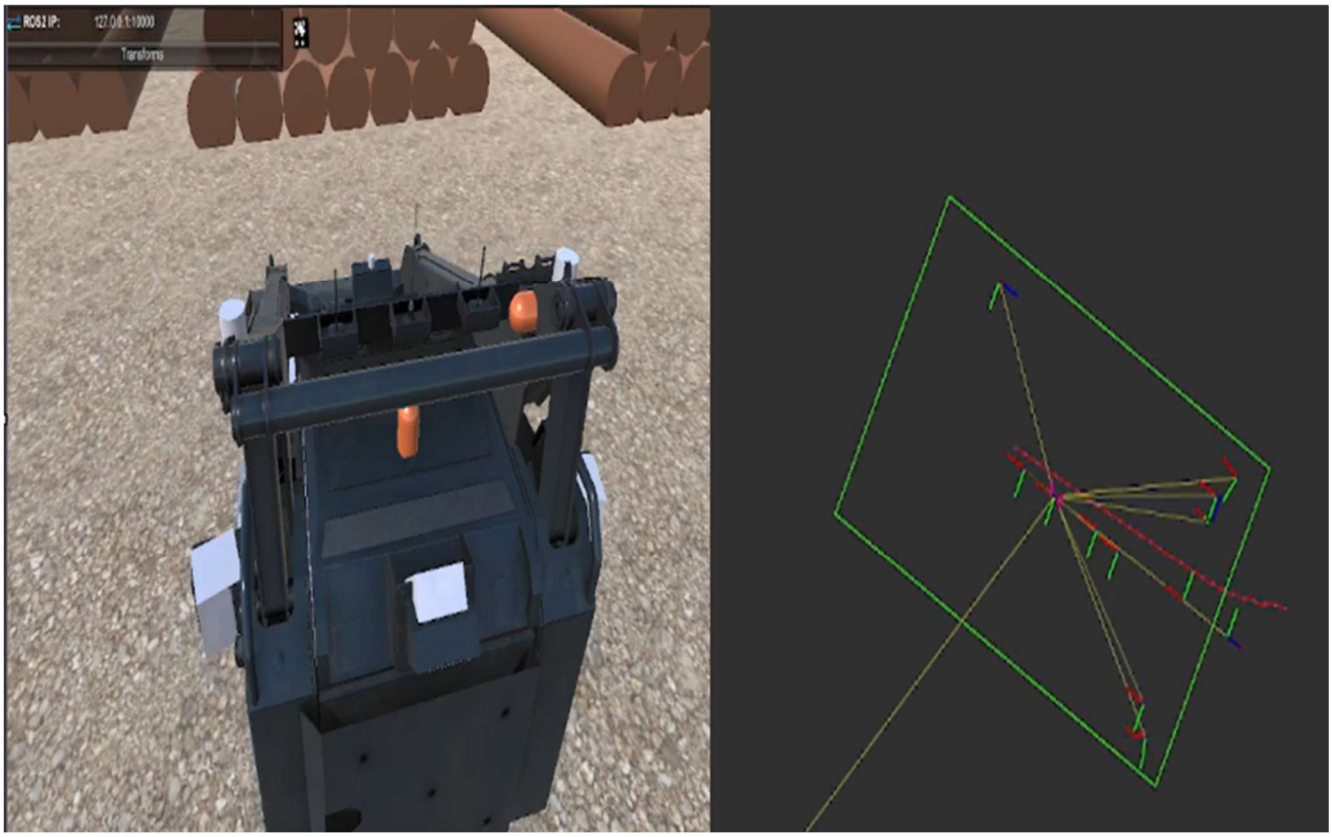

Рисунок 21 – Обнаружение ArUco-маркера и планирование пути к целевой точке

Слева представлена сцена симуляции в Unity, где камера на крыше мини-погрузчика захватывает поле зрения с маркером. Справа – визуализация в Rviz.

Как видим, после обнаружения маркера навигационный стек построил путь до целевой точки (красная линия) и начал движение в сторону ArUco.

На точке разгрузки погрузчик активировал управление ковшом и стрелой для выгрузки материала в самосвал (Рисунок 22).

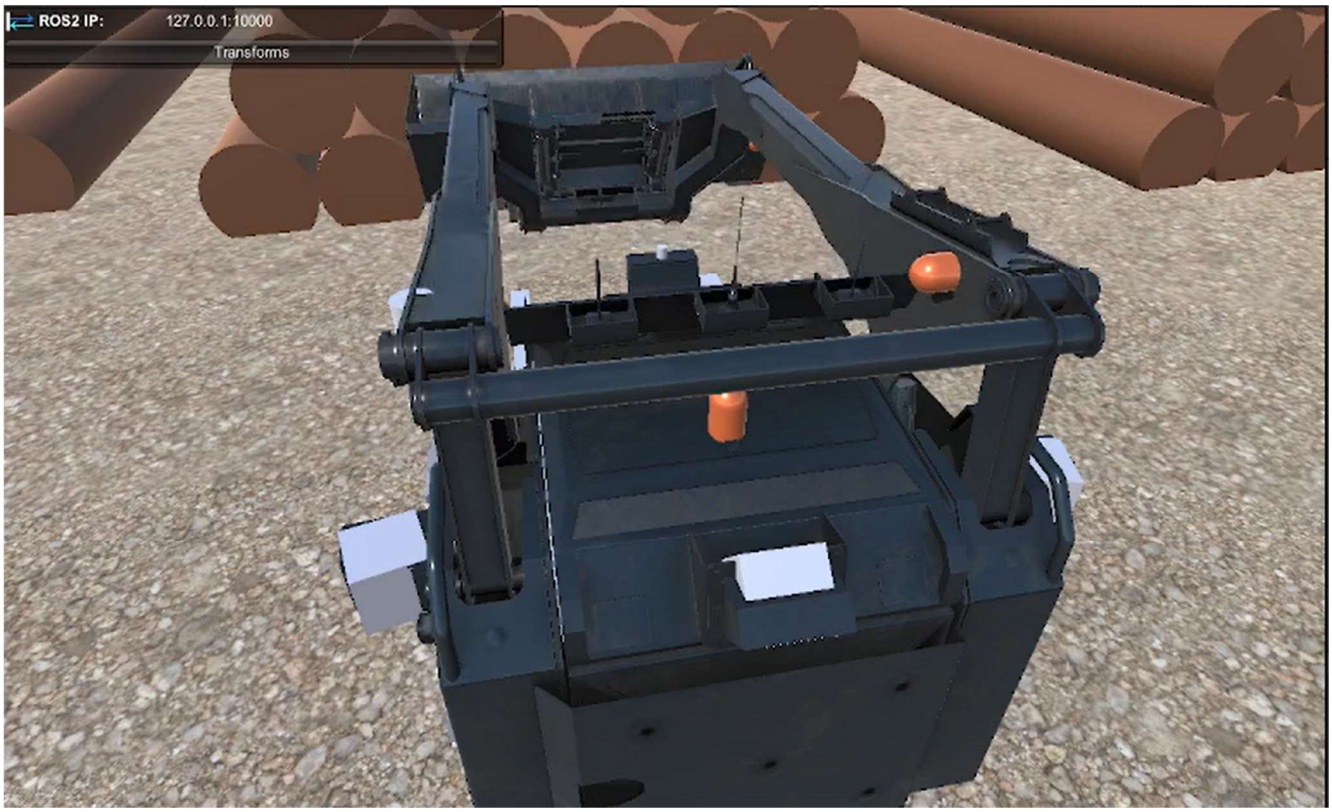

Рисунок 22 – поднятие стрелы и ковша

После завершения разгрузки мини-погрузчик строил обратный маршрут и возвращался на стартовую позицию.

Если в момент запуска программного модуля маркер отсутствовал в поле зрения камеры, система автоматически переходила в режим активного поиска. Для этого активировался узел «Spin» из навигационного стека Nav2, который инициировал медленное вращение погрузчика на месте. Камера непрерывно сканировала изображение на наличие маркера. Как только маркер появлялся в кадре, система фиксировала его координаты, прекращала вращение и переходила к следующему действию – приближению к точке разгрузки.

Данный механизм позволил исключить ситуации, при которых робот не мог начать выполнение задачи из-за начального положения вне прямой видимости цели.

4.1. ВЫВОД ПО ГЛАВЕ ЧЕТЫРЕ

Разработанный программный модуль продемонстрировал стабильную и надёжную работу в условиях виртуальной симуляции. Поведенческая логика, реализованная с использованием библиотеки BehaviorTree.CPP, обеспечила эффективное и последовательное выполнение всех необходимых действий, что позволило добиться высокого уровня контроля над процессом.

Обнаружение ArUco-маркеров происходило корректно и своевременно, что служило основой для точного позиционирования. Планирование траекторий движения осуществлялось без каких-либо сбоев или задержек, что подтверждает качество алгоритмов навигации, внедрённых в систему. Управление рабочими органами выполнялось с допустимой погрешностью, что гарантировало безопасность и стабильность выполнения операций.

Таким образом, проведённые тесты и испытания позволяют с уверенностью утверждать, что разработанное программное решение полностью соответствует предъявленным требованиям и может быть рекомендовано для дальнейшего использования и внедрения в реальных условиях.

5. ЗАКЛЮЧЕНИЕ

В рамках выпускной работы разработана и протестирована программная система для автономной разгрузки мини-погрузчика с использованием визуальных маркеров, дерева поведения и ROS 2, направленная на повышение автоматизации погрузочно-разгрузочных операций.

В первой главе проведён анализ предметной области, рассмотрены решения ведущих производителей и ключевые технологии автономного позиционирования, включая ArUco-маркеры, визуальную и лидарную одометрию, а также структуру Behavior Tree.

Во второй главе определены требования и описана архитектура системы с выбором ROS 2, стека Nav2 и иерархичной модели управления на основе дерева поведения.

Третья глава посвящена реализации компонентов системы, включая управление ковшом и стрелой через Action-интерфейсы и PID-регулирование для точных и плавных манёвров.

В четвёртой главе описано тестирование в симуляторе Unity, где успешно проверены локализация, построение маршрута, разгрузка и возврат, продемонстрирована устойчивость и соответствие требованиям.

Таким образом, цели достигнуты, и разработанный модуль может служить основой для внедрения автономных решений в строительной и горнодобывающей технике, а также для дальнейших исследований в мобильной робототехнике.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ничего себе тест-драйв. Беспилотным погрузчиком БЕЛАЗ управляли на дистанции 2500 километров – https://abw.by/news/commercial/2018/07/11/nichego-sebe-testdraiv-bespilotnym-pogruzchikom-belaz-upravlyali-na-distancii-2500-kilometrov.
2. Volvo Construction Equipment. LX03 автономный электрический погрузчик нового поколения. – https://www.volvoce.com/global/en/about-us/what-we-believe-in/innovation-at-our-core/our-innovation-concepts/lx03/.
3. Прокопьева, В.М. Обзор роботизированной техники в горном деле // В.М. Прокопьева, М.В. Каймонов. – Интерактивная наука. – 2023. – № 8 (84). – С. 49-53. – https://cyberleninka.ru/article/n/obzor-robotizirovannoy-tehniki-v-gornom-dele/viewer.
4. Xu, J.S. Robust stereo visual odometry for autonomous rover // J.S. Xu. – Proceedings of the 6th WSEAS International Conference on Signal, Speech and Image Processing. – 2006. – P. 123-130.
5. Nistér, D.N. Visual odometry for ground vehicle applications // D.N. Nistér. – Journal of Field Robotics. – 2006. – Vol. 23, Iss. 1. – P. 3–20.
6. Beets E.B. Aircraft Pose Estimation from Homography. // E.B. Beets. – Technical report MECSE-1-2004. Monash University, 2004.
7. Девятериков, Е.А. Визуальный одометр // Е.А. Девятериков, Б.Б. Михайлов. – Инженерный журнал: наука и инновации. – 2012. – № 6 (6). – https://cyberleninka.ru/article/n/vizualnyy-odometr.
8. Лидары. – https://uav-bpla.com/unmanned-vehicles-avto/lidars/.
9. Мезенцева А.Д. Алгоритм обнаружения внезапно возникающего препятствия по показаниям LIDAR // А.Д. Мезецнева. – Научно-практический электронный журнал «Аллея Науки». – 2018. – № 1(17). – https://alley-science.ru/domains_data/files/12January/ALGORITM OBNARUZhENIYa

VNEZAPNO VOZNIKAYuShEGO PREPYaTSTVIYa PO POKAZANIYaM LIDAR.pdf?clckid=bbcecdb6.

10. Liu, W.C. LiDAR odometry survey: recent advancements and remaining challenges // W.C. Liu. – ResearchGate. – 2024. – https://www.researchgate.net/publication/378106065_LiDAR_odometry_survey_recent_advancements_and_remaining_challenges.
11. Понятский В.М. Применение фильтра Калмана для задач управления подвижными объектами // В.М. Понятский, Б.В. Зенов. – Современные информационные технологии и ИТ-образование. 2018. №3. – https://cyberleninka.ru/article/n/primenenie-filtra-kalmana-dlya-zadach-upravleniya-podvizhnymi-obektami (дата обращения: 23.05.2025).
12. Ahmad, N. Reviews on various inertial measurement unit (IMU) sensor applications // N. Ahmad. – International Journal of Signal Processing Systems. – 2013. – Vol. 1, No. 2. – P. 256–262.
13. Как работает GPS. – https://skillbox.ru/media/.
14. Локализация по Aruco маркерам. – https://habr.com/ru/articles/482220/.
15. robot_localization – ROS-пакет для локализации робота – https://github.com/cra-ros-pkg/robot_localization.
16. Шереужев, М.А. Моделирование группового управления сельскохозяйственными роботами с использованием конечных автоматов и онтологий // М.А. Шереужев, Ф.В. Девяткин, Д.И. Арабаджиев, М.А. Шереужев. – Известия КБНЦ РАН. – 2023. – № 6 (116). – https://cyberleninka.ru/article/n/modelirovanie-gruppovogo-upravleniya-selskohozyaystvennymi-robotami-s-ispolzovaniem-konechnyh-avtomatov-i-ontologiy.
17. Wang, X. Deep reinforcement learning-aided autonomous navigation with landmark generators // X. Wang. – Frontiers in Neurorobotics. – 2023. – Vol. 17. – Art. no. 1200214. – P. 10–25.
18. Обучение с подкреплением для самых маленьких – https://habr.com/ru/articles/308094/.

19. Документация ROS 2: Humble Hawksbill – https://docs.ros.org/en/humble/index.html.
20. The Robot Operating System (ROS1 & 2): Programming paradigms and deployment – https://www.researchgate.net/publication/363842243_The_Robot_Operating_System_ROS1_2_Programming_Paradigms_and_Deployment.
21. BehaviorTree.CPP – официальный сайт библиотеки поведения – https://www.behaviortree.dev/.
22. Nav2 – документация по навигационному стеку ROS 2 – https://docs.nav2.org/.
23. Nav2 – руководство по созданию пользовательского Behavior-плагина – https://docs.nav2.org/plugin_tutorials/docs/writing_new_behavior_plugin.html.
24. CMake – официальный сайт – https://cmake.org/.
25. Groot – редактор деревьев поведения – https://github.com/BehaviorTree/Groot.
26. RecoveryNode – документация Nav2 – https://docs.nav2.org/configuration/packages/btplugins/controls/RecoveryNode.html.
27. ComputePathToPose – документация Nav2 – https://docs.nav2.org/configuration/packages/btplugins/actions/ComputePathToPose.html.
28. FollowPath – документация Nav2– https://docs.nav2.org/configuration/packages/bt-plugins/actions/FollowPath.html.
29. DriveOnHeading – документация Nav2– https://docs.nav2.org/configuration/packages/bt-plugins/actions/DriveOnHeading.html.
30. AGX Unity – физический движок для симуляции в Unity– https://www.algoryx.se/agx-unity/.