Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Направление подготовки: 09.04.04 – Программная инженерия

Магистерская программа: Робототехника

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Моделирование виртуальных гусениц робота Сервосила Инженер при помощи зубчатых колес в симуляторе Gazebo

Обучающийся 2 курса

группы 11-031 Габдрахманов Р.Т.

Научный руководитель

PhD (технические науки), доцент, профессор
кафедры интеллектуальной робототехники Магид Е.А.

Директор ИТИС КФУ

канд. техн. наук Абрамский М.М.

Казань – 2022

Оглавление

ВВЕДЕНИЕ

В процессе исследований часто возникает необходимость проверки новых теорий и разработок не путем экспериментов и испытаний в реальном мире, а в симуляциях. Причиной этого может быть опасность эксперимента, его дороговизна, сложность воссоздания его условий в реальном мире или вопросы законности или этичности воспроизводства условий эксперимента. Хотя симуляции часто не соответствуют реальному миру в полной степени, их полезность уже признана во многих областях науки, и робототехника является одной из областей, где симуляции наиболее востребованы.

Симуляция (или моделирование) в робототехнике позволяет создать виртуальную модель робота и среду, в которой он выполняет различные задачи [50, 46]. Поведение виртуальной модели, экспериментальная установка и характеристики сцены виртуального эксперимента обычно пытаются максимально точно приблизить реального робота к реальной среде [4]. Моделирование используется во многих областях робототехники, включая проектирование и конструирование роботов, тестирование кинематики и динамики, разработку алгоритмов управления, предварительную демонстрацию работы, обучение операторов и т.д. [56].

Моделирование в робототехнике широко используется по многим причинам, и в процессе работы было выделено только три наиболее популярных из них. Замеченные преимущества и недостатки этих методов послужили мотивацией и движущим фактором для исследования, представленного в этой работе:

1. Цена ошибки при работе с дорогим роботом очень высока, поэтому новые концепции, поведенческие алгоритмы и модули системы управления изначально создаются и тестируются в рамках симуляции, где любые повреждения и проблемы полностью виртуальны [44, 28].

2. Моделирование позволяет легко воспроизводить и повторять одни и те же эксперименты несколько раз, что сложно сделать в реальных условиях. Эта функция крайне важна на этапе отладки [55].
3. Моделирование может сэкономить исследователю существенное количество времени, так как настройка новой виртуальной среды или модернизация модели робота занимают значительно меньше времени, чем подготовка реальной среды и сборка реального робота [17, 26, 28].

Как правило, гусеничные роботы имеют намного более высокий уровень проходимости по пересечённой местности в сравнении с колесными роботами аналогичного размера и мощности, но существенно уступают по скорости, энергоэффективности на плоской поверхности и простоте изготовления [24].

В этой работе описывается новый метод моделирования гусениц мобильного робота, который представляет собой модификацию оригинальной модели из работы [31], в которой используются несколько стандартных круглых колес вместо настоящих цепных гусениц.

Новизна метода заключается в использовании в модели пересекающихся колес в виде шестерен, что позволяет приближенно имитировать выступы на реальных гусеницах, повышая проходимость и, при определенных конфигурациях, нивелируя наиболее критические различия в поведении шасси относительно шасси реального робота.

Целью метода является устранение наиболее критических недостатков метода моделирования ходовой части с использованием нескольких колес без существенного ухудшения характеристик. В качестве тестового стенда для имплементации и тестирования нового метода используется робот «Сервосила Инженер» и его модель.

Задачи исследования:

1. Изучить актуальные работы, связанные с моделированием гусеничных роботов.
2. Реализовать новый метод моделирования ходовой части гусеничного робота на модели робота «Сервосила Инженер». Достичь как можно более близких к реальному роботу показателей проходимости.
3. Создать воспроизводимую модель среды поисково-спасательных операций в городской среде, используя подход RSE.
4. Провести испытания модели робота «Сервосила Инженер» с ходовой частью, сделанной по новому методу. Убедиться в улучшении соответствия характеристик проходимости модели реальному роботу.

Актуальность предложенного решения обоснована тем, что созданная нами модель более близка к реальному роботу по динамике движения и позволит проводить первые этапы разработки и тестирования других прикладных и научных программных решений, что позволит сократить время работы, вести разработку и тестирование нескольким разработчикам параллельно (даже если физический робот всего один). Особенно полезно это для поисково-спасательных операций в городской среде, поскольку, во-первых, среду этой задачи может быть затруднительно воспроизвести в реальности на приемлемом расстоянии от места разработки (нужны настоящие развалины здания), во-вторых, робот для выполнения этой задачи обязательно должен уметь преодолевать препятствия, а не просто объезжать их [19]. В дополнение, если различия в динамике движения будут малы, возможно будет проводить машинное обучение на модели, что позволит достичь результатов за обозримые сроки, поскольку для обучения, к примеру, нейросети даже несложной задаче требуется несколько сотен обучающих прогонов и сравнимое количество тестовых прогонов [52].

1. Обзор литературы

Симуляции признаются во множестве работ эффективным решением для начальных этапов разработки и тестирования. В то же время многие отмечают, что поведение модели и реального робота могут отличаться, что может привести к проблемам, когда программное обеспечение для робота, разработанное и испытанное в симуляции, начинают тестировать в реальном мире [2, 33, 35].

1.1. Существующие подходы

На момент написания этой работы существовало несколько методов моделирования шасси гусеничных роботов. Эти методы можно разделить на три основные группы: прямое программное ускорение всей модели, подход на основе использования множества стандартных круглых колес и подход с применением сегментных гусениц. Каждый подход имеет определенные преимущества и недостатки, которые описаны в этом разделе.

1.1.1. Прямое программное ускорение

Грубая сила ускорения прикладывается непосредственно к шасси робота или к гусеницам робота, при этом подвижная часть шасси (гусеницы) моделируется с помощью полозьев или пассивных колес (Рис. 1). Этот метод является наиболее производительным с точки зрения использования процессора и памяти, и обеспечивает лучший RTF (Real Time Factor, рус. Фактор Реального Времени), и его легко реализовать. К сожалению, динамика движения такой модели существенно отличается от реального физического робота [34].

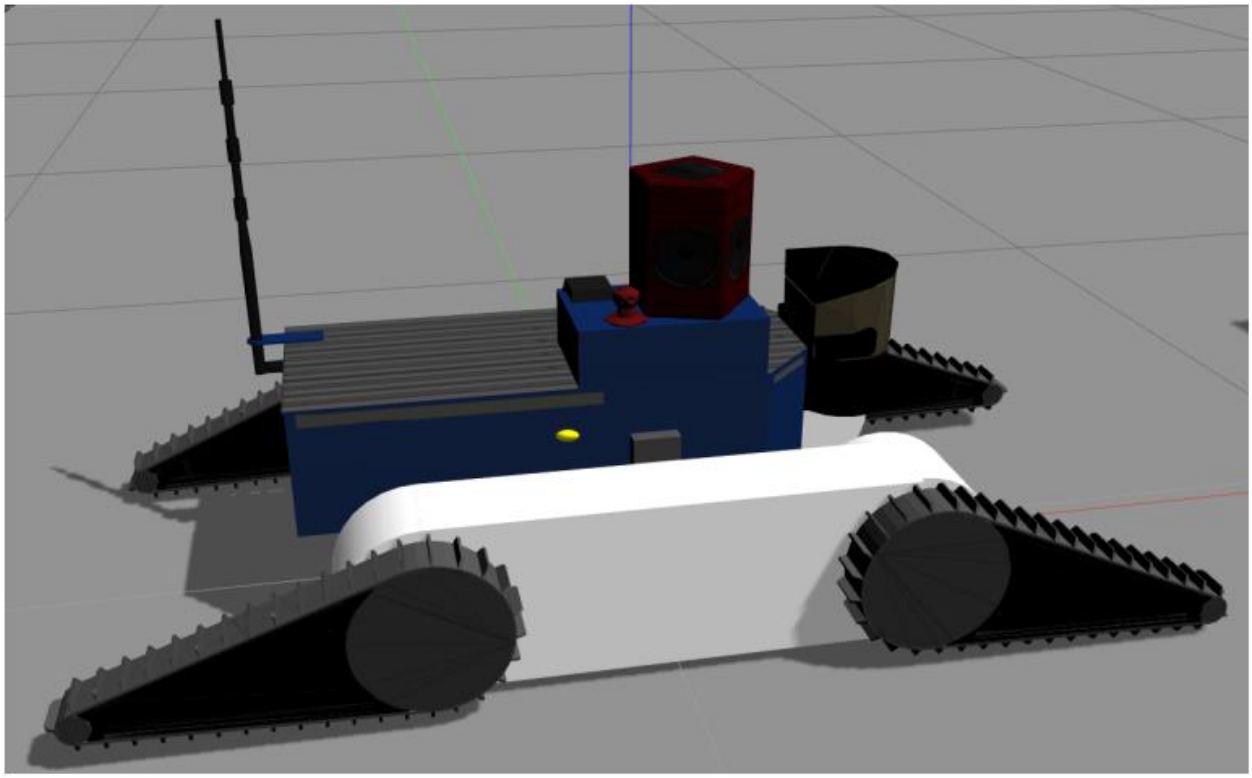


Рис. 1 Полозья, имитирующие гусеницы программным ускорением [34].

Самый простой и быстрый способ реализовать модель работающего робота. Возможно добиться реалистичного поведения на плоской поверхности, но в других случаях поведение модели может сильно отличаться от такового у реального робота [34].

1.1.2. Применение множества колёс

В этом подходе используются стандартные круглые колеса [43], чтобы симулировать гусеницы реального робота. Колеса могли быть видны, когда они полностью заменяли резиновую или металлическую гусеницу, или могли быть представлены в виртуальном виде. В виртуальной форме пользователь видит визуализацию гусеницы, а колеса, часто называемые псевдоколесами, отвечают за физическое взаимодействие с окружающей средой. В обоих случаях колеса расположены таким образом, что позволяют повторять форму и направление движущего импульса гусениц реального робота. Более того, колеса могут частично перекрываться друг другом, при этом физика таких столкновений между колесами игнорируется. В процессе исследований было выделено два наиболее популярных подхода на основе колес: один ряд

больших круглых колес (далее именуемый «линейный подход» - англ. LWA, Linear Wheels Approach) и большое количество круглых колес, распределенных вдоль гусеницы (далее именуемых «периметральный подход» - англ. CWA, Circumferential Wheels Approach).

1.1.2.1. Линейный подход

Линейный подход формирует структуру, в которой все колеса расположены на одной прямой линии. Это стандартные фиксированные колеса диаметром, равным высоте гусеницы. Колеса могут виртуально перекрывать друг друга без учета физики столкновения между колесами (Рис. 2) или располагаться без физического пересечения (Рис. 3 [31]), (Рис 4 [34]).

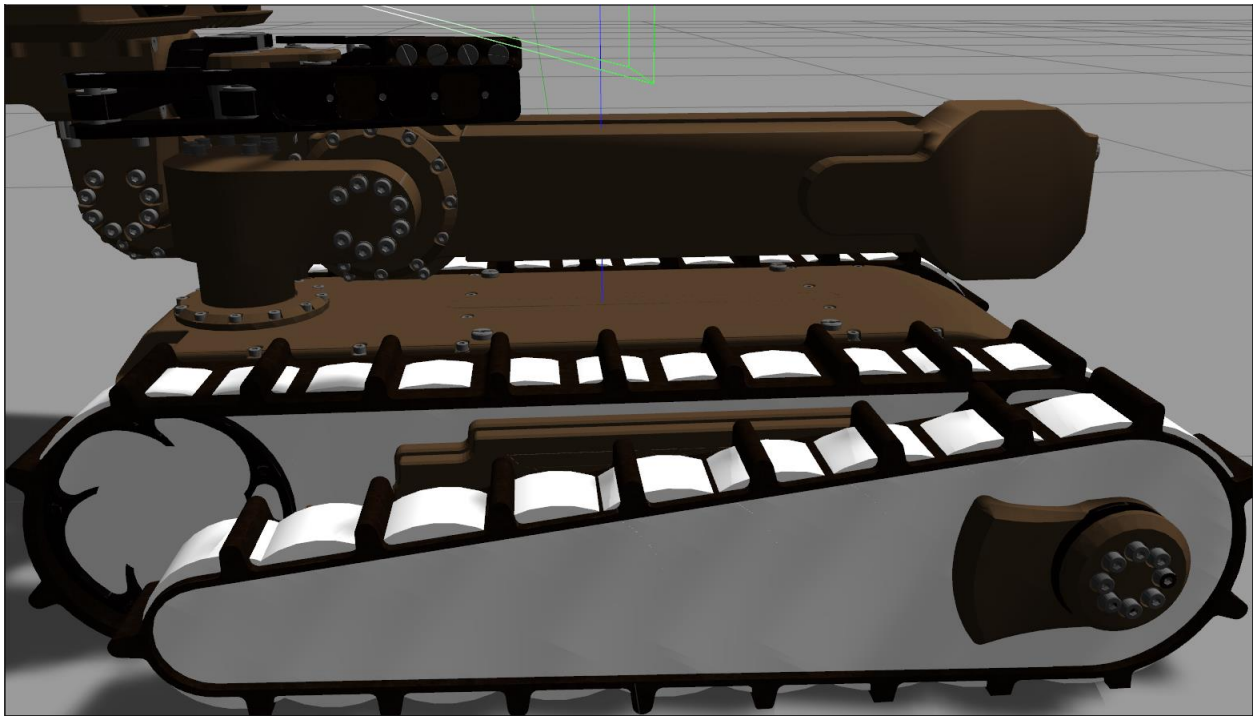


Рис. 2 Линейный подход с большими виртуальными самопересекающимися колесами (одна из дополнительных моделей, построенных в процессе работы)

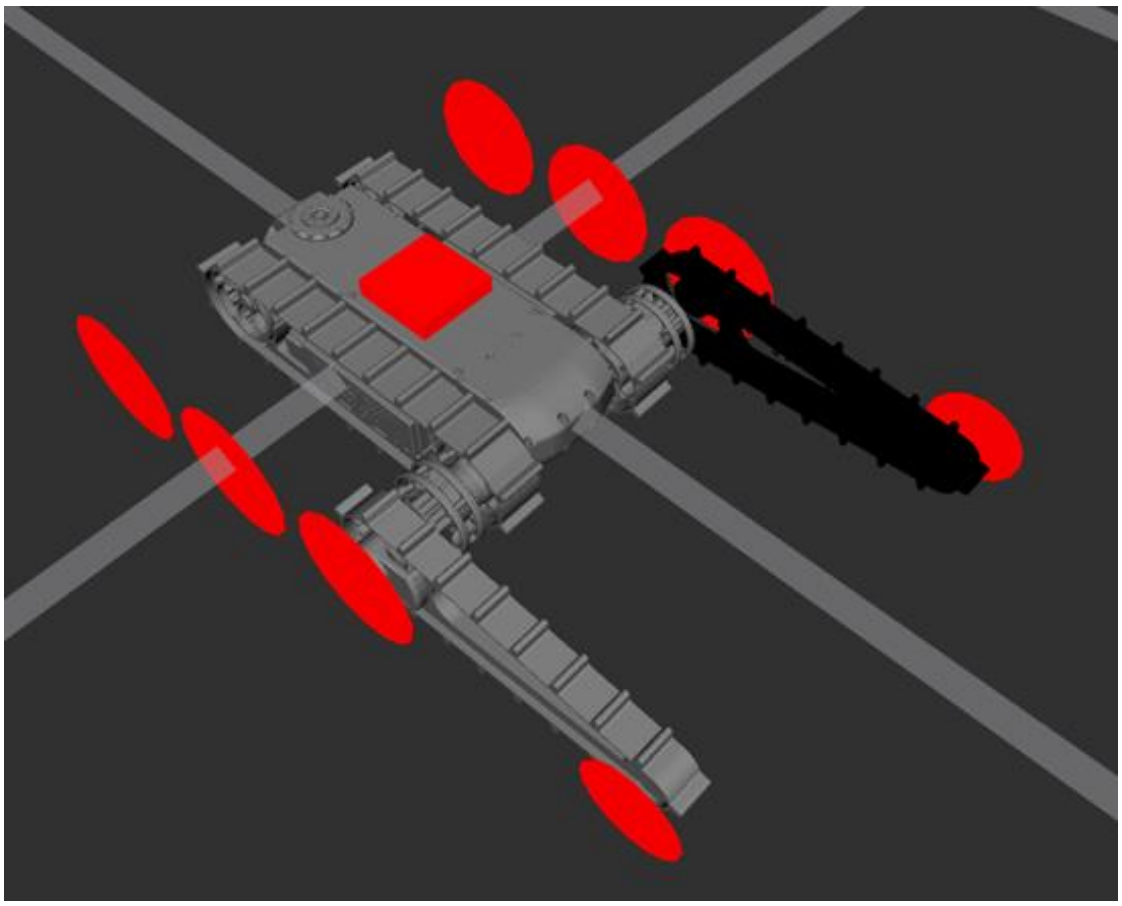


Рис. 3 Модель робота с колёсами, расположенными линейно на значительном расстоянии [31]

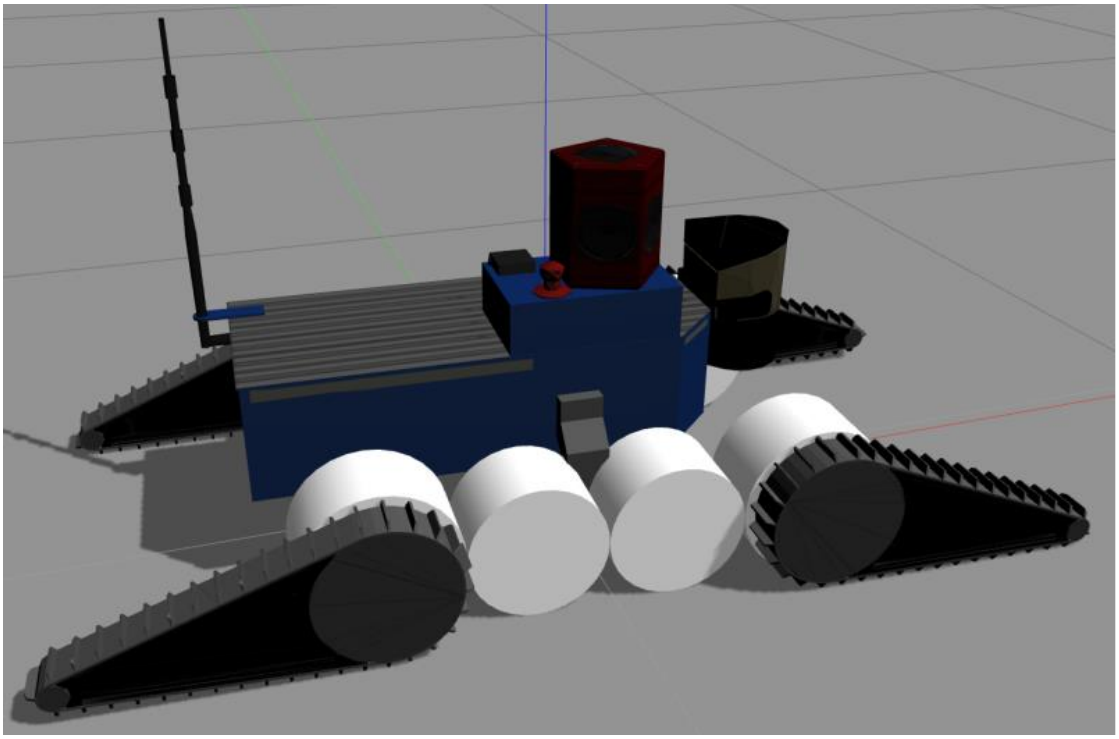


Рис. 4 Модель робота с колёсами, расположенными линейно с нулевым расстоянием между колёсами [34]

Линейный подход, особенно с небольшим количеством колес большого размера, выглядят привлекательно с точки зрения производительности моделирования (нагрузка центрального процессора (ЦП), память, RTF) и сложности разработки. Но эти подходы подходят только для плоской поверхности, так как робот может неестественно застрять при подъеме на препятствие, если острый выступ препятствия попадет между колесами [54] (Рис. 5). Увеличение количества пересекающихся колес частично решает проблему [34], однако большее количество колес неизбежно снижает производительность, в то время как робот все еще может застрять аналогичным образом, особенно если он имеет большую массу.

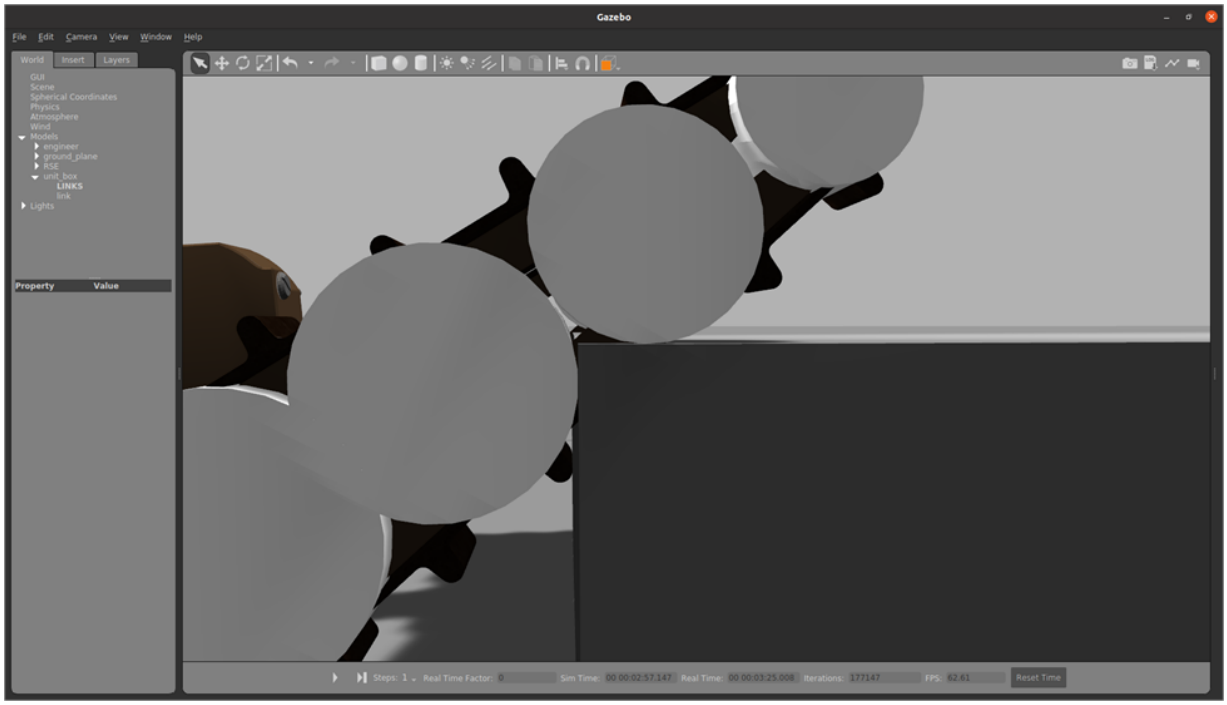


Рис. 5 Робот с колесами с линейным подходом застревает на остром углу препятствия.

1.1.2.2. Периметральный подход

Данный подход предполагает использование большого количества колес, расположенных по периметру каждой гусеницы. Колеса обычно имеют одинаковый радиус, который значительно меньше высоты гусениц (Рис. 6). Такая реализация обеспечивает большее сходство с точки зрения геометрии

без пересечения колес, но страдает низкой производительностью моделирования, связанной с большим количеством колес.

Также стоит упомянуть гибридные версии, в которых для имитации гусеницы сочетаются окружные и линейные подходы (Рис. 7).

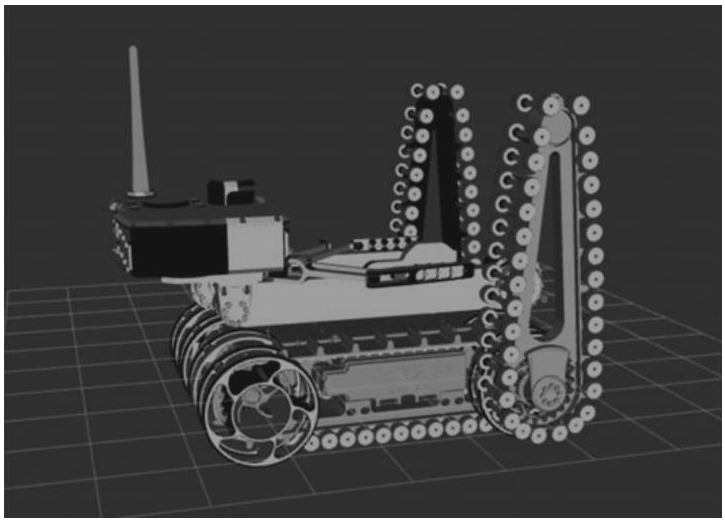


Рис. 6 Периметральный подход: Малые колёса с пространством между ними [31]

Рис. 7 а-б Гусеничный робот Talon (слева) и его модель в симуляторе USAR-Sim (справа) [35]

Несмотря на меньший размер колёс, данная модель также может застрять на острых выступах препятствия (Рис. 8)

Рис. 8 Робот с периметральным подходом застревает на остром углу препятствия.

1.1.3. Сегментарные гусеницы

В этом подходе гусеница собирается из сегментов прямоугольной или более сложной формы, которые соединяются в единую цепь пассивными связями. Сегменты растягиваются между как минимум двумя активными роликами с передней и задней кромки шасси и удерживаются там с помощью физики моделирования или программно (Рис. 9, 10). Могут использоваться дополнительные промежуточные пассивные или активные ролики. Теоретически, такой подход наиболее надежен, так как имеет почти абсолютное геометрическое подобие, но, как правило, имеет низкую производительность [30].

С учетом трех подходов, описанных в этом разделе, сегментные гусеницы кажутся наиболее привлекательными для построения модели гусеничного робота с точки зрения сходства с реальным роботом, поскольку геометрически такая реализация наиболее близка к реальным гусеницам.

Однако низкая вычислительная эффективность и низкая устойчивость этого метода делают его неудовлетворительным на практике [53, 21].

Более того, робот с сегментными гусеницами также может застрять, если в зазор между сегментами гусеницы попадет достаточно острое или небольшое препятствие.

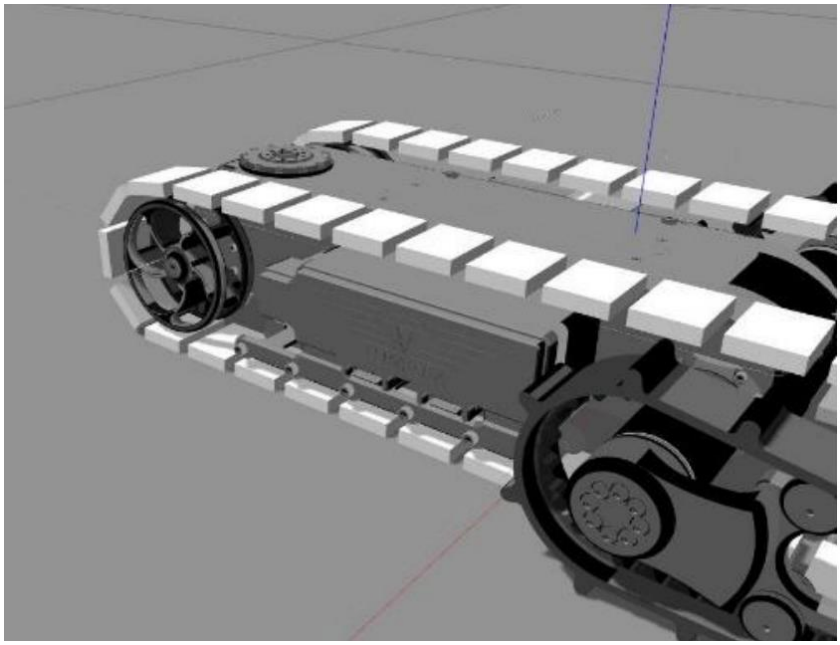


Рис. 9 Одна из имплементаций сегментарных гусениц [53]

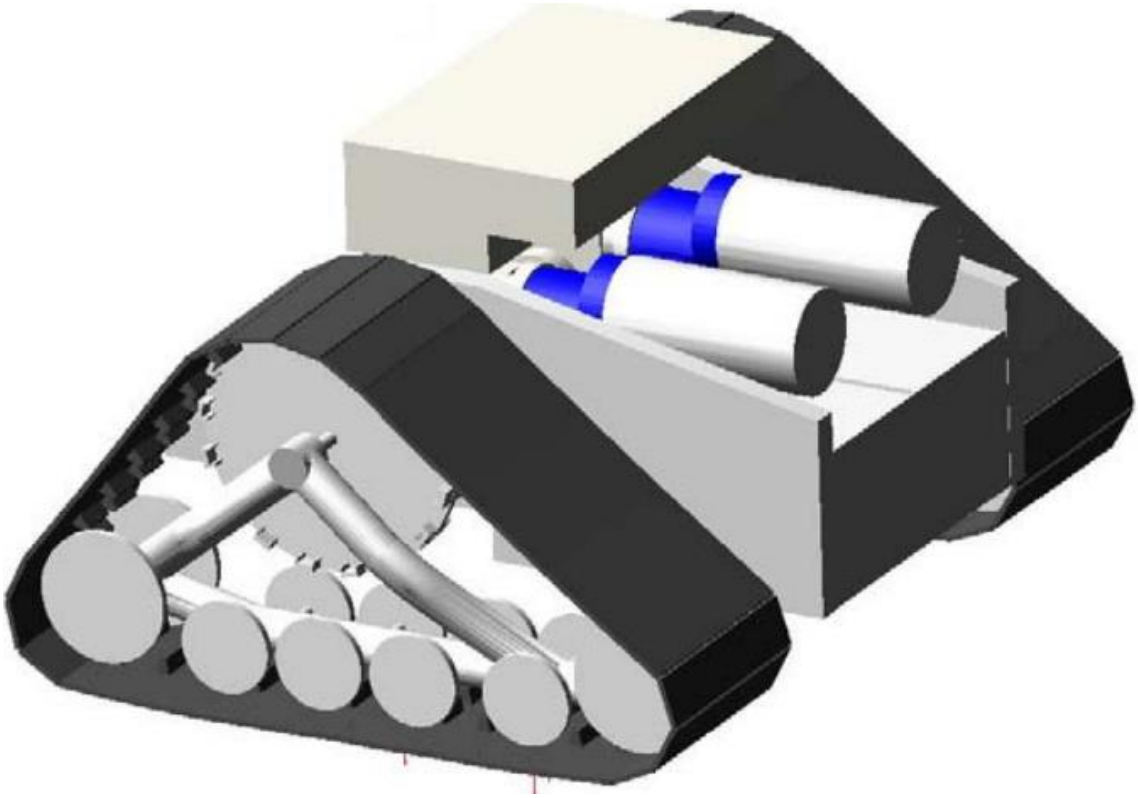


Рис. 10 Динамические сегментарные гусеницы [30]

1.2. Дополнительные источники

Также был изучен имеющийся на момент написания статьи опыт симуляции различных мобильных роботов и исследования, изучающие вопросы корректности физики симуляции и её производительности.

В статье [8] рассматриваются несколько инструментов, использующихся для симуляции мобильных роботов. Среди этих инструментов: Carnegie Mellon Robot Navigation Toolkit [29], Player-Stage-Gazebo (PSG) [40], Open Dynamics Engine (ODE) [51], Breve [22], OpenRave [10], OpenSim [9], Simbad [16], UsarSim [6].

В статье [42] рассматривается симуляция сферического мобильного робота, планирование траектории, и дальнейшие испытания модели в симуляции. Из статьи почерпнута информация о корректном проведении испытаний в симуляции и настройке физики модели.

В статье [20] авторы создали модель автономного мобильного робота, предназначенного для агросопровождения культур на поле. В статье рассматриваются вопросы моделирования как самого робота, так и его рабочей среды. Авторы использовали инструменты ROS и симулятор Gazebo. Информация о процессе и параметрах моделирования улучшила понимание построения моделей URDF в Gazebo и помогла в разработке моей модели ходовой части для робота «Сервосила Инженер»

2. Используемые инструменты и робот

2.1. Используемые инструменты

Python – популярный объектно-ориентированный язык программирования, ключевыми особенностями которого являются динамическая типизация, простота синтаксиса и семантики, обширный выбор дополнительных библиотек и модулей.

Для своего исследования я выбрал язык программирования Python из-за его совместимости с ROS и более простого синтаксиса, что позволяет вести разработку быстрее, чем на другом совместимом языке C++.

ROS (Robot Operating System) - Робототехническая Операционная Система. Это набор инструментов и библиотек, упрощающих разработку ПО для роботов. Этот открытый проект содержит в себе множество алгоритмов, драйверов, инструментов разработки и отладки, а также некоторое количество обучающих материалов и примеров [38]. На момент написания данной выпускной квалификационной работы ROS стал де-факто одним из самых популярных инструментов в робототехнике, и сейчас активно дополняется и применяется [7, 11, 39]

Gazebo — это физический и базовый графический симулятор, разработанный специально для моделирования роботов и их окружения [13]. Он включает в себя несколько физических движков (ODE [51], Bullet [12], Simbody [49] и DART [57]), графический рендеринг OGRE и имеет встроенные инструменты для создания чистых или зашумленных данных датчиков. Будучи совместимым с Робототехнической Операционной Системой (ROS [38]), Gazebo позволяет моделировать роботов, имеющих системы управления на базе ROS.

К другим наиболее популярным симуляторам относятся Webots [36], VRep (ранее известный как CoppeliaSim) [58] и другие.

Для данной работы выбран симулятор Gazebo из-за его совместимости с ROS и высококачественной физики, которые являются наиболее важными характеристиками для задачи моделирования и валидации модели высокоманевренного гусеничного робота. В Gazebo доступно несколько инструментов, подходящих для реализации ходовой части роботов, однако большинство из них предназначено для имитации колесных или шагающих роботов. Поскольку не существует готового шаблонного решения для моделирования активных гусениц, исследователи ранее использовали несколько различных приближений на основе инструментов для моделирования гусеничных роботов при использовании традиционных подходов колесных роботов.

Random Step Environment (RSE) — это имитация загроможденной среды, предложенная Национальным институтом стандартов и технологий (NIST) для оценки производительности поисково-спасательных роботов (Рис. 14) [18]. Несколько ступенчатых блоков собраны в такие конфигурации, как ступенчатое поле «тире» - последовательный ряд из пяти деревянных поддонов на прямой линии, который широко использовался в квалификационном этапе на международном соревновании RoboCup Rescue Robot League [18]. Хотя оба названия, «Random Stepfield» и «Random Step Environment», широко используются в литературе, в работе далее используется только второй вариант и его аббревиатура «RSE».

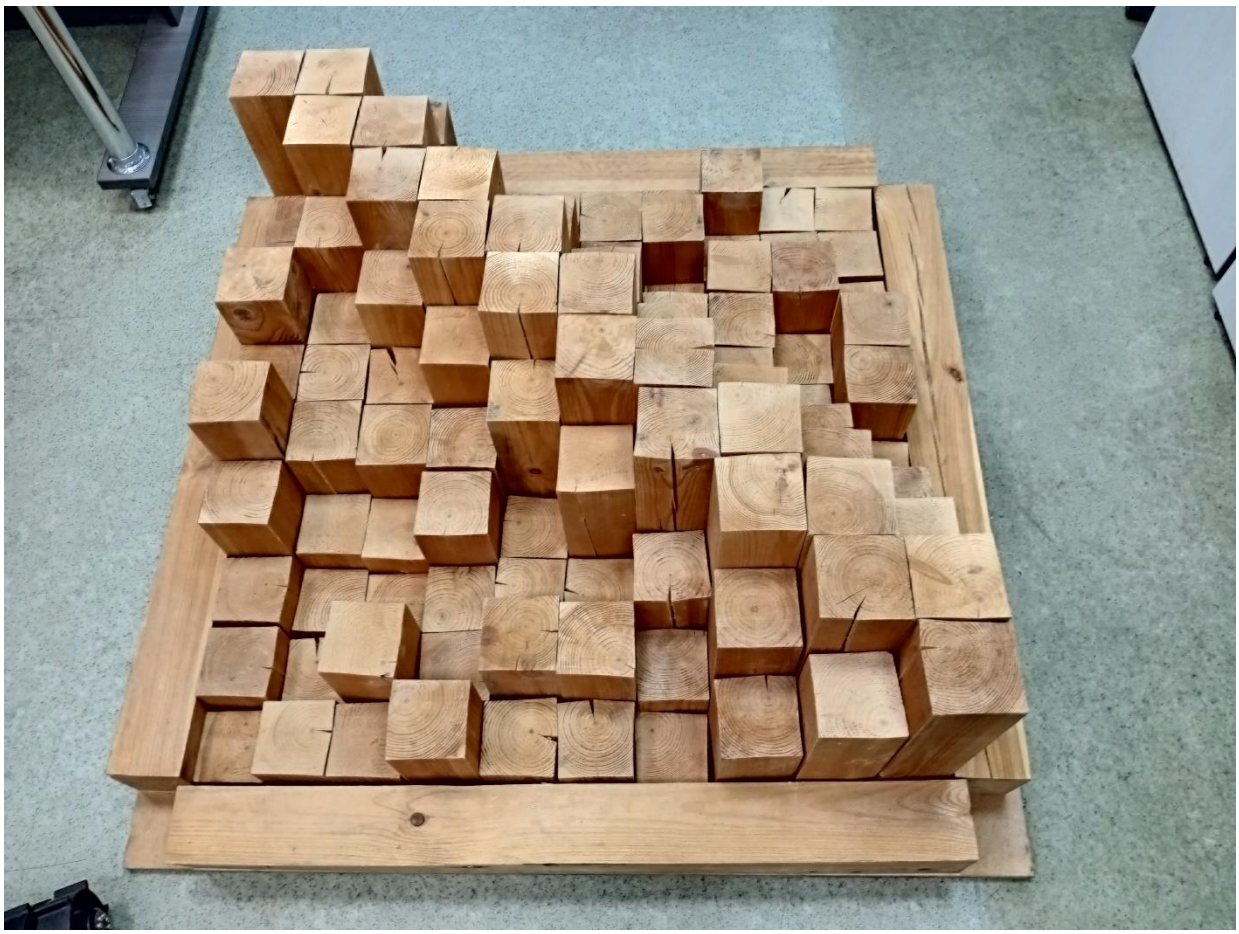


Рис. 14 Одиночная паллета RSE в ЛИРС, кабинет 1410, Институт ИТИС, КФУ.

RSE активно используются при разработке и тестировании алгоритмов, связанных с преодолением пересеченной местности. В [25] авторы представили классификацию измерения статического баланса двухгусеничного робота и использовали RSE для имитации пересеченной местности с определениями типов поз, которые строго соответствовали позе робота на RSE.

В [35] авторы выполнили физическую валидацию моделирования роботов, используя RSE в качестве одного из основных элементов своих тестовых площадок.

В [47] авторы предложили алгоритм извлечения особенностей местности из дальномерных изображений. Алгоритм обнаруживал местность RSE в виде трехмерных облаков точек и позволял извлекать данные о

местоположении робота, которые в дальнейшем можно было использовать для автономного обхода пересечённой среды. Для разработки и тестирования алгоритма на основе машинного обучения, заменяющего обычные контроллеры в задаче преодоления препятствий [48], использовалась RSE с относительно небольшими перепадами высот.

Паллеты RSE служат полигоном для испытаний мобильности и автономности в соревнованиях RoboCup Rescue Robot League [18], где в виде RSE строится как минимум несколько арен разной сложности (Рис. 15).

Рис. 15 Поле RSE на соревновании RoboCupRescue 2005 [18]

2.2. Робот «Сервосила Инженер»

Робот «Сервосила Инженер» — это мобильный гусеничный робот, разработанный российской компанией «Сервосила» для работы в условиях пересеченной местности (Рис. 16).

Ходовая часть робота представляет собой две основные фиксированные активные гусеницы, и два гусеничных активных флиппера, прикреплённых к передним концам основных гусениц. Флипперы могут вращаться вокруг этого крепления, при этом вращение обоих флипперов происходит одновременно: повернуть отдельно один из них нельзя.

Робот также имеет руку манипулятора пятью степенями свободы с захватом. Бортовые датчики включают лазерный дальномер, три передние RGB-камеры, две из которых образуют стереопару и заднюю RGB-камеру, датчик IMU (Inertial Measurement Unit), и прочие внутренние датчики.

Масса головы составляет существенную долю массы робота, что с одной стороны, привело к тому, что центр масс находится довольно высоко, что уменьшает устойчивость робота, с другой стороны при правильном управлении можно использовать голову в качестве балансира.

Рис. 16 Робот «Сервосила Инженер» в лаборатории ЛИРС

Перед началом работы были проведены замеры характеристик проходимости экземпляра реального робота, с которым я работал, и проведено сравнение с характеристиками, заявленными производителем [60].

Поскольку цель работы заключается в максимально приближённой симуляции реального робота, было решено взять за целевые показатели характеристики реального робота.

Параметр	Реальный робот	Заявлено
Макс. высота препятствия	20[1] см	22 см
Макс. линейная скорость	0,4[2] м/с	1,39 м/с
Макс. скорость поворота	8,5 об/мин	-
Макс. угол подъёма	35°[3]	35°
Тормозной путь с макс. линейной скорости	~1см	-

Таблица 1. Сравнение текущих динамических характеристик реального робота и заявленных

[1] Предположительно, возможно около 60 см, при правильном управлении. Поскольку суммарная длинна фиксированных гусениц и флипперов составляет 80 см, а длинна манипулятора с массивной «головой» немного более 80, для него возможно сохранять равновесие при уклоне около 80°, повернув руку манипулятора так, что «голова» будет находиться максимально впереди робота, что позволит забираться на препятствия такой высоты не переворачиваясь. Однако, не известно, хватит ли мощности робота для такого манёвра.

[2] Мы предполагаем, что на нашем роботе скорость ограничена в ПО на уровне контроллера в целях безопасности.

[3] Достижимо 45° и более при балансировке с помощью головы робота. Поскольку рабочая область манипулятора составляет почти полную полусферу над шасси робота, а масса его головы составляет значительную долю от массы робота, его, предположительно, всегда можно повернуть так, что центр масс останется в пределах опорной плоскости, при углах до 80°.

3. Новая модель

В ходе выполнения этой работы был предложен и разработан новый подход к моделированию гусениц, в котором используются виртуальные зубчатые колеса.

3.1. Описание идеи

Использование такой модификации для линейного подхода с большими колесами могло бы кардинально решить проблему застревания без ущерба для вычислительной производительности. В процессе работы были добавлены выступы к стандартной модели с фиксированными колесами, которые аналогичны тем, которые можно найти на большинстве реальных гусениц. Такое колесо было названо зубчатым колесом, хотя на самом деле оно не повторяет форму реального зубчатого колеса (Рис. 17).

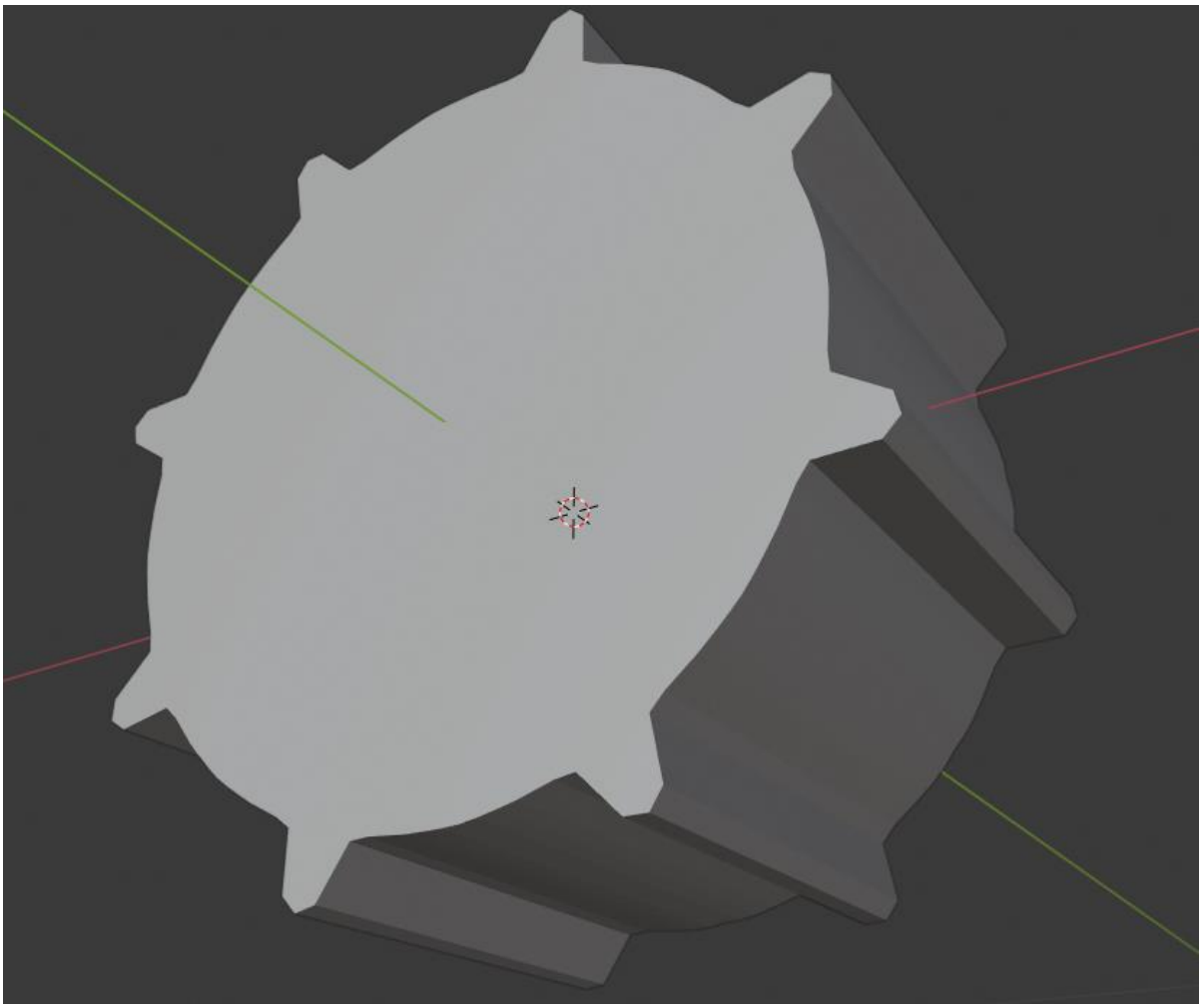


Рис. 17 Модель нового колеса

3.2. Вариант с малыми зубчатыми колёсами

В качестве отправной точки я использовали модель робота «Сервосила Инженер», созданную ранее командой лаборатории ЛИРС [31] с помощью периметрального похода с небольшими колесами, называемыми псевдоколёсами (Рис. 18). Белые цилиндры — это псевдоколеса (стандартные фиксированные колеса), а черная дорожка — это просто текстура, которая формирует визуальную резиновую гусеницу без какой-либо физики. Реальная физика взаимодействия с окружающей средой (опорной плоскостью) делегирована псевдоколёсам, которые можно включать/выключать для наглядности. Далее все малые псевдоколёса были заменены новыми зубчатыми колёсами примерно такого же размера (Рис. 19).

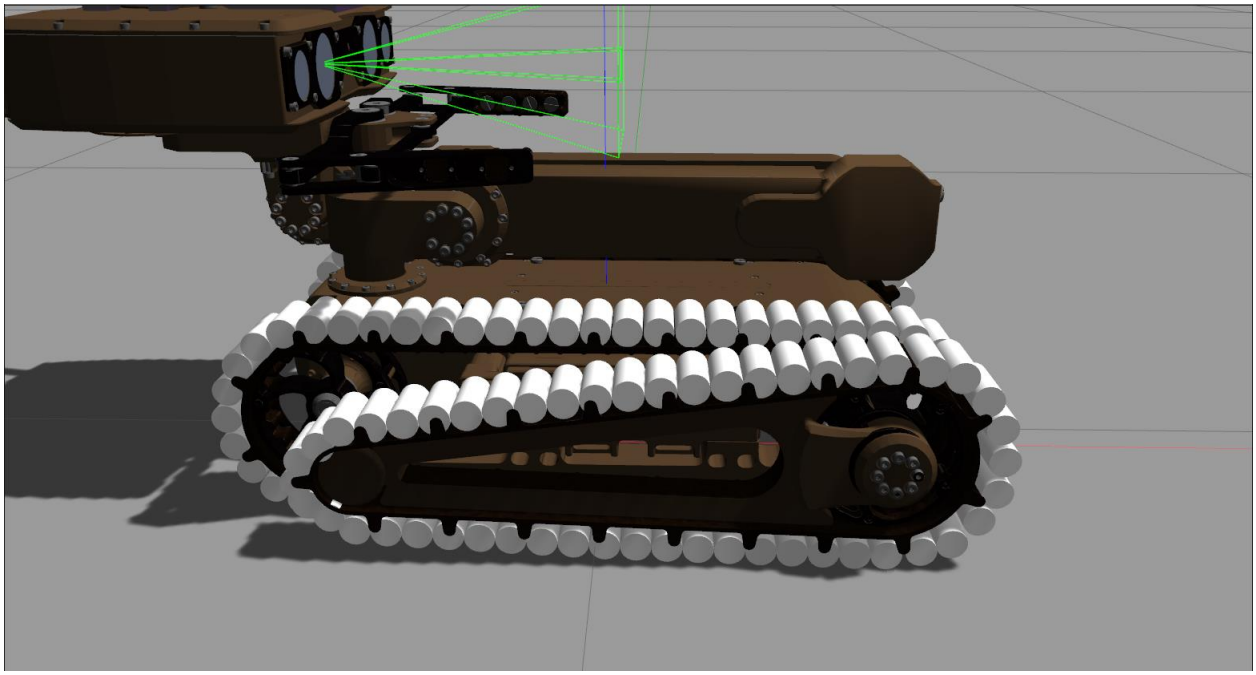


Рис. 18 Исходная модель с малыми колёсами

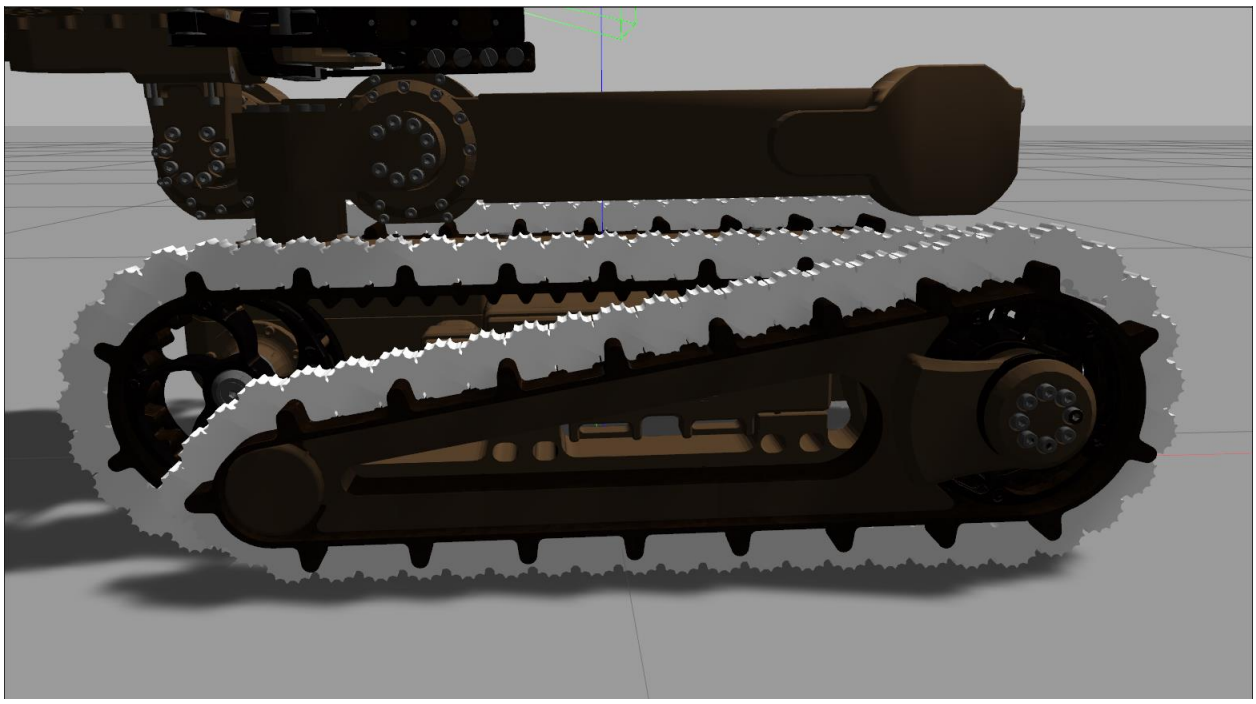


Рис. 19 Новая модель с малыми зубчатыми колёсами

Для виртуального тестирования новой концепции модели ходовой части, а также для дальнейшего сравнительного тестирования с другими моделями была использована среда RSE или случайное поле шагов, которое обеспечивает хорошее приближение к неровной местности [18]. Новая модель добилась заявленной производителем проходимости и избавилась от проблемы застревания при преодолении острых выступов препятствий. С другой стороны, поскольку новые модели коллизий колёс были сложнее, RTF падал до 0,2 для статических случаев и 0,17 для динамических случаев, когда новая модель робота проходит через RSE. К сожалению, полученный RTF новой модели неприемлем для комфортного использования модели. Приемлемым считается значение 0.7 [1].

3.3. Вариант с большими пересекающимися зубчатыми колёсами

В попытке улучшить производительность модели с точки зрения RTF была построена другая модель, в которой используется линейный подход с большими зубчатыми колесами, размер и форма выступов которых аналогичны размерам и форме выступов реального робота (Рис. 20).

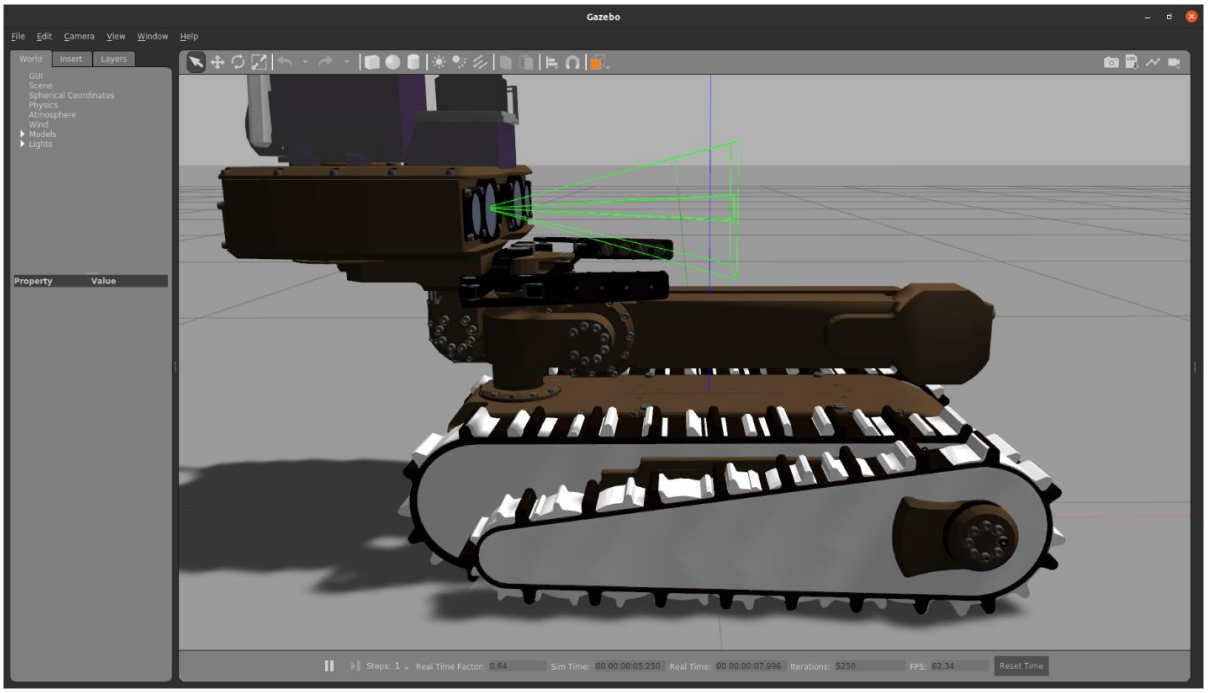


Рис. 20 Новая модель с большими зубчатыми колёсами

Начальное угловое положение каждого колеса рассчитывается с использованием соотношения $\theta / 3 \times n$, где θ — угол между двумя ближайшими отрезками, проведенными от центра колеса к наиболее удаленной (от центра колеса) точке выступа на колесе (Рис. 21); n - счетный номер колеса - счет начинается с $n=1$ для последнего (самого заднего) колеса каждой стороны моделируемой дорожки.

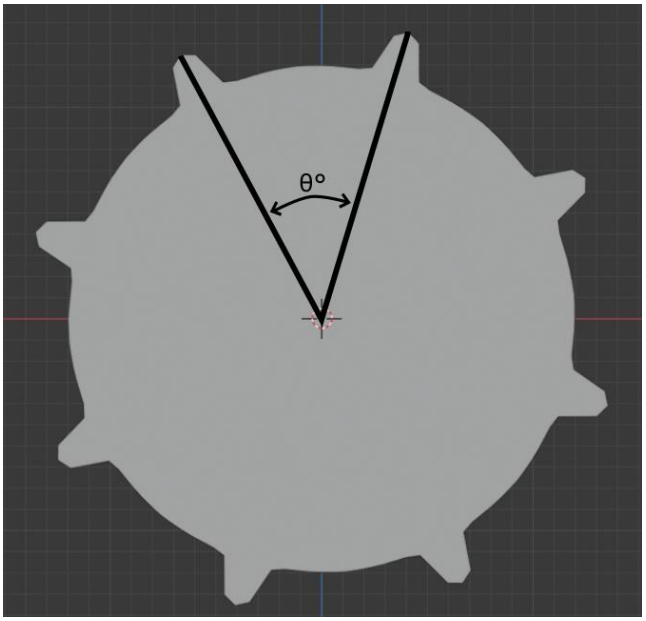


Рис. 21 Наглядное определение угла θ для вычисления начальных углов поворота колёс

Такой подход позволяет избежать излишней неестественной тряски робота во время движения из-за синхронного вращения некруглых колес. Радиус колеса для обработки одометрических данных задается как радиус описанной окружности для зубчатого колеса.

Зубчатые колеса позволили (практически полностью) решить проблему заклинивания гусеницы при преодолении острых выступов препятствий, вызванную стандартными круглыми колесами. Выступы колес позволяют модели робота, аналогично реальному гусеничному роботу, «цепляться» за поверхность препятствия при преодолении препятствия и достигать характеристик маневренности, максимально соответствующих реальному роботу. Однако из-за своей формы выступы усложняют одометрию и создают эффект тряски робота при движении по ровной поверхности. Тем не менее, эффект тряски соответствует тряске реального гусеничного робота, что приближает поведение модели к реальному роботу.

Реализация больших зубчатых колёс с линейным подходом также показала достаточную маневренность и способность преодолевать подъем без застревания. По сравнению с периметральным походом, RTF был значительно улучшен, достигнув 0,6 для статических случаев и 0,5 для динамических случаев, когда новая модель робота проходит через RSE. Кроме того, модель больших колес с линейным подходом отличается высокой степенью сходства геометрии с реальным роботом.

3.4. Настройка модели. Разделение стартовых позиций и одометрия

В качестве исходного варианта была взята модель робота «Сервосила Инженер» с разработанным в предыдущих этапах работы решением с зубчатыми колёсами (Рис. 22), которая в свою очередь является модификацией сделанных ранее моделей [31, 53]. Тёмно-оранжевые круги – реальные физические колёса, в то время как чёрный трак является лишь текстурой без физики.

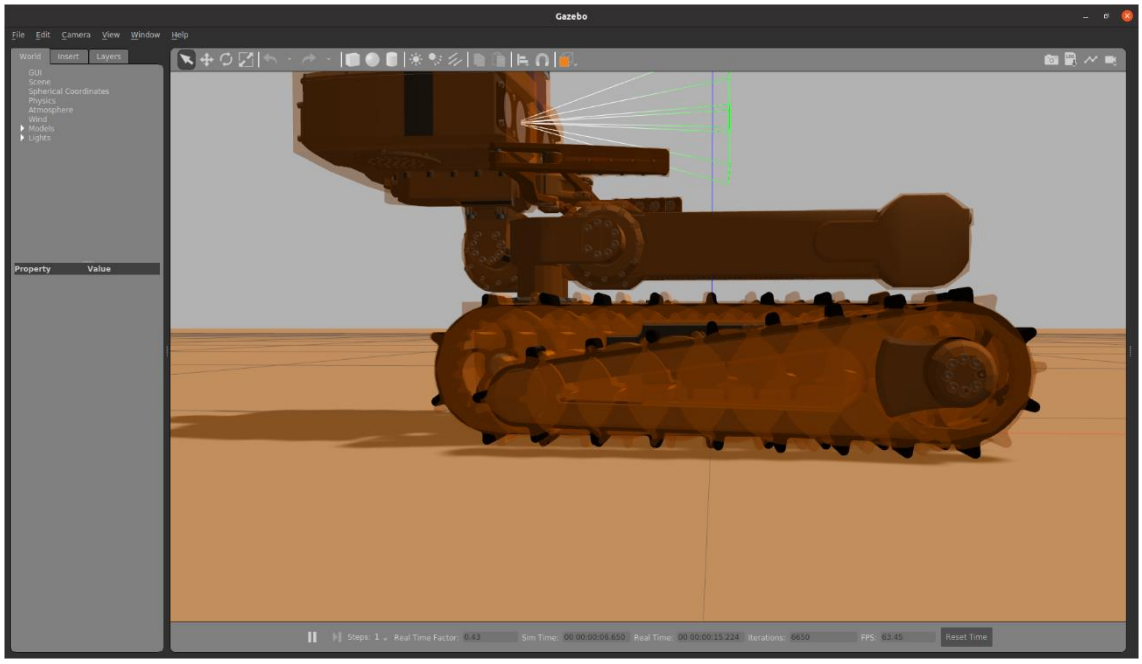


Рис. 22 Исходная модель с зубчатыми колёсами (визуализация)

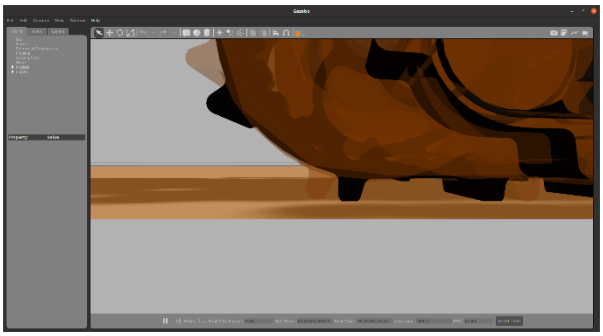
Поскольку форма новых зубчатых колёс не соответствует цилиндрической, автоматически рассчитанная одометрия некорректна. В качестве альтернативы было решено задать радиус колеса вручную как радиус описанного (вокруг зубчатого колеса) круга.

3.4.1. Стартовые позиции колёс

В начале работы были найдены следующие проблемы нового метода эмуляции траков:

1. При синхронном вращении колёс с одинаковым начальным углом поворота и небольшим количеством зубцов возникает эффект «квадратного колеса». Иначе говоря, робот опускается и поднимается по мере вращения колёс (Рис. 23 а, б). Как можно видеть на (Рис. 23 а) визуальная текстура гусеницы частично погружена в поверхность. Это приводит к появлению эффекта «подкидывания» робота по мере движения. Проблема актуальна, в основном, на ровной поверхности и небольших скоростях; на больших она менее заметна, поскольку робот не успевает полностью опускаться по мере движения.
2. При большом количестве зубцов теряется геометрическое сходство, что проявляется в том, что динамика движения робота начинает больше

отличаться от реального; в частности почти не возникает проскальзывания трака до следующего выступа при попытке взобраться на высокий уступ, поскольку расстояние между зубцами намного меньше, что делает очевидное решение первой проблемы увеличением количества зубцов неподходящим с точки зрения соответствия поведению и геометрии реальной модели.



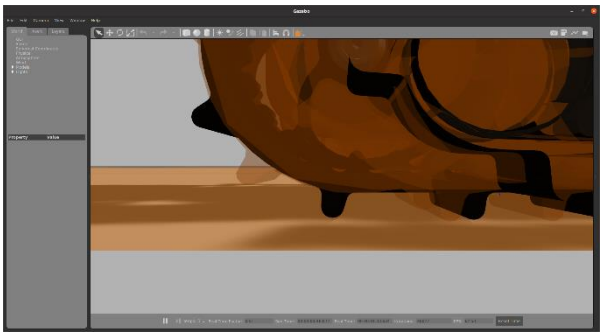


Рис. 23 а) Эффект квадратных колёс (спуск) Рис. 23 б) Эффект квадратных колёс (подъём)

На основании исследования геометрии и характера движения было решено задать различные начальные углы поворота отдельных колёс, довернув их таким образом, что в любой момент времени поверхности будут касаться разные зубцы по очереди, таким образом эффект «квадратных колёс» будет в значительной степени ослаблен, приблизившись к обычной тряске.

После изменения начальных углов поворота колёс (Рис. 24) проходимость робота осталась прежней, при этом нежелательные «подбрасывания» (Рис. 23 а, Рис. 23, б) сменились относительно реалистичной тряской.

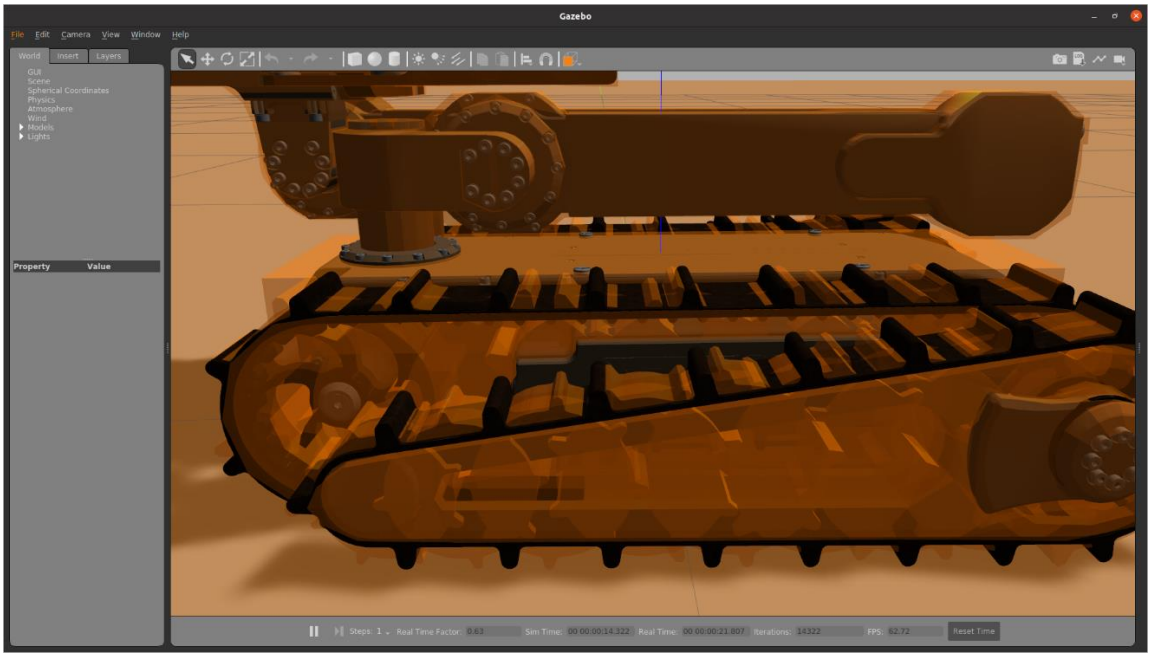


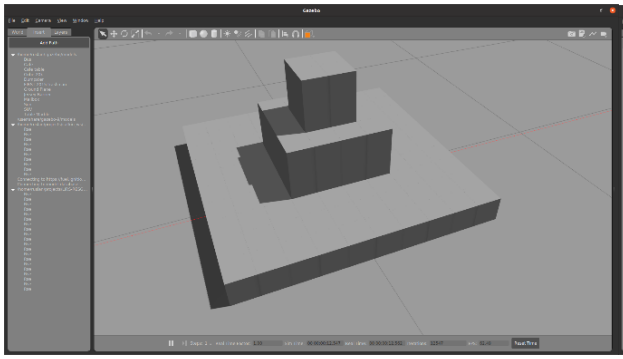
Рис. 24 Изменённая модель с зубчатыми колёсами (визуализация)

4. Генератор RSE

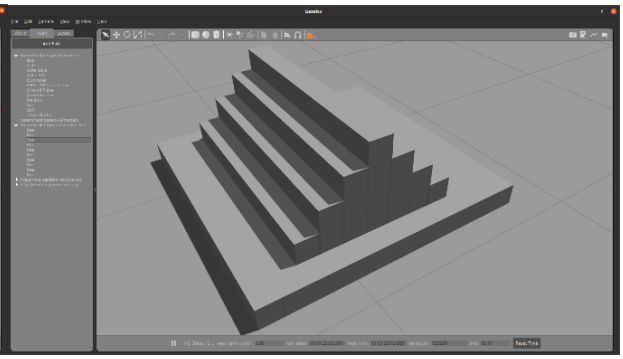
Поскольку для полноценного тестирования и отладки новой модели ходовой части требуется большое количество прогонов каждой модели на различных RSE, было решено вместо рутинной работы по ручной сборке большого количества моделей RSE из кубических примитивов, разработать инструмент для их генерации, что позволит создавать тестовые окружения в большом количестве за небольшое время.

Одиночная паллета RSE представляет собой деревянную рамку с внешней границей, которая содержит матрицу размера 10x10. В каждой ячейке находится деревянный блок, который образует одну «ступеньку» RSE.

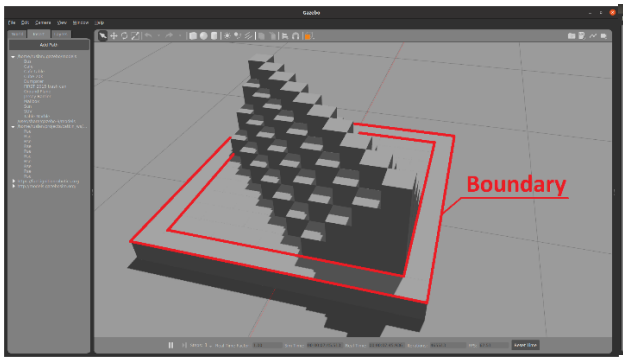
Хотя размеры блоков могут незначительно различаться, одно из типичных соглашений (которое также используется в наших исследовательских группах Казанского федерального университета и Университета Рицумейкан (Ritsumeikan University)) использует блоки шириной 10 см и глубиной 10 см и высотой H, где $H \in \{0, 5, 10, 20, 30, 40\}$ см. Внешняя граница паллеты RSE имеет высоту 10 см, глубину 10 см и ширину 120 см, и, поскольку каждая из двух соседних паллет имеет свою границу; вместе они всегда образуют двойную границу между ними, если используются стандартные паллеты. При использовании в крупномасштабной установке RSE блоки с H=10 представляют собой уровень земли, блоки с H=0 имитируют неглубокие ямы (или можно использовать блоки H=5), а другие блоки соответствуют возвышенностям различной высоты. Смешивание этих препятствий в различных комбинациях позволяет создавать широкий спектр полигонов для испытаний на пересеченной местности, которые могут образовывать типичные для реального мира стабильные структуры препятствий [35]. Данные структуры могут содержать горизонтальные барьеры (Рис. 25 б), диагональные барьеры (Рис. 25 в), проходимые или непроходимые пики (Рис. 25 а) или случайные профили местности (Рис. 25 г).



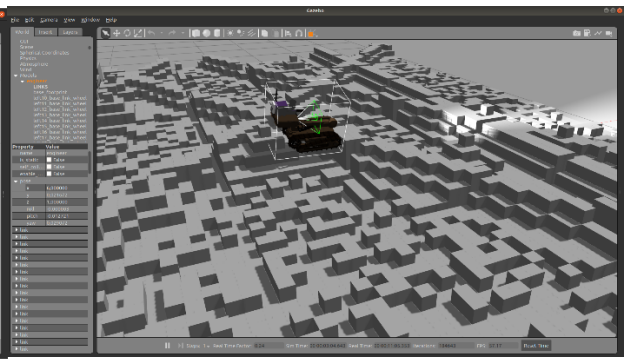
а) непроходимый пик



б) горизонтальный барьер



в) диагональный барьер



г) случайная пересечённая местность

Рис. 25 Типичные примитивные препятствия (а-в), из которых собирается сложное поле (г)

RSE, которые используются в соревнованиях RoboCup Rescue в качестве основной части полей для тестирования мобильности и навыков телеоперации, могут отличаться от стандартной высоты {0, 5, 10, 20, 30, 40} см, а также иметь паллеты размера отличного от 10x10 блоков. [18]. Отличия от стандартных высот могут выражаться в ряде непредсказуемых и, как правило, непреодолимых высоких препятствий в виде одного пика или стены (Рис 26). В случае использования паллет нестандартного размера создаются длинные коридоры и лабиринты, которые не имеют внутренних границ глубиной 20 см между каждыми участками 10x10 (Рис. 26), что приводит к более реалистичному приближению к естественному неструктурированному мусору.

Рис. 26 RSE с непроходимыми пиками и стенами без границ между паллетами на соревновании RoboCup Rescue [19]

Как имитация среды городских поисково-спасательных работ, RSE имеет ряд преимуществ, к которым относятся простота конструкции, невысокая стоимость строительных материалов, высокая мобильность и простота хранения полигона, большое разнообразие возможных форм среды и возможность быстрой перестройки среды в режиме реального времени в ходе эксперимента. Более того, поскольку конструкция модульная, любой поврежденный или разрушенный блок («ступенька») можно легко заменить. С точки зрения виртуального моделирования, поскольку RSE имеет небольшое количество полигонов, формирующих не более 5 видимых граней для каждого блока, может быть достигнута высокая производительность и низкие требования к памяти (при правильной реализации).

К недостаткам подхода RSE можно отнести невозможность построения некоторых типичных для реального мира конструкций и препятствий (например, наклонную плоскость) и простоту модели по сравнению с реальной

средой городских поисково-спасательных работ [17]. Эти проблемы снижают эффективность тестирования новых подходов на основе RSE, поскольку дальнейший переход к реальному приложению может продемонстрировать неожиданно значительное снижение производительности.

Были изучены различные существующие подходы к построению RSE с точки зрения структуры RSE, определены типовые препятствия, их формирование на поддоне RSE и параметры, влияющие на форму, высоту и другие характеристики этих препятствий и RSE как целостной конструкции. Был разработан алгоритм, который получает заданное пользователем желаемое распределение блоков и настроек RSE для формирования двумерной матрицы, которая представляет высоты блоков в каждой записи матрицы. Затем универсальный модуль строит 3D-модель RSE из матрицы.

Созданы два базовых модуля формирования 3D модели. Первый модуль формирует из матрицы единую оптимизированную модель формата .obj, которую можно в дальнейшем импортировать как стандартную модель Gazebo. Второй модуль генерирует мир Gazebo, в котором каждый блок RSE представлен как отдельная независимая модель. Второй модуль создает мир, который значительно менее продуктивен, но допускает дальнейшее ручное редактирование мира непосредственно в Gazebo.

Созданные модули были инкапсулированы в единый генератор, названный LIRS-RSEGen, с простым графическим пользовательским интерфейсом GUI (Рис. 27). Генератор был протестирован в различных режимах и настройках, при этом оценивалась пригодность построенных моделей для отработки алгоритмов навигации по пересеченной местности и преодоления препятствий для БНР в рамках типичных сред поисково-спасательных работ.

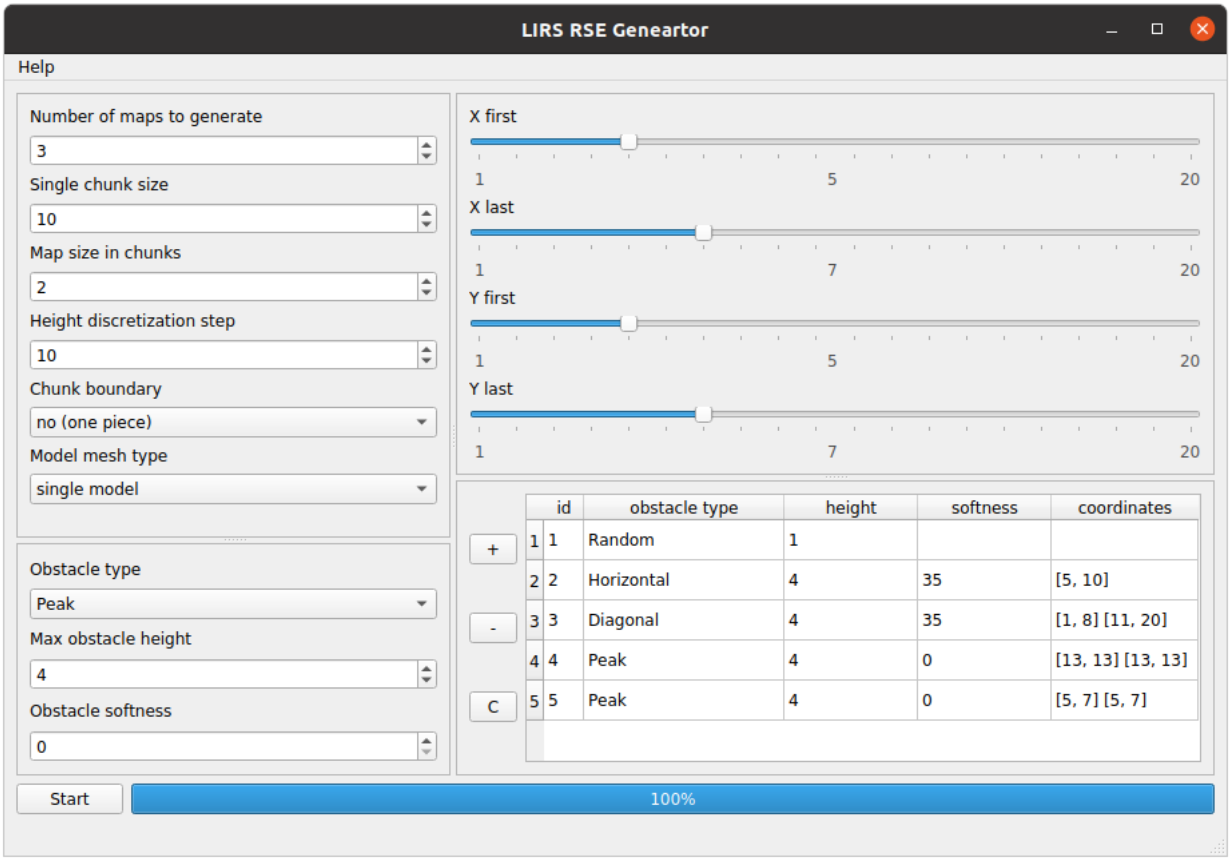


Рис. 27 Интерфейс генератора RSE

Влияние созданных моделей на производительность симуляции оценивалось с использованием RTF, FPS (Frames Per Second), системной загрузки ЦП и ГП (Графический Процессор), увеличением занимаемой памяти и скоростью загрузки модели в симуляцию.

LIRS-RSEGen создает модели и миры RSE, подходящие для импорта или для непосредственного запуска в симуляторе Gazebo. Движок Gazebo 3D для моделирования роботов и их окружения позволяет работать с миром симуляции во время выполнения симуляции, а также позволяет воссоздавать и импортировать модели RSE не перезапуская симуляцию.

Генератор работает в два этапа: первый (построение 2D-матрицы) выполняется генератором матрицы высот блоков, а второй (построение 3D-мира Gazebo) выполняется инструментом для формирования 3D-блоков из заданных элементов матрицы.

В настоящее время генератор матриц LIRS-RSEGen поддерживает пять режимов генерации, каждый из которых использует свой алгоритм построения матрицы:

1. Случайное (Random) - генерируется случайное распределение блоков по всей RSE (Рис. 25 г).
2. Случайное Гаусса (Random Gaussian) - генерирует случайное гауссово распределение блоков по всей RSE, которое немного более гладкое, чем Random.
3. Горизонтальный барьер (Horizontal barrier) - горизонтальное препятствие типа барьера (Рис. 25 б), которое появляется между двумя противоположными границами (или между двумя ячейками RSE, имеющими одну и ту же координату X или одну и ту же координату Y; эти две ячейки определяют центральную ось барьера) RSE и следует определенным пользовательским настройкам, которые определяют его длину, высоту и форму.
4. Диагональный барьер (Diagonal barrier) - диагональное препятствие барьерного типа (Рис. 25 в), которое появляется между двумя противоположными границами RSE и соответствует заданным пользователем настройкам, определяющим его длину, высоту и форму.
5. Пик (Peak) - препятствие типа пик (Рис. 25 а), длина, высота и форма которого также определяются пользовательскими настройками.

Сначала программа создает исходную квадратную нулевую матрицу M размера K x K, где K рассчитывается с использованием ряда пользовательских параметров: выбранного размера ребра карты в паллетах (chunks) (параметр «Размер карты в паллетах (Map size in chunks)» в графическом интерфейсе), размер паллеты (параметр «Размер одной паллеты (Single chunk size)») и выбор разделения паллеты (параметр "Граница паллеты (Chunk boundary)"). Паллета определяется как один фрагмент квадратной формы размером N x N поля RSE, где N — размер одной паллеты. Каждая паллета отделяется от

других паллет двойной границей высотой 10 см, если параметр границы паллеты установлен как «да (разделенный)» (англ. «Divided»), или он не имеет границы, если параметр границы паллеты установлен как «нет». Например, если пользователь выбирает карту без границ, карта А с *size_in_chunks=2* и *chunk_size=10* сформирует точно такую же начальную нулевую матрицу M_{20x20}, что и карта Б с *size_in_chunks=1* и *chunk_size=20*.

Пользователь указывает препятствия одно за другим, и они формируют список препятствий, которые будут заполнять мир Gazebo, список отображается в виде таблицы в правой нижней части графического интерфейса (Рис. 27). Указанные препятствия накладываются на исходную матрицу итеративно в том же порядке, в котором они добавлялись в список препятствий. Эта процедура может привести к ситуации, когда препятствие, добавленное в момент времени *t+1*, может частично (или даже полностью) наложиться на препятствие, добавленное в момент времени *t*, если их расположения частично (или полностью) совпадают. Эта запланированная логика работы LIRS-RSEGen не позволяет высотам препятствий суммироваться при пересечении, достигая высот выше установленного предела.

На рисунках (Рис. 28 а-б) представлены алгоритмы построения диагонального препятствия (справа) и пикового препятствия (слева).

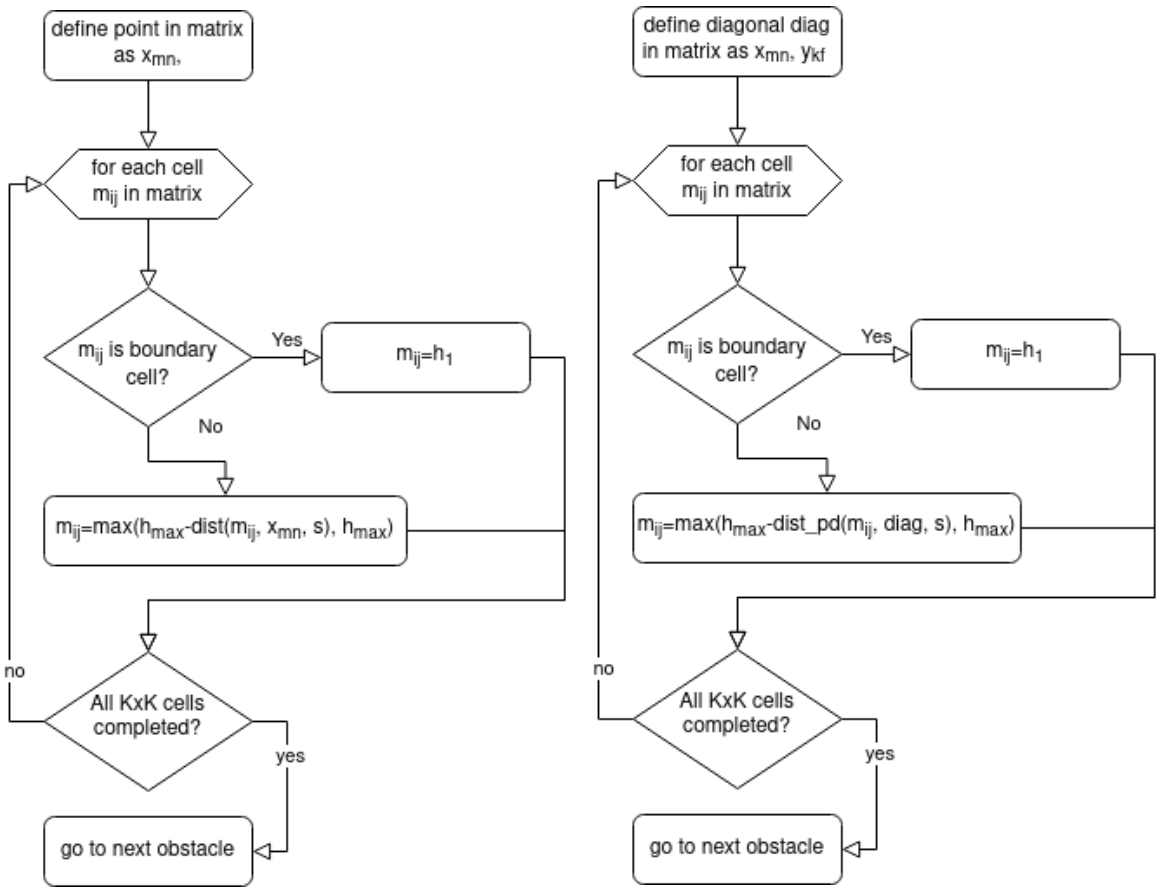


Рис. 28 а-б Алгоритмы расчёта высоты блоков RSE

Значение K определяет размер матрицы $M_{K \times K}$ (весь RSE), где m_{ij} — это один элемент матрицы $M_{K \times K}$, где $i,j \in [0,K-1]$. Массив *diag* содержит два конца $x_{m,n}$ и $y_{k,f}$ (где m, k обозначают столбцы, а n, f обозначают строки матрицы M), которые определяют случайную диагональную линию между двумя противоположными границами всего RSE.

Функция *dist_pd* вычисляет кратчайшее расстояние между координатой $m_{i,j}$ и заданной линией *diag*. Функция *dist* вычисляет расстояние между координатой $m_{i,j}$ и началом вершины $x_{m,n}$. Функция *max(a,b)* возвращает максимальное значение двух входных значений a и b.

Параметр h_1 обозначает выбранный в графическом интерфейсе шаг дискретизации высоты (Height discretization step) (в см), который является глобальным для всего RSE. Параметр h_{max} обозначает максимально возможную высоту препятствия в см, которая рассчитывается как

произведение шага дискретизации высоты (см) и "Максимальной высоты препятствия (Max obstacle height)" (которую пользователь определяет для каждого препятствия самостоятельно, в шагах дискретизации высоты). Параметр s обозначает «мягкость препятствия» (Obstacle softness), которая представляет собой скорость изменения функции высоты для каждой записи матрицы, отражающую изменения высоты с расстоянием от диагонали (или начала пика соответственно).

Генератор имеет большое количество настроек, доступных в GUI (Рис. 26) и определяющих параметры каждого типового препятствия в рамках построенной модели RSE. Список содержит два типа параметров: первые шесть параметров имеют глобальный характер и задаются для всего мира RSE (т. е. для всех препятствий), а остальные параметры имеют локальный характер и задаются для каждого препятствия независимо. В следующем списке параметры расположены в порядке их появления, сверху вниз графического интерфейса, слева направо:

1. Количество карт для генерации (Number of maps to generate): количество сгенерированных карт (миров RSE Gazebo).
2. Размер одной паллеты (Single chunk size): длина ребра одного фрагмента L_C в блоках RSE. Каждый фрагмент имеет квадратную форму L_C x L_C.
3. Размер карты в паллетах (Map size in chunks): длина ребра карты M_C в чанках, т.е. в настоящее время можно генерировать только квадратные карты. Например, при выборе $M_C=2$ и $L_C=10$ генерируется карта размером 20 x 20 ячеек RSE (или 24 x 24, включая границы, если параметру *Chunk boundary* присваивается значение «да» ("yes"), в этом случае две строки и столбец соответствуют внешним границам всего RSE, а две другие - внутренней границе между двумя соседними чанками).
4. Шаг дискретизации высоты (Height discretization step): высота одной ступеньки высоты H_{DS} в см. Например, выбор *Max obstacle height* в 5

единиц и $H_{DS}=10$ см дает возможные высоты блоков RSE в пределах {0, 10, 20, 30, 40}

5. Граница паллеты (Chunk boundary): определяет, создавать ли одну внешнюю границу и двойные внутренние границы между фрагментами (значение «yes (есть границы)») или нет (значение «no»). На рисунке (Рис. 25 а) показаны примеры фрагментов ячеек 10 x 10 RSE; в то время как для (Рис. 25 б) и (Рис. 25 в) выбраны границы, и это автоматически увеличивает общий размер фрагмента до 12 x 12 ячеек RSE (граница в одну ячейку выделена с красным цветом на (Рис. 25 в), на (Рис. 25 а) показан пик без границ фрагмента, сохраняющий исходный размер 10 x 10 ячеек RSE фрагмента.
6. Тип меша модели (Model mesh type): «Одиночная модель» (Single model) создает единую большую модель в мире Gazebo, которая содержит все блоки RSE; «Мультимодель» (Multimodel) создает одну независимую модель для каждого блока RSE (последний имеет более низкую производительность, но позволяет легко редактировать мир).
7. Тип препятствия (Obstacle type): тип нового шаблона препятствия, который является выбором из списка (диагональный барьер, горизонтальный барьер, пик, случайный, случайный с распределением Гаусса).
8. Максимальная высота препятствия (Max obstacle height): максимальная высота сгенерированного препятствия в шагах дискретизации высоты, где каждая единица имеет высоту H_{DS} см.
9. Мягкость препятствия (Obstacle softness): значение s, которое можно выбрать из [0..100]; большее значение создает препятствие, которое будет набирать высоту постепенно, образуя структуру в виде лестницы, а $s=0$ создает вертикальное препятствие. Вычисления выполняются относительно текущего чанка.
10. X first, Y first, X last и Y last координатные ползунки (правая верхняя часть графического интерфейса): эти ползунки позволяют установить

положение нового препятствия. Например, положение диагонального барьера определяют координаты вершин, служащих концами диагонального барьера, на матрице RSE. Для пика они аналогичным образом определяют точку расположения пика. В дополнение к ползункам, чтобы гарантировать, что пользователь может проверить выбранные значения, соответствующие координаты появляются под центральной частью каждого ползунка.

После завершения настройки параметров информация о текущем препятствии сохраняется в генераторе. Затем можно добавить новое препятствие с помощью кнопки «+» (Рис. 26, центральная нижняя часть), а также установить три локальных параметра и четыре ползунка координат. Информация обо всех запланированных на данный момент препятствиях в рамках RSE доступна в таблице в правой нижней части графического интерфейса. В таблице указан уникальный идентификационный номер препятствия, тип, высота, мягкость и координаты (*X, Y*). Пользователь может удалить последнее запланированное препятствие, нажав кнопку «-», или удалить все препятствия, нажав кнопку «C» (удалить все).

Кнопка *Start* в левом нижнем углу запускает генератор с запланированными препятствиями, которые генерируются итеративно в порядке их появления в списке, а индикатор выполнения справа от кнопки показывает ход процесса генерации. Наконец, кнопка *Help* в левом верхнем углу предоставляет пользователю краткое руководство по инструменту.

Второй этап генерации формирует модель для Gazebo из 3D-блоков используя построенную 2D-матрицу высот в качестве разметки высоты блоков. Инструмент работает по-разному в зависимости от настройки типа сетки модели. Выбор одной модели (Single model) преобразует матрицу, созданную на первом этапе, в трехмерную модель RSE и создает пустой мир с

этой единственной моделью. Выбор нескольких моделей (Multimodel) создает мир, в котором каждый блок будет отдельной моделью в gazebo.

Режим одиночной модели более сложен и работает напрямую с моделями .obj. Этот преобразователь создает набор из 8 вершин и 6 полигонов для каждого блока, строит блок желаемой высоты в заданной позиции и применяет один набор нормалей вершин ко всем блокам. После создания файла .obj создаются все дополнительные файлы и структуры, такие как файл .mtl и другие. Таким образом, полученную модель можно легко импортировать непосредственно в любой мир Gazebo или в работающую симуляцию не перезапуская её, также в директории /generated будет создан пустой мир Gazebo со сгенерированной моделью.

Генератор реализован на языке python 3.6 с использованием библиотек *NumPy* [37] и *PyQt5* [59]. LIRS-RSEGen следует стандарту моделей SDF для генерации мира и формату файлов геометрии OBJ для 3D-моделей.

4.1. Валидация виртуальных моделей RSE

Миры, созданные LIRS-RSEGen, были протестированы в симуляторе Gazebo с использованием виртуальной модели гусеничного робота Servosila Engineer и трех одновременно работающих колесных БНР TurtleBot3 [3]. Тесты продемонстрировали эффективность построенных RSE-миров с точки зрения коэффициента реального времени Gazebo (RTF), загрузки процессора и памяти, которые имели приемлемые значения даже для относительно больших RSE размером 80x80 блоков.

Наиболее перспективно дальнейшее использование генератора для разработки алгоритмов машинного обучения, поскольку возможность создания тысяч различных миров, не перегружающих систему, за короткий промежуток времени могла бы значительно ускорить исследовательские работы в области картографии, навигации и преодоление пересеченной местности. Инструмент доступен для бесплатного академического

использования в учетной записи Gitlab Лаборатории интеллектуальных робототехнических систем (LIRS) {LIRS-RSEGen или генератор LIRS RSE, GitLab, https://gitlab.com/LIRS_Projects/LIRS-RSEGen}.

5. Сравнительные испытания моделей и робота на RSE

Для виртуального тестирования был построен ряд типичных для среды городских поисково-спасательных работ препятствий в виде RSE и проверены четыре разные модели робота «Сервосила Инженер» с одинаковыми настройками в режиме телеоперации.

5.1. Случайное препятствие

На RSE со случайными блоками модель «Сервосила Инженер» со стандартными круглыми колесами и периметральным подходом застряла и не преодолела препятствие во всех запусках (Рис. 29 a). Модель с зубчатыми колесами и линейным подходом успешно преодолела препятствие во всех запусках (Рис. 29 b).

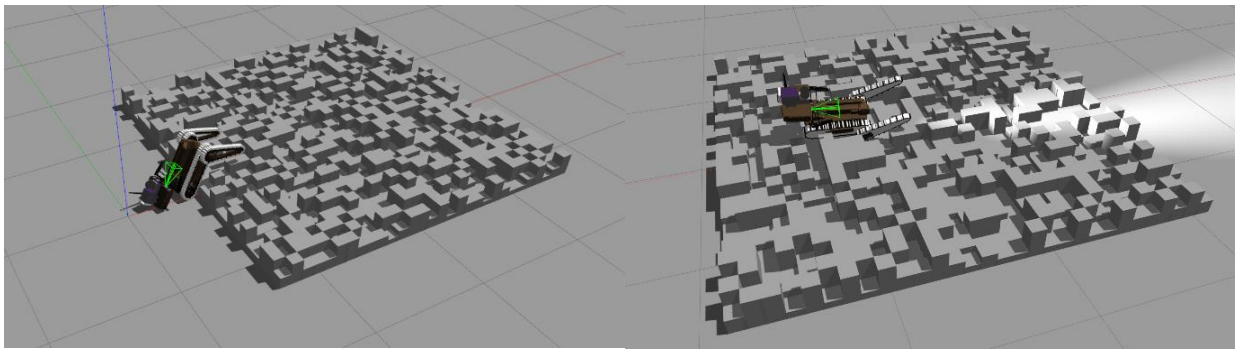


Рис. 29 а-б Старая и новая модели робота на рандомном RSE

5.2. Горизонтальный барьер

На RSE с горизонтальным барьерным препятствием модель со стандартными круглыми колесами и периметральным подходом успешно преодолевала границу паллеты во всех запусках, но не преодолела само препятствие, так как застряла в наклонном положении из-за попадания углов блоков между колёсами (Рис. 30 a). Модель с зубчатыми колесами и линейным подходом успешно преодолела препятствие во всех запусках (Рис. 30 b).

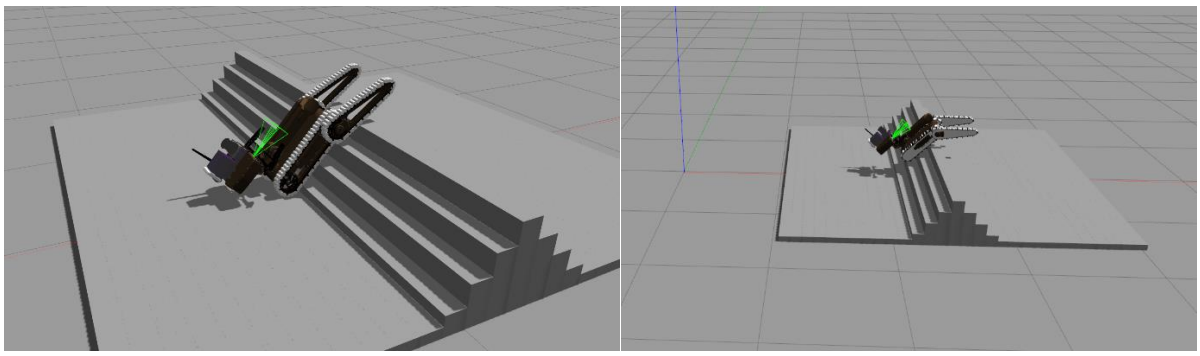


Рис. 30 а-б Старая и новая модели робота на RSE с горизонтальным барьером

5.3. Диагональный барьер

На RSE с диагональным барьером модель со стандартными круглыми колесами и периметральным подходом не преодолела препятствие, так как застряла в наклонном положении или, если ей удавалось начать взбираться на барьер, переворачивалась во всех испытаниях (Рис. 31 a). Модель с зубчатыми колесами и линейным подходом успешно преодолела препятствие в 40% запусков и перевернулась в 60% запусков (Рис. 31 b).

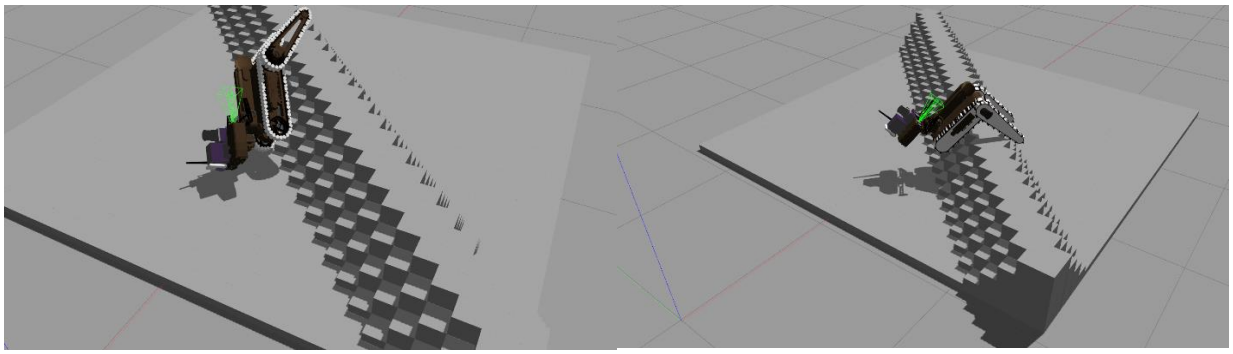


Рис. 31 а-б Старая и новая модели робота на RSE с диагональным барьером

Результаты всех испытаний по преодолению препятствий собраны в Таблице 2. Для каждой модели было проведено 20 запусков.

Модель	Тип препятствия и процент успешных запусков		
	Рандомный RSE	RSE с горизонтальным барьером	RSE с диагональным барьером
Модель с малыми колёсами	0%	0%	0%
Модель с большими колёсами	15%	10%	0%
Модель с малыми зубчатыми колёсами	100%	100%	50%
Модель с большими зубчатыми колёсами	100%	100%	40%

Таблица 2. Статистика испытаний прохода моделей через стандартные препятствия

5.4. Испытания характеристик проходимости

Были проведены испытания ключевых характеристик проходимости всех предыдущих моделей робота [31, 32, 53] и новой модели, а также реального робота (Рис. 32). Были экспериментально проверены:

- Предельная скорость
- Максимальная преодолённая высота
- Скорость поворота
- Тормозной путь
- Ускорение
- Скорость симуляции (для реального робота всегда считаем за 1)

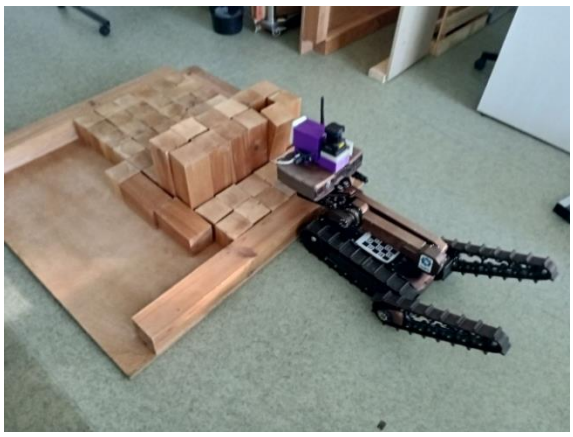



Рис. 32 а-б Испытания проходимости робота через горизонтальный барьер высотой в 20 см

Модель	Предельная скорость	Максимальная преодолённая высота[4]	Скорость поворота	Тормозной путь	Ускорение	Скорость симуляции (RTF)
Реальный робот	0,4[5] м/с	20 см	8,5 об/мин	0,02 м	0,2 м/с^2	1.0
Модель с малыми колёсами	0,5[2] м/с	10 см	4 об/мин	1 м	0,1 м/с^2	0.2
Модель с большими колёсами	0,4[2] м/с	15 см	2 об/мин	1 м	0,1 м/с^2	0.6
Модель с малыми зубчатыми колёсами	0,4[2] м/с	20 см	2 об/мин	0,5 м	0,1 м/с^2	0.2
Модель с большими зубчатыми колёсами	0,4[2] м/с	20 см	4 об/мин	0,3 м	0,2 м/с^2	0.65

Таблица 3. Сравнение ключевых ходовых характеристик реального робота и разных моделей

После завершения доработок и тестов новая модель была синхронизирована с репозиторием GitLab[6] и внесена в него на ветку gear_wheels_new. Версию ходовой части можно выбирать, используя параметр

[4] Препятствие в виде параллелепипеда на ровной поверхности. Робот двигается без раскачивания или других нефункциональных действий, позволяющих преодолеть большую высоту.
[5] Предположительно ограниченно программно на низком уровне. В спецификациях указана скорость до 1,39 м/с
[6] https://gitlab.com/LIRS_Projects/Engineer-gazebo-model

drive_version (доступны версии v1-v4) по умолчанию используется v4. Версии моделей следующие:

- v1: Периметральный подход с круглыми колёсами; ранее разработанная модель ЛИРС Ильи Москвина [31], [32]
- v2: Периметральный подход с зубчатыми колёсами
- v3: Линейный подход с круглыми колёсами
- v4: Линейный подход с зубчатыми колёсами

Проведённые тесты показали, что доработанный метод показывает хорошее соответствие реальной модели, отклоняясь по ключевым параметрам не более, чем на 15% по проходимости и 5% по скорости на ровной поверхности. При этом, улучшенная версия модели не демонстрирует какого-либо несвойственного реальной модели поведения в определённых ситуациях, таких как застревание на уступах, эффект «квадратного колеса» и прочих. На основе всего вышесказанного можно сделать вывод, что данная модель пригодна для демонстрации и разработки на её основе, если не требуется полное соответствие реальной модели.

6. Дальнейшее развитие проекта

После настройки позиций колёс и параметров одометрии модели, аналогичные параметры были введены в модуль 3d SLAM Rtabmap и проведены тесты на корректность формирования облака точек и карты (Рис. 33).

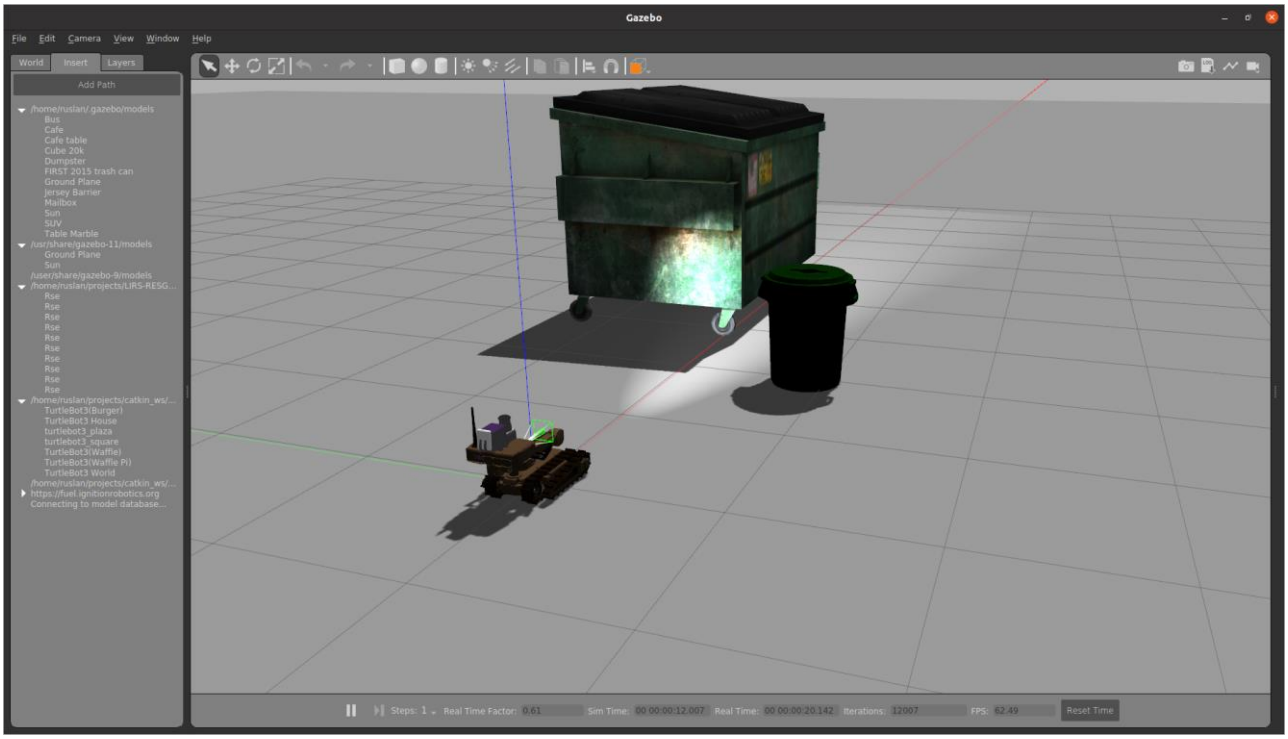


Рис. 33 Реальный вид в Gazebo

В процессе тестирования выявлено, что новая модель поддерживает корректное формирование облака точек и локализацию в нём, что также подтверждает корректность заданных параметров одометрии (Рис. 34-35).

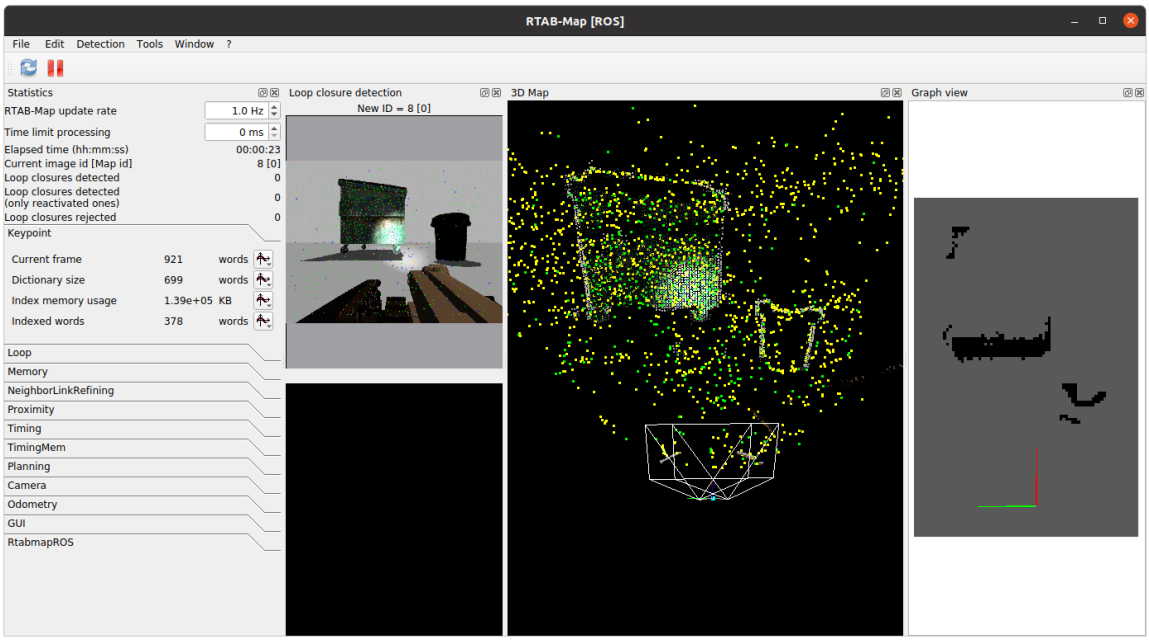


Рис. 34 Вид в Rtabmap GUI

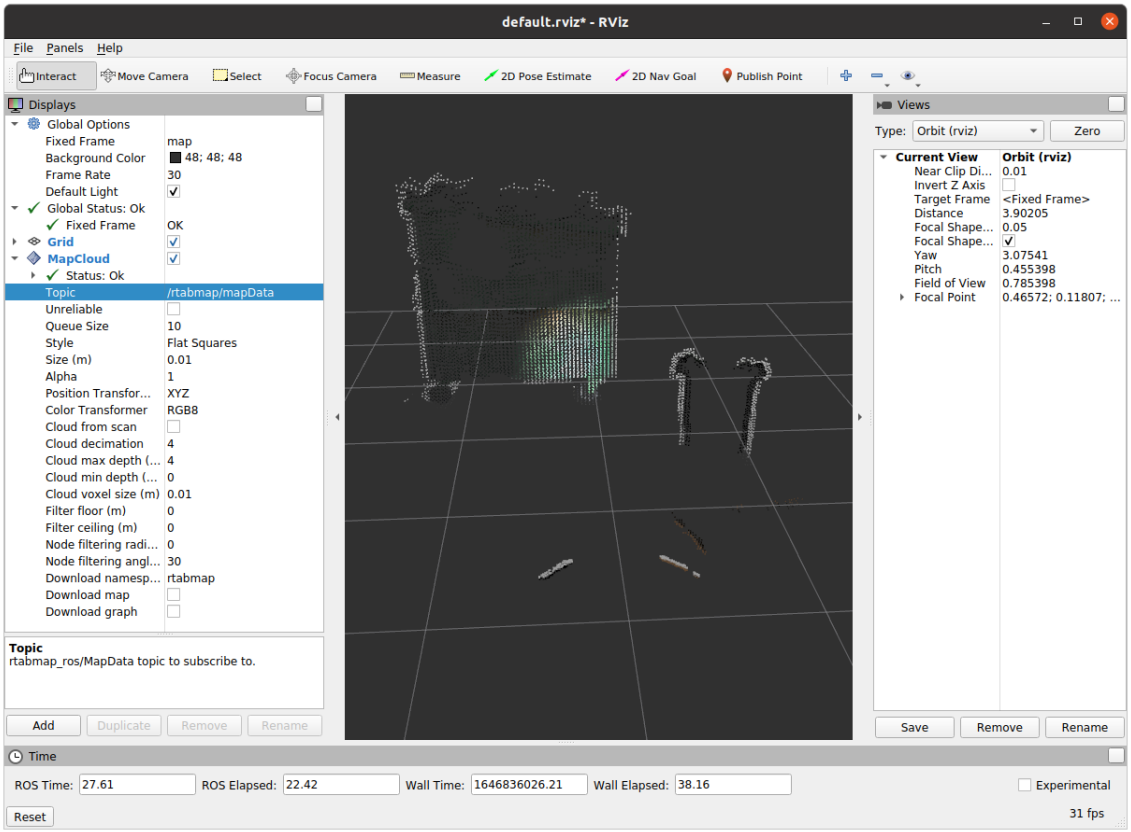


Рис. 35 Облако точек в RViz

В дальнейшем созданное облако точек может быть использовано для решения задач локализации и картографии, что вместе с применением модуля управления движением позволит роботу передвигаться к заданной цели автоматически, без участия оператора [23]

ЗАКЛЮЧЕНИЕ

Моделирование колесных мобильных роботов хорошо изучено и для него уже сформированы общепринятые подходы и стандартные инструменты. В то же время, реалистичное моделирование гусеничного робота представляет собой ещё нерешённую до конца задачу. Фактически на сегодня не существует однозначно правильного подхода, не имеющего каких-либо существенных недостатков. В этой работе рассмотрены существующие подходы к моделированию гусеничных роботов в симуляторе Gazebo и других симуляторах, и представлен новый подход, который аппроксимирует каждую гусеницу робота набором зубчатых колес.

Был применён метод сравнительного анализа нескольких подходов к моделированию гусеничных роботов на примере робота «Сервосила Инжненер» в симуляторе Gazebo. Оценено соответствие динамики движения моделей реальному роботу и факторы производительности симуляции в виде RTF относительно эталонного времени (1.0), нагрузки ЦП и ГП. В качестве эталонных значений в испытаниях использовались реальные параметры робота. Для виртуальных испытаний был построен ряд типовых препятствий среды городских поисково-спасательных работ и проверены четыре разные модели робота «Сервосила Инженер» с одинаковыми настройками в телеоперационном режиме.

Результаты сравнения показали, что два новых подхода на основе зубчатых колес осуществимы с точки зрения нагрузки на ЦП и обеспечивают лучшее приближение к реальной производительности робота. Кроме того, они успешно устранили проблему застревания при преодолении острых выступов препятствий, что характерно для подходов на основе псевдоколес.

В заключение, можно сказать, что разработанный метод является хорошим решением, если для модели робота важно близкое соответствие динамике реального робота и относительно высокая производительность при не слишком больших затратах на разработку. Если основной акцент делается на соответствие динамике робота, то качественно реализованные

сегментарные гусеницы, предположительно, позволят достичь еще большего соответствия. Если производительность имеет первостепенное значение или если робот будет двигаться исключительно по ровной поверхности, то стандартные круглые колеса являются лучшим решением.

Результаты работы представлены на международной конференции The 24th International Conference Series on Climbing and Walking Robots (CLAWAR 2021) [14] и приняты для доклада на конференции The 19th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2022) [15]. Исходный код новых моделей робота «Сервосила Инженер» [15] размещен по ссылке https://gitlab.com/LIRS_Projects/Engineer-gazebo-model. Исходный код генератора RSE-миров [14] размещен по ссылке https://gitlab.com/ LIRS_Projects/LIRS-RSEGen.

ГЛОССАРИЙ

- Мобильный робот, или же Беспилотный Наземный Аппарат (БНА) или Беспилотный Наземный Робот (БНР), (англ. «Unnamed Ground Vehicle», UGV) – этими терминами принято называть любого робота, самостоятельно передвигающегося по опорной поверхности различного типа, в том числе по пересеченной местности. БНА является наиболее широко используемым типом роботов для решения самых разных задач, включая робототехнику для городских поисково-спасательных работ.
- Гусеничный робот — это тип мобильных роботов, которые используют различные типы гусениц в качестве ходовой части. Гусеничные роботы используются, когда задача требует повышенной проходимости транспортного средства по пересечённой местности, включая исследование неизвестной местности, добычу полезных ископаемых, городские поисково-спасательные работы.
- Поиск И Спасение (англ. «Search And Rescue», SAR) – это задача по поиску жертв природных или антропогенных катастроф за ограниченное время, а затем оказание им своевременной необходимой медицинской (и иной) помощи для предотвращения угрозы жизни, а также минимизации последствий катастрофы для здоровья пострадавших. Поскольку эта задача требует незамедлительного реагирования, в ней всё чаще задействуют роботов, особенно в случаях, когда среда слишком опасна для человека, например, при пожаре рядом с взрывоопасными веществами или заражении местности токсичными газами [45].
- Городские поисково-спасательные работы (англ. «Urban Search And Rescue», USAR) – это подвид задачи поиска и спасения, является наиболее очевидным примером задачи, требующей высокого уровня проходимости робота. USAR был введен в конце 20-го века как отдельная ветвь полевой робототехники, которая концентрируется на механике роботов-спасателей, их навигации, картографировании и других классических задачах робототехники, а также взаимодействии

человека с роботом [5], рассматриваемых через призму задач поиска и спасения. Одной из основных задач USAR является поиск пострадавших в частично поврежденных или полностью разрушенных техногенных сооружениях. Таким образом, типичная среда городских поисково-спасательных работ для БНА содержит груды обломков и мусора, образованные строительными материалами, мебелью, различной техникой, бытовыми и офисными предметами, что затрудняет наблюдение, локализацию и картографирование окружающей среды [45, 27, 41].

- Фактор Реального Времени (англ. RTF, Real Time Factor) – Отношение между скоростью симуляции к скорости реального времени. Значения выше 1 значат, что время в симуляции идёт быстрее, чем в реальности. 1 соответствует реальному времени, значения ниже 1 значат, что время в симуляции медленнее, чем в реальности.
- Одометрия – данные о движении робота, полученные от приводов этого робота. Используется для определения пройденного пути и положения подвижных частей робота относительно шасси. Некоторые приводы не предоставляют данные одометрии.
- RGB-камера – Камера, воспринимающая свет в 3 каналах: красный, зелёный и голубой. Стандартный тип цветных камер, используемый повсеместно на момент написания работы.
- SLAM - Simultaneous Localization and Mapping, Одновременная локализация и картографирование, эта задача подразумевает одновременное составление карты окружающей местности и определение роботом своего положения на ней только с помощью данных бортовых сенсоров (или данных сенсоров всех связанных роботов, если речь о роевой робототехнике), без каких-либо исходно заданных данных об окружающей местности или собственном положении в пространстве.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Abbyasov B. et al. Automatic tool for gazebo world construction: from a grayscale image to a 3d solid model // 2020 IEEE International Conference on Robotics and Automation (ICRA). – IEEE, 2020. – P. 7226-7232.
2. Acosta B., Yang W., Posa M. Validating Robotics Simulators on Real-World Impacts // IEEE Robotics and Automation Letters. – 2022.
3. Amsters R., Slaets P. Turtlebot 3 as a robotics education platform // International Conference on Robotics in Education (RiE). – Springer, Cham, 2019. – P. 170-181.
4. Borisov A. V. et al. Describing the motion of a body with an elliptical cross section in a viscous uncompressible fluid by model equations reconstructed from data processing // Technical Physics Letters. – 2016. – Vol. 42. – №. 9. – P. 886-890.
5. Burke J. L. et al. Final report for the DARPA/NSF interdisciplinary study on human-robot interaction // IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews). – 2004. – Vol. 34. – №. 2. – P. 103-112.
6. Carpin S. et al. USARSim: a robot simulator for research and education // Proceedings 2007 IEEE International Conference on Robotics and Automation. – IEEE, 2007. – P. 1400-1405.
7. Cashmore M. et al. Rosplan: Planning in the robot operating system // Proceedings of the International Conference on Automated Planning and Scheduling. – 2015. – Vol. 25. – P. 333-341.
8. Castillo-Pizarro P., Arredondo T. V., Torres-Torriti M. Introductory survey to open-source mobile robot simulation software // 2010 Latin American Robotics Symposium and Intelligent Robotics Meeting. – IEEE, 2010. – P. 150-155.
9. Delp S. L. et al. OpenSim: open-source software to create and analyze dynamic simulations of movement // IEEE transactions on biomedical engineering. – 2007. – Vol. 54. – №. 11. – P. 1940-1950.

10. Diankov R., Kuffner J. Openrave: A planning architecture for autonomous robotics // Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34. – 2008. – Vol. 79.
11. Dieber B. et al. Security for the robot operating system // Robotics and Autonomous Systems. – 2017. – Vol. 98. – P. 192-203.
12. Erez T., Tassa Y., Todorov E. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx // 2015 IEEE international conference on robotics and automation (ICRA). – IEEE, 2015. – P. 4397-4404.
13. Foundation, O. S. R. (2021). Gazebo official site. http://gazebosim.org/.
14. Gabdrahmanov, R., Tsoy, T., Bai, Y., Svinin, M., Magid, E. (2021). Automatic Generation of Random Step Environment Models for Gazebo Simulator. Lecture Notes in Networks and Systems, 324, p. 408-420.
15. Gabdrahmanov, R., Tsoy, T., Bai, Y., Svinin, M., Magid, E. (2022). Gear wheels based simulation of crawlers for mobile robot. The 19th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2022), p – (*принята к печати*).
16. Hugues L., Bredeche N. Simbad: an autonomous robot simulation package for education and research // International Conference on Simulation of Adaptive Behavior. – Springer, Berlin, Heidelberg, 2006. – P. 831-842.
17. Jakobi N., Husbands P., Harvey I. Noise and the reality gap: The use of simulation in evolutionary robotics // European Conference on Artificial Life. – Springer, Berlin, Heidelberg, 1995. – P. 704-720.
18. Jacoff A. et al. Stepfield pallets: Repeatable terrain for evaluating robot mobility // Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems. – 2008. – P. 29-34.
19. Jacoff A. et al. Using competitions to advance the development of standard test methods for response robots // Proceedings of the Workshop on Performance Metrics for Intelligent Systems. – 2012. – P. 182-189.

20. Iqbal J. et al. Simulation of an autonomous mobile robot for LiDAR-based in-field phenotyping and navigation // Robotics. – 2020. – Vol. 9. – №. 2. – P. 46.
21. Kenwright B., Morgan G. Practical introduction to rigid body linear complementary problem (lcp) constraint solvers // Algorithmic and Architectural Gaming Design: Implementation and Development. – IGI Global, 2012. – P. 159-201.
22. Klein J. Breve: a 3d environment for the simulation of decentralized systems and artificial life // Proc. of the Int. Conf. on Artificial Life. – 2003. – P. 329-334.
23. Lavrenov R., Zakiev A. Tool for 3D Gazebo map construction from arbitrary images and laser scans // 2017 10th International Conference on Developments in eSystems Engineering (DeSE). – IEEE, 2017. – P. 256-261.
24. Lee C. H. et al. Double-track mobile robot for hazardous environment applications // Advanced Robotics. – 2003. – Vol. 17. – №. 5. – P. 447-459.
25. Magid E., Tsubouchi T. Static balance for rescue robot navigation: Discretizing rotational motion within random step environment // International Conference on Simulation, Modeling, and Programming for Autonomous Robots. – Springer, Berlin, Heidelberg, 2010. – P. 423-435.
26. Makoviychuk V. et al. Isaac gym: High performance gpu-based physics simulation for robot learning // arXiv preprint arXiv:2108.10470. – 2021.
27. Malov D., Edemskii A., Saveliev A. Proactive localization system as a part of a cyberphysical smart environment // 2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM). – IEEE, 2019. – P. 1-5.
28. Michel O. Cyberbotics Ltd. Webots™: professional mobile robot simulation // International Journal of Advanced Robotic Systems. – 2004. – Vol. 1. – №. 1. – P. 5.
29. Montemerlo M., Roy N., Thrun S. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit //

Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453). – IEEE, 2003. – Vol. 3. – P. 2436-2441.

30. Morita S. et al. Kinematic track modelling for fast multiple body dynamics simulation of tracked vehicle robot // 2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR). – IEEE, 2018. – P. 910-915.
31. Moskvin I., Lavrenov R. Modeling tracks and controller for servosila engineer robot // Proceedings of 14th International Conference on Electromechanics and Robotics «Zavalishin's Readings». – Springer, Singapore, 2020. – P. 411-422.
32. Moskvin I. et al. Modelling a crawler robot using wheels as pseudo-tracks: model complexity vs performance // 2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA). – IEEE, 2020. – P. 1-5.
33. Ohnishi R., Hoshino Y. Position Estimation Using Stereo Camera Images and Physics Engine Simulation for Robot Control // 2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS). – IEEE, 2018. – P. 753-757.
34. Pecka M., Zimmermann K., Svoboda T. Fast simulation of vehicles with non-deformable tracks // 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – IEEE, 2017. – P. 6414-6419.
35. Pepper C., Balakirsky S., Scrapper C. Robot simulation physics validation // Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems. – 2007. – P. 97-104.
36. Prabhakar M. et al. Design and simulation of an automated guided vehicle through webots for isolated COVID-19 patients in hospitals // 2020 IEEE 4th Conference on Information & Communication Technology (CICT). – IEEE, 2020. – P. 1-5.

37. Oliphant T. E. A guide to NumPy. – USA : Trelgol Publishing, 2006. – Vol. 1. – P. 85.
38. Quigley M. et al. ROS: an open-source Robot Operating System // ICRA workshop on open source software. – 2009. – Vol. 3. – №. 3.2. – P. 5.
39. Quigley M., Gerkey B., Smart W. D. Programming Robots with ROS: a practical introduction to the Robot Operating System. – «O'Reilly Media, Inc.», 2015.
40. Rusu R. B. et al. Extending Player/Stage/Gazebo towards cognitive robots acting in ubiquitous sensor-equipped environments // ICRA workshop for networked robot systems. – 2007.
41. Safin R., Lavrenov R., Martínez-García E. A. Evaluation of visual slam methods in usar applications using ros/gazebo simulation // Proceedings of 15th International Conference on Electromechanics and Robotics «Zavalishin's Readings». – Springer, Singapore, 2021. – P. 371-382.
42. Sang S. et al. Modeling and simulation of a spherical mobile robot // Computer Science and Information Systems. – 2010. – Vol. 7. – №. 1. – P. 51-62.
43. Shabalina K., Sagitov A., Magid E. Comparative analysis of mobile robot wheels design // 2018 11th International Conference on Developments in esystems Engineering (dese). – IEEE, 2018. – P. 175-179.
44. Shabalina K. et al. Avrora unior car-like robot in gazebo environment // International Conference on Artificial Life and Robotics. – 2019. – P. 116-119.
45. Shah B., Choset H. Survey on urban search and rescue robots // Journal of the Robotics Society of Japan. – 2004. – Vol. 22. – №. 5. – P. 582-586.
46. Sheh R. et al. Advancing the state of urban search and rescue robotics through the robocuprescue robot league competition // Field and service robotics. – Springer, Berlin, Heidelberg, 2014. – P. 127-142.
47. Sheh R. et al. Extracting terrain features from range images for autonomous random stepfield traversal // 2007 IEEE International Workshop on Safety, Security and Rescue Robotics. – IEEE, 2007. – P. 1-6.

48. Sheh R., Hengst B., Sammut C. Behavioural cloning for driving robots over rough terrain // 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. – IEEE, 2011. – P. 732-737.
49. Sherman M. A., Seth A., Delp S. L. Simbody: multibody dynamics for biomedical research // Procedia Iutam. – 2011. – Vol. 2. – P. 241-261.
50. Simakov N. et al. Modeling USAR maps for the collection of information on the state of the environment // 2019 12th International Conference on Developments in eSystems Engineering (DeSE). – IEEE, 2019. – P. 918-923.
51. Smith R. et al. Open dynamics engine. – 2005.
52. Soekhoe D., Putten P., Plaat A. On the impact of data set size in transfer learning using deep neural networks // International symposium on intelligent data analysis. – Springer, Cham, 2016. – P. 50-60.
53. Sokolov M. et al. Modelling a crawler-type UGV for urban search and rescue in Gazebo environment // Artificial Life and Robotics (ICAROB 2017), International Conference on. – 2017. – P. 360-362.
54. Sokolov M. et al. 3D modelling and simulation of a crawler robot in ROS/Gazebo // Proceedings of the 4th International Conference on Control, Mechatronics and Automation. – 2016. – P. 61-65.
55. Timperley C. S. et al. Crashing simulated planes is cheap: Can simulation detect robotics bugs early? // 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). – IEEE, 2018. – P. 331-342.
56. Yakovlev K., Baskin E., Hramoin I. Grid-based angle-constrained path planning // Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). – Springer, Cham, 2015. – P. 208-221.
57. Yu H. et al. A cascaded deep learning framework for real-time and robust grasp planning // 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO). – IEEE, 2019. – P. 1380-1386.
58. Yumbla F., Yumbla E. Q., Moon H. The bioloid GP robot with different configurations for simulation in V-REP controlled by the robot operating

system (ROS) // 2020 6th International Conference on Control, Automation and Robotics (ICCAR). – IEEE, 2020. – P. 54-58.

59. Willman J. Overview of pyqt5 // Modern PyQt. – Apress, Berkeley, CA, 2021. – P. 1-42.

60. Официальный сайт «Сервосила Инженер» [Электронный ресурс] https://www.servosila.com/ru/mobile-robots/

ПРИЛОЖЕНИЕ А

Код urdf ходовой части новой модели робота «Сервосила Инженер»

```
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">


<xacro:property name="PI" value="3.1415"/>
<xacro:property name="big_wheel_radius" value="0.1248"/>
<xacro:property name="wheel_length" value="${0.1*scale}"/>
<xacro:property name="wheel_mass" value="0.15"/>
<xacro:property name="scale" value="0.1"/>
<xacro:property name="inv_scale" value="0.01"/>

<xacro:macro name="inertial_macro"
                 params="x y z roll pitch yaw mass ixx ixy ixz iyy iyz izz">
        <inertial>
            <origin xyz="${x} ${y} ${z}" rpy="${roll} ${pitch} ${yaw}"/>
            <mass value="${mass}"/>
            <inertia ixx="${mass/12*(3*big_wheel_radius**2 + wheel_length**2)}" ixy="0.0" ixz="0.0"
                     iyy="${mass/12*(3*big_wheel_radius**2 + wheel_length**2)}" iyz="0.0"
                     izz="${mass/2*(big_wheel_radius**2)}"/>
        </inertial>
    </xacro:macro>

<xacro:macro name="wheel" params="name_link x y z roll pitch yaw radius wheel_length mass parent scale
scale_width turn">
        <link name="${name_link}">
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                    <cylinder radius='0.01' length="-0.0"/>
                </geometry>
            </collision>
            <xacro:inertial_macro x="0" y="0" z="0"
                                  roll="0" pitch="0" yaw="0" mass="${mass}"
                                  ixx="1" ixy="0.0" ixz="0.0" iyy="1" iyz="0.0" izz="1"/>
        </link>
        <joint name="joint_${name_link}" type="continuous">
            <parent link="${parent}"/>
            <child link="${name_link}"/>
            <origin xyz="${x} ${y} ${z}" rpy="${roll} ${pitch} ${yaw}"/>
            <axis xyz="0 0 1"/>
            <limit effort="140.0" lower="0.0" upper="0.0" velocity="14.0"/>
            <joint_properties damping="0.8" friction="1"/>
        </joint>

        <transmission name="tran_joint_${name_link}">
            <type>transmission_interface/SimpleTransmission</type>
            <joint name="joint_${name_link}">
                <hardwareInterface wheel_radius="0.1"
radius="0.1">hardware_interface/VelocityJointInterface</hardwareInterface>
            </joint>
            <actuator name="motor_joint_${name_link}">
                <mechanicalReduction>0.8</mechanicalReduction>
            </actuator>
        </transmission>

        <link name="${name_link}_cog">
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0"/>
                <geometry>
                  <mesh
filename="package://engineer_description/meshes/collision/wheel_cog_sharp_7_${turn}.dae" scale="${scale}
${scale_width} ${scale}"/>
                </geometry>
            </collision>
            <xacro:inertial_macro x="0" y="0" z="0"
                                  roll="0" pitch="0" yaw="0" mass="${mass}"
                                  ixx="1" ixy="0.0" ixz="0.0" iyy="1" iyz="0.0" izz="1"/>
        </link>

        <gazebo reference="${name_link}_cog">
```

```
            <selfCollide>false</selfCollide>
            <mu1 value="100.0"/>
            <mu2 value="50.0"/>
        </gazebo>

        <joint name="joint_${name_link}_cog" type="fixed">
            <parent link="${name_link}"/>
            <child link="${name_link}_cog"/>
            <origin xyz="0 0 0" rpy="${PI*0.5} 0 0"/>
            <axis xyz="0 0 1"/>
        </joint>
    </xacro:macro>

<xacro:wheel name_link="right1_base_link_wheel" x="${-1.855*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="1"/>
    <xacro:wheel name_link="left1_base_link_wheel" x="${-1.855*scale}" y="${1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="1"/>

<xacro:wheel name_link="right2_base_link_wheel" x="${-1.39375*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="0"/>
    <xacro:wheel name_link="left2_base_link_wheel" x="${-1.39375*scale}" y="${1.0*scale}" z="${-
0.3*scale}" roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="0"/>

<xacro:wheel name_link="right3_base_link_wheel" x="${-0.9325*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="2"/>
    <xacro:wheel name_link="left3_base_link_wheel" x="${-0.9325*scale}" y="${1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="2"/>

<xacro:wheel name_link="right4_base_link_wheel" x="${-0.47125*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="1"/>
    <xacro:wheel name_link="left4_base_link_wheel" x="${-0.47125*scale}" y="${1.0*scale}" z="${-
0.3*scale}" roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="1"/>

<xacro:wheel name_link="right5_base_link_wheel" x="${-0.01*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="0"/>
    <xacro:wheel name_link="left5_base_link_wheel" x="${-0.01*scale}" y="${1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="0"/>

<xacro:wheel name_link="right6_base_link_wheel" x="${0.45125*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="2"/>
    <xacro:wheel name_link="left6_base_link_wheel" x="${0.45125*scale}" y="${1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="2"/>

<xacro:wheel name_link="right7_base_link_wheel" x="${0.9125*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="1"/>
    <xacro:wheel name_link="left7_base_link_wheel" x="${0.9125*scale}" y="${1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="1"/>
```

```
<xacro:wheel name_link="right8_base_link_wheel" x="${1.37375*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="0"/>
    <xacro:wheel name_link="left8_base_link_wheel" x="${1.37375*scale}" y="${1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="0"/>

<xacro:wheel name_link="right9_base_link_wheel" x="${1.835*scale}" y="${-1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="2"/>
    <xacro:wheel name_link="left9_base_link_wheel" x="${1.835*scale}" y="${1.0*scale}" z="${-0.3*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI*0.5}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="base_link" scale="0.46" scale_width="0.39" turn="2"/>


    <xacro:wheel name_link="right1_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${3.32*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.26" scale_width="0.28" turn="0"/>
    <xacro:wheel name_link="left1_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${3.32*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.26" scale_width="0.28" turn="0"/>

    <xacro:wheel name_link="right2_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${2.975*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.285" scale_width="0.28" turn="1"/>
    <xacro:wheel name_link="left2_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${2.975*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.285" scale_width="0.28" turn="1"/>

    <xacro:wheel name_link="right3_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${2.55*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.31" scale_width="0.28" turn="2"/>
    <xacro:wheel name_link="left3_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${2.55*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.31" scale_width="0.28" turn="2"/>

    <xacro:wheel name_link="right4_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${2.125*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.335" scale_width="0.28" turn="0"/>
    <xacro:wheel name_link="left4_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${2.125*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.335" scale_width="0.28" turn="0"/>

    <xacro:wheel name_link="right5_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${1.7*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.36" scale_width="0.28" turn="1"/>
    <xacro:wheel name_link="left5_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${1.7*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.36" scale_width="0.28" turn="1"/>

    <xacro:wheel name_link="right6_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${1.275*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.385" scale_width="0.28" turn="2"/>
    <xacro:wheel name_link="left6_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${1.275*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.385" scale_width="0.28" turn="2"/>

    <xacro:wheel name_link="right7_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${0.85*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.41" scale_width="0.28" turn="0"/>
```

```
    <xacro:wheel name_link="left7_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${0.85*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                 radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.41" scale_width="0.28" turn="0"/>

    <xacro:wheel name_link="right8_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${0.425*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                 radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.435" scale_width="0.28" turn="1"/>
    <xacro:wheel name_link="left8_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${0.425*scale}"
roll="0" pitch="${PI*0.5}" yaw="${PI}"
                 radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.435" scale_width="0.28" turn="1"/>

    <xacro:wheel name_link="right9_flipper_wheel" y="${0*scale}" x="${0.06*scale}" z="${0*scale}" roll="0"
pitch="${PI*0.5}" yaw="${PI}"
                 radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="right_flipper" scale="0.46" scale_width="0.28" turn="2"/>
    <xacro:wheel name_link="left9_flipper_wheel" y="${0*scale}" x="${-0.06*scale}" z="${0*scale}" roll="0"
pitch="${PI*0.5}" yaw="${PI}"
                 radius="${big_wheel_radius}" wheel_length="${wheel_length}" mass="${wheel_mass}"
parent="left_flipper" scale="0.46" scale_width="0.28" turn="2"/>


</robot>
```

ПРИЛОЖЕНИЕ Б

Параметры контроллера diff_drive новой модели робота «Сервосила Инженер»

```
mobile_base_controller:
  type: diff_drive_controller/DiffDriveController
  publish_rate: 50
  left_wheel:
['joint_left1_base_link_wheel','joint_left2_base_link_wheel','joint_left3_base_link_wheel','joint_left4_ba
se_link_wheel','joint_left5_base_link_wheel','joint_left6_base_link_wheel','joint_left7_base_link_wheel','
joint_left8_base_link_wheel','joint_left9_base_link_wheel','joint_left1_flipper_wheel','joint_left2_flippe
r_wheel','joint_left3_flipper_wheel','joint_left4_flipper_wheel','joint_left5_flipper_wheel','joint_left6_
flipper_wheel','joint_left7_flipper_wheel','joint_left8_flipper_wheel','joint_left9_flipper_wheel']
  right_wheel:
['joint_right1_base_link_wheel','joint_right2_base_link_wheel','joint_right3_base_link_wheel','joint_right
4_base_link_wheel','joint_right5_base_link_wheel','joint_right6_base_link_wheel','joint_right7_base_link_w
heel','joint_right8_base_link_wheel','joint_right9_base_link_wheel','joint_right1_flipper_wheel','joint_ri
ght2_flipper_wheel','joint_right3_flipper_wheel','joint_right4_flipper_wheel','joint_right5_flipper_wheel'
,'joint_right6_flipper_wheel','joint_right7_flipper_wheel','joint_right8_flipper_wheel','joint_right9_flip
per_wheel']
  pose_covariance_diagonal: [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
  twist_covariance_diagonal: [0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
  base_frame_id: engineer/base_footprint
  enable_odom_tf: true
  odom_frame_id: engineer/odom
  wheel_radius: 0.06256
  linear:
    x:
      has_velocity_limits: true
      max_velocity: 0.4  # m/s
      min_velocity: -0.4 # m/s
      has_acceleration_limits: true
      max_acceleration: 0.2  # m/s^2
      min_acceleration: -0.2 # m/s^2
      has_jerk_limits: true
      max_jerk: 0.30  # m/s^3
  angular:
    z:
      has_velocity_limits: true
      max_velocity: 0.5  # rad/s
      has_acceleration_limits: true
      max_acceleration: 0.5  # rad/s^2
      has_jerk_limits: true
      max_jerk: 0.2  # rad/s^3
```

ПРИЛОЖЕНИЕ В

Код логического модуля генератора RSE

```
#!/usr/bin/env python
# author: Ruslan Gabdrahmanov
import math
import os
from datetime import datetime

import numpy

from strings import mtl, config, sdf, vertex_str, import_xml_str


class Generator:
    def __init__(self, size, divided, chunk_count, modeling="single model", height_step=0, obstacles=None,
                 first_half=True):
        if obstacles is None:
            obstacles = []
        self.size = size
        if divided == "no (one piece)":
            self.division = 0
        elif divided == "yes (divided)":
            self.division = 2
        else:
            self.division = 1
        self.chunk_count = chunk_count
        self.modeling = modeling
        self.height_step = height_step / 100
        self.scaled_height_step = self.height_step / 10
        self.obstacles = obstacles
        self.first_half = first_half
        try:
            self.base = open('surroundings', 'r')
            self.block = open('block', 'r')
            if not os.path.isdir('generated'):
                os.mkdir('generated')
            self.save_dir = 'generated'
        except FileNotFoundError:
            self.base = open('engineer_gazebo/world/utils/test-rse-gen/surroundings', 'r')
            self.block = open('engineer_gazebo/world/utils/test-rse-gen/block', 'r')
            if not os.path.isdir('engineer_gazebo/world/utils/test-rse-gen/generated'):
                os.mkdir('engineer_gazebo/world/utils/test-rse-gen/generated')
            self.save_dir = 'engineer_gazebo/world/utils/test-rse-gen/generated'
        if not os.path.isdir(self.save_dir + '/models'):
            os.mkdir(self.save_dir + '/models')
        self.base_str = self.base.read()
        self.block_str = self.block.read()
        self.full_size = (self.size + self.division) * self.chunk_count - 1 * self.division
        self.block_hw = 0.1
        self.base_vertex = [[self.block_hw, 0.000000, 0.000000],
                            [self.block_hw, 0.000000, self.height_step],
                            [0.000000, 0.000000, self.height_step],
                            [0.000000, 0.000000, 0.000000],
                            [self.block_hw, self.block_hw, 0.000000],
                            [self.block_hw, self.block_hw, self.height_step],
                            [0.000000, self.block_hw, self.height_step],
                            [0.000000, self.block_hw, 0.000000]]

    def generate(self, i):
        """
        Run generation
        """
        matrix = numpy.zeros((self.full_size, self.full_size))
        for obstacle in self.obstacles:
            gen_function = obstacle["gen_function"]
            if gen_function == "Random Gaussian":
                matrix = self.normal_matrix(matrix, obstacle)
            elif gen_function == "Random":
                matrix = self.random_matrix(matrix, obstacle)
            elif gen_function == "Diagonal":
```

```
                matrix = self.diagonal_matrix(matrix, obstacle)
            elif gen_function == "Horizontal":
                matrix = self.diagonal_matrix(matrix, obstacle, True)
            elif gen_function == "Peak":
                matrix = self.peak_matrix(matrix, obstacle)
            else:
                return None
        xml_str = ''
        #print(matrix)

        if self.modeling == "single model":
            prefix = f"""# File Created: {datetime.now()}
mtllib rse.mtl
"""
            xml_str = import_xml_str(i)
            obj_str, flat_str = "", ""
            for x in range(self.full_size):  # line
                for y in range(self.full_size):  # column
                    obj_str, flat_str = self.set_obj_block(obj_str, flat_str, matrix, x, y)
            if self.division > 0:
                obj_str, flat_str = self.add_obj_borders(obj_str, flat_str)
            name = self.save_dir + "/models/test_rse" + str(i) + "/meshes/rse.obj"
            mtl_name = self.save_dir + "/models/test_rse" + str(i) + "/meshes/rse.mtl"
            sdf_name = self.save_dir + "/models/test_rse" + str(i) + "/model.sdf"
            config_name = self.save_dir + "/models/test_rse" + str(i) + "/model.config"
            if not os.path.isdir(self.save_dir + '/models/test_rse' + str(i)):
                os.mkdir(self.save_dir + '/models/test_rse' + str(i))
            if not os.path.isdir(self.save_dir + '/models/test_rse' + str(i) + "/meshes"):
                os.mkdir(self.save_dir + '/models/test_rse' + str(i) + "/meshes")
            res = open(name, 'w')
            res.writelines(prefix + obj_str + vertex_str + flat_str)
            res = open(mtl_name, 'w')
            res.writelines(mtl)
            res = open(sdf_name, 'w')
            res.writelines(sdf(i))
            res = open(config_name, 'w')
            res.writelines(config)
        elif self.modeling == "multi model":
            if self.division > 0:
                xml_str = self.add_borders(xml_str)
            for x in range(self.full_size):  # line
                for y in range(self.full_size):  # column
                    xml_str = self.set_block(xml_str, matrix, x, y)
        name = self.save_dir + "/test_rse" + str(i) + ".sdf"
        res = open(name, 'w')
        res.writelines(self.base_str.replace('###models###', xml_str))

    def normal_matrix(self, matrix, obstacle):
        """
        Normalized random obstacle
        """
        strength = obstacle["strength"]
        limit = numpy.max(matrix)
        limit = strength * 10 if limit < strength * 10 else limit
        summ = self.size + self.division
        strength2 = (strength + 1)/2
        for line in range(self.chunk_count):
            for column in range(self.chunk_count):
                matrix[line * summ:line * summ + self.size,
                column * summ:column * summ + self.size] = numpy.clip(
                    matrix[line * summ:line * summ + self.size,
                           column * summ:column * summ + self.size] + numpy.round(numpy.random.normal(
                    strength2, strength2, (self.size, self.size))) * 10, 0, limit)
        return matrix

    def random_matrix(self, matrix, obstacle):
        """
        Random obstacle
        """
        strength = obstacle["strength"]
        limit = numpy.max(matrix)
        limit = strength * 10 if limit < strength * 10 else limit
        summ = self.size + self.division
        for line in range(self.chunk_count):
            for column in range(self.chunk_count):
                matrix[line * summ:line * summ + self.size,
                column * summ:column * summ + self.size] = numpy.clip(
```

```
                    matrix[line * summ:line * summ + self.size,
                           column * summ:column * summ + self.size] + numpy.random.random_integers(
                    0, strength, (self.size, self.size)) * 10, 0, limit)
        return matrix

    def diagonal_matrix(self, matrix, obstacle, horizontal=False):
        """
        Diagonal obstacle
        """
        left_first = obstacle["left_first"] if obstacle["left_first"] < obstacle["left_second"] else
obstacle[
            "left_second"]
        left_second = obstacle["left_second"] if obstacle["left_first"] < obstacle["left_second"] else
obstacle[
            "left_first"]
        if not horizontal:
            right_first = obstacle["right_first"] if obstacle["right_first"] < obstacle["right_second"]
else obstacle[
                "right_second"]
            right_second = obstacle["right_second"] if obstacle["right_first"] < obstacle["right_second"]
else obstacle[
                "right_first"]
        strength = obstacle["strength"]
        softness = obstacle["softness"] / 100 / self.chunk_count
        summ = self.size + self.division
        mid_height = (strength * 10)
        if not horizontal:
            diagonal = numpy.array([numpy.random.random_integers(left_first - 1, left_second - 1),
                                    numpy.random.random_integers(right_first - 1, right_second - 1)])
        else: #  horizontal obstacle is diagonal with same x or y coords
            y = numpy.random.random_integers(left_first - 1, left_second - 1)
            diagonal = numpy.array([y, y])
        diagonal = numpy.array([[diagonal[0], 0], [diagonal[1], self.full_size - 1]])
        for line in range(self.chunk_count):
            for column in range(self.chunk_count):
                for a in range(line * summ, line * summ + self.size):
                    for b in range(column * summ, column * summ + self.size):
                        dist = numpy.linalg.norm(
                            (
                                numpy.cross(
                                    diagonal[1] - diagonal[0], diagonal[0] - [a, b]
                                )
                            ) / numpy.linalg.norm(
                                diagonal[1] - diagonal[0]
                            ))
                        if softness != 0:
                            inv_height = dist / summ / self.chunk_count * mid_height / softness
                            inv_height = inv_height - inv_height % 10
                        else:
                            inv_height = 0 if dist == 0 else mid_height

                        matrix[a][b] = round(max(
                            matrix[a][b],
                            mid_height - inv_height if not (
                                    math.isnan(inv_height) or math.isinf(inv_height)) else 0
                        ), obstacle["strength"])
        return matrix

    def peak_matrix(self, matrix, obstacle):
        """
        Peak obstacle
        """
        left_first = obstacle["left_first"] if obstacle["left_first"] < obstacle["left_second"] else
obstacle[
            "left_second"]
        left_second = obstacle["left_second"] if obstacle["left_first"] < obstacle["left_second"] else
obstacle[
            "left_first"]
        right_first = obstacle["right_first"] if obstacle["right_first"] < obstacle["right_second"] else
obstacle[
            "right_second"]
        right_second = obstacle["right_second"] if obstacle["right_first"] < obstacle["right_second"] else
obstacle[
            "right_first"]
        strength = obstacle["strength"]
        softness = obstacle["softness"] / 100 / self.chunk_count
        mid_height = (strength * 10)
```

```
        summ = self.size + self.division
        x = numpy.random.random_integers(left_first - 1, left_second - 1)
        y = numpy.random.random_integers(right_first - 1, right_second - 1)
        points = numpy.array([[x, y], [x-1, y], [x, y-1], [x-1, y-1]])
        for line in range(self.chunk_count):
            for column in range(self.chunk_count):
                for a in range(line * summ, line * summ + self.size):
                    for b in range(column * summ, column * summ + self.size):
                        min_dist = self.size
                        for point in points:
                            dist = numpy.linalg.norm(point - [a, b])
                            if dist % 1 != 0:
                                if dist % 1 > 0.49:
                                    dist = dist //  1 + 1
                                else:
                                    dist = dist //  1
                            if dist < min_dist:
                                min_dist = dist
                        if softness != 0:
                            inv_height = min_dist / summ / self.chunk_count * mid_height / softness
                            inv_height = inv_height - inv_height % 10
                        else:
                            inv_height = 0 if min_dist == 0 else mid_height
                        matrix[a][b] = round(max(
                            matrix[a][b],
                            mid_height - inv_height if not (
                                    math.isnan(inv_height) or math.isinf(inv_height)) else 0
                        ), obstacle["strength"])
        return matrix

    def add_borders(self, xml_str):
        """
        Create frame around entire RSE in xml
        """
        xml_str += self.block_str.replace(
            'NAME', 'border_right').replace(
            'X', str(self.full_size / 20.0 - 0.05)).replace(
            'Y', str(-0.1)).replace(
            'S', str(self.height_step)).replace(
            'A', str(self.full_size / 10.0)).replace(
            'B', str(0.1)).replace(
            'VP', str(self.height_step / 2))
        xml_str += self.block_str.replace(
            'NAME', 'border_left').replace(
            'X', str(self.full_size / 20.0 - 0.05)).replace(
            'Y', str(self.full_size / 10)).replace(
            'S', str(self.height_step)).replace(
            'A', str(self.full_size / 10.0)).replace(
            'B', str(0.1)).replace(
            'VP', str(self.height_step / 2))
        xml_str += self.block_str.replace(
            'NAME', 'border_forward').replace(
            'X', str(-0.1)).replace(
            'Y', str(self.full_size / 20.0 - 0.05)).replace(
            'S', str(self.height_step)).replace(
            'A', str(0.1)).replace(
            'B', str(self.full_size / 10.0 + 0.2)).replace(
            'VP', str(self.height_step / 2))
        xml_str += self.block_str.replace(
            'NAME', 'border_back').replace(
            'X', str(self.full_size / 10.0)).replace(
            'Y', str(self.full_size / 20.0 - 0.05)).replace(
            'S', str(self.height_step)).replace(
            'A', str(0.1)).replace(
            'B', str(self.full_size / 10.0 + 0.2)).replace(
            'VP', str(self.height_step / 2))
        return xml_str

    def set_block(self, xml_str, matrix, x, y):
        """
        Set one block of RSE in xml
        """
        xml_str += self.block_str.replace(
            'NAME', 'block' + str(x) + str(y)).replace(
            'X', str(x / 10.0)).replace(
            'Y', str(y / 10.0)).replace(
```

```
            'S', str(matrix[x][y] * self.scaled_height_step if matrix[x][y] > 0 else
self.height_step)).replace(
            'A', str(0.1)).replace(
            'B', str(0.1)).replace(
            'VP', str(matrix[x][y] * self.scaled_height_step / 2))
        return xml_str

    def set_obj_block(self, obj_str, flat_str, matrix, x, y):
        """
        Set one block of RSE in mesh
        """
        round_height = round(matrix[x][y] * self.scaled_height_step, 6) if matrix[x][y] > 0 else
self.height_step / 2
        obj_str += f"""v {self.base_vertex[0][0] + x / 10} {self.base_vertex[0][1] + y / 10}
{self.base_vertex[0][2]}
v {self.base_vertex[1][0] + x / 10} {self.base_vertex[1][1] + y / 10} {round_height}
v {self.base_vertex[2][0] + x / 10} {self.base_vertex[2][1] + y / 10} {round_height}
v {self.base_vertex[3][0] + x / 10} {self.base_vertex[3][1] + y / 10} {self.base_vertex[3][2]}
v {self.base_vertex[4][0] + x / 10} {self.base_vertex[4][1] + y / 10} {self.base_vertex[4][2]}
v {self.base_vertex[5][0] + x / 10} {self.base_vertex[5][1] + y / 10} {round_height}
v {self.base_vertex[6][0] + x / 10} {self.base_vertex[6][1] + y / 10} {round_height}
v {self.base_vertex[7][0] + x / 10} {self.base_vertex[7][1] + y / 10} {self.base_vertex[7][2]}
"""
        number_count = (y + x * self.full_size) * 8
        flat_str += f"""f {number_count + 1}//1 {number_count + 2}//1 {number_count + 3}//1 {number_count
+ 4}//1
f {number_count + 5}//2 {number_count + 8}//2 {number_count + 7}//2 {number_count + 6}//2
f {number_count + 1}//3 {number_count + 5}//3 {number_count + 6}//3 {number_count + 2}//3
f {number_count + 2}//4 {number_count + 6}//4 {number_count + 7}//4 {number_count + 3}//4
f {number_count + 3}//5 {number_count + 7}//5 {number_count + 8}//5 {number_count + 4}//5
f {number_count + 5}//6 {number_count + 1}//6 {number_count + 4}//6 {number_count + 8}//6
"""
        return obj_str, flat_str

    def add_obj_borders(self, obj_str, flat_str):
        """
        Create frame around entire RSE in mesh
        """
        obj_str += f"""v {self.block_hw + self.full_size / 10} {-self.block_hw} {0}
v {self.block_hw + self.full_size / 10} {-self.block_hw} {self.height_step}
v {0} {-self.block_hw} {self.height_step}
v {0} {-self.block_hw} {0}
v {self.block_hw + self.full_size / 10} {0} {0}
v {self.block_hw + self.full_size / 10} {0} {self.height_step}
v {0} {0} {self.height_step}
v {0} {0} {0}
"""
        number_count = (self.full_size * self.full_size) * 8
        flat_str += f"""f {number_count + 1}//1 {number_count + 2}//1 {number_count + 3}//1 {number_count
+ 4}//1
f {number_count + 5}//2 {number_count + 8}//2 {number_count + 7}//2 {number_count + 6}//2
f {number_count + 1}//3 {number_count + 5}//3 {number_count + 6}//3 {number_count + 2}//3
f {number_count + 2}//4 {number_count + 6}//4 {number_count + 7}//4 {number_count + 3}//4
f {number_count + 3}//5 {number_count + 7}//5 {number_count + 8}//5 {number_count + 4}//5
f {number_count + 5}//6 {number_count + 1}//6 {number_count + 4}//6 {number_count + 8}//6
"""
        obj_str += f"""v {self.block_hw + self.full_size / 10} {0} {0}
v {self.block_hw + self.full_size / 10} {0} {self.height_step}
v {self.full_size / 10} {0} {self.height_step}
v {self.full_size / 10} {0} {0}
v {self.block_hw + self.full_size / 10} {self.block_hw + self.full_size / 10} {0}
v {self.block_hw + self.full_size / 10} {self.block_hw + self.full_size / 10} {self.height_step}
v {self.full_size / 10} {self.block_hw + self.full_size / 10} {self.height_step}
v {self.full_size / 10} {self.block_hw + self.full_size / 10} {0}
"""
        number_count += 8
        flat_str += f"""f {number_count + 1}//1 {number_count + 2}//1 {number_count + 3}//1 {number_count
+ 4}//1
f {number_count + 5}//2 {number_count + 8}//2 {number_count + 7}//2 {number_count + 6}//2
f {number_count + 1}//3 {number_count + 5}//3 {number_count + 6}//3 {number_count + 2}//3
f {number_count + 2}//4 {number_count + 6}//4 {number_count + 7}//4 {number_count + 3}//4
f {number_count + 3}//5 {number_count + 7}//5 {number_count + 8}//5 {number_count + 4}//5
f {number_count + 5}//6 {number_count + 1}//6 {number_count + 4}//6 {number_count + 8}//6
"""
        obj_str += f"""v {self.full_size / 10} {self.full_size / 10} {0}
v {self.full_size / 10} {self.full_size / 10} {self.height_step}
v {-self.block_hw} {self.full_size / 10} {self.height_step}
```

```
v {-self.block_hw} {self.full_size / 10} {0}
v {self.full_size / 10} {self.block_hw + self.full_size / 10} {0}
v {self.full_size / 10} {self.block_hw + self.full_size / 10} {self.height_step}
v {-self.block_hw} {self.block_hw + self.full_size / 10} {self.height_step}
v {-self.block_hw} {self.block_hw + self.full_size / 10} {0}
"""
        number_count += 8
        flat_str += f"""f {number_count + 1}//1 {number_count + 2}//1 {number_count + 3}//1 {number_count
+ 4}//1
f {number_count + 5}//2 {number_count + 8}//2 {number_count + 7}//2 {number_count + 6}//2
f {number_count + 1}//3 {number_count + 5}//3 {number_count + 6}//3 {number_count + 2}//3
f {number_count + 2}//4 {number_count + 6}//4 {number_count + 7}//4 {number_count + 3}//4
f {number_count + 3}//5 {number_count + 7}//5 {number_count + 8}//5 {number_count + 4}//5
f {number_count + 5}//6 {number_count + 1}//6 {number_count + 4}//6 {number_count + 8}//6
"""
        obj_str += f"""v {0} {-self.block_hw} {0}
v {0} {-self.block_hw} {self.height_step}
v {-self.block_hw} {-self.block_hw} {self.height_step}
v {-self.block_hw} {-self.block_hw} {0}
v {0} {self.full_size / 10} {0}
v {0} {self.full_size / 10} {self.height_step}
v {-self.block_hw} {self.full_size / 10} {self.height_step}
v {-self.block_hw} {self.full_size / 10} {0}
"""
        number_count += 8
        flat_str += f"""f {number_count + 1}//1 {number_count + 2}//1 {number_count + 3}//1 {number_count
+ 4}//1
f {number_count + 5}//2 {number_count + 8}//2 {number_count + 7}//2 {number_count + 6}//2
f {number_count + 1}//3 {number_count + 5}//3 {number_count + 6}//3 {number_count + 2}//3
f {number_count + 2}//4 {number_count + 6}//4 {number_count + 7}//4 {number_count + 3}//4
f {number_count + 3}//5 {number_count + 7}//5 {number_count + 8}//5 {number_count + 4}//5
f {number_count + 5}//6 {number_count + 1}//6 {number_count + 4}//6 {number_count + 8}//6
"""
        return obj_str, flat_str
```