«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	09.03.04 - Программная инженерия
Профиль	Разработка программно-информационных систем
Факультет	КТИ
Кафедра	МО ЭВМ

К защите допустить
Зав. кафедрой Кринкин К.В.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: ИССЛЕДОВАНИЕ АЛГОРИТМОВ ЛОКАЛЬНОГО ПЛАНИРОВАНИЯ ТРАЕКТОРИЙ КОЛЕСНЫХ РОБОТОВ

Студент(ка)		_____ *подпись*	Гришин К. И.
Руководитель	к. т. н., доцент *(Уч. степень, уч. звание)*	_____ *подпись*	Кринкин К. В.
Консультанты	асс. каф. МО ЭВМ *(Уч. степень, уч. звание)*	_____ *подпись*	Чайка К. В.
	к. т. н., доцент *(Уч. степень, уч. звание)*	_____ *подпись*	Иванов А. Н.

Санкт-Петербург
2022

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю

Зав. кафедрой МО ЭВМ

_____ Кринкин К. В.

«___»_____2022 г.

Студент Гришин К. И. Группа 8303

Тема работы: Исследование алгоритмов локального планирования траекторий колесных роботов

Место выполнения ВКР: СПбГЭТУ «ЛЭТИ» кафедра МО ЭВМ

Исходные данные (технические требования):

Провести исследование алгоритмов локального планирования на роботе с дифференциальным приводом, имеющим лазерный сенсор ограниченной видимости.

Содержание ВКР:

Введение, Классификация алгоритмов планирования траекторий, Обзор алгоритмов локального планирования, Среда для исполнения алгоритмов, Сравнение алгоритмов, Заключение.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал

Дополнительные разделы: Безопасность жизнедеятельности

Дата выдачи задания «22» апреля 2022 г.

Дата представления ВКР к защите «___»_____20___ г.

Студент _____ Гришин К. И.
подпись

Руководитель к.т.н, доцент _____ Кринкин К. В.
(Уч. степень, уч. звание) *подпись*

Консультант асс. каф. МО ЭВМ _____ Чайка К. В.
(Уч. степень, уч. звание) *подпись*

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ Кринкин К.В.
«___»_____2022 г.

Студент Гришин К.И. Группа 8303

Тема работы: Исследование алгоритмов локального планирования траекторий колесных роботов

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	20.04 – 25.04
2	Обзор алгоритмов локального планирования	26.04 – 1.05
3	Разработка среды для запуска алгоритмов	2.05 – 9.05
4	Сравнение алгоритмов	10.05 – 15.05
5	Безопасность жизнедеятельности	15.05 – 18.05
6	Оформление пояснительной записки	19.05 – 24.05
7	Оформление иллюстративного материала	25.05 – 1.06
8	Предзащита	28.05

Студент _____ Гришин К. И.
подпись

Руководитель к.т.н, доцент _____ Кринкин К. В.
(Уч. степень, уч. звание) *подпись*

Консультант асс. каф. МО ЭВМ _____ Чайка К. В.
(Уч. степень, уч. звание) *подпись*

РЕФЕРАТ

Пояснительная записка 67 стр., 31 рис., 15 табл., 26 ист.

МОБИЛЬНЫЙ РОБОТ, КОЛЕСНЫЙ РОБОТ, ЛОКАЛЬНОЕ ПЛАНИРОВАНИЕ, DYNAMIC WINDOW APPROACH, TIMED-ELASTIC-BAND, TRAJECTORY ROLLOUT, MODEL PREDICTIVE CONTROL.

Объектом исследования являются алгоритмы локального планирования мобильных роботов.

Предмет исследования — траектории движения робота, составленные алгоритмами локального планирования.

Цель данной работы: Провести оценку алгоритмов локального планирования траекторий для робота с дифференциальным приводом и ограниченной видимостью.

Для автоматизации движения мобильных роботов необходимо решить три основные задачи: выбор глобального маршрута, определение своего положения в пространстве, выбор локального маршрута и построение траектории движения. В данной работе подробно разобраны различные способы решения последней задачи, а именно локального планирования. Выявлены критерии оценки методов локального планирования. Произведено сравнение различных по своей структуре и идее методов.

ABSTRACT

Mobile robot movement automation consists of three main tasks: Global route selection; robot position determination; local route selection and building a movement trajectory. In this work, various methods for solving the local route selection, otherwise – local planning or motion planning are analyzed in detail. Criteria for evaluating methods of local planning have been defined. Methods different in their structure and idea are compared.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ

Глобальное планирование — поиск пути в пространстве от одной точки до другой используя статическую карту местности.

Локальное планирование — поиск траектории движения в пространстве в непосредственной близости робота путем использования различных бортовых сенсоров.

Мобильный робот — это автоматический механизм, способный перемещаться в окружающем пространстве, не привязанный к одной локации и выполняющий действия согласно интегрированной базе знаний.

Дифференциальный привод — привод в основе которого лежит два мотора, по одному на каждое колесо.

Визуализатор — программное средство, предназначенное для преобразования информации в зрительные образы.

DWA — Dynamic Window Approach

ROS — Robot Operating System

MPC — Model Predictive Control

TEB — Timed-Elastic-Band

TR — Trajectory Rollout

ВВЕДЕНИЕ

Навигация автономных мобильных роботов – быстроразвивающаяся область. За последние десятилетия произошел огромный рост данной отрасли. Автономные роботы или транспортные средства значительно снижают вклад человеческих ошибок. Роботы все чаще работают в закрытых помещениях, предназначенных для людей, в среде, в которой необходимо избегать неожиданные препятствия. Автономные роботы должны безопасно и эффективно перемещаться из точки *А* в точку *Б* с учетом времени, расстояния, энергии и других факторов.

Во время навигации, роботизированные системы запускают процессы, которые включают моделирование окружающей среды и локализацию положения системы в окружающей среде, управление движением, обнаружение и предотвращение препятствий, а также движение в динамических средах [1].

В автономной навигации выделяется три основные задачи: выбор глобального маршрута, определение своего положения, выбор локального маршрута [2]. Первая заключается в том, чтобы построить глобальный маршрут из одной точки в другую на статической карте; Вторая – точное определение местоположения робота в текущей среде. Последняя заключается в том, чтобы правильно вычислить команды, которые направятся роботу, для движения, чтобы отслеживать глобальный путь и избегать столкновений с препятствиями.

В данной работе уделено внимание локальным планировщикам, в частности реализациям и внедрению некоторых планировщиков в Robot Operating System (ROS) для роботов с дифференциальным приводом.

ROS – это фреймворк, широко используемый в сообществе робототехники. Его основная цель – сделать разработку ПО для роботов более гибкой. ROS — это набор инструментов, библиотек и соглашений, используемых для взаимодействия с роботизированными платформами.

В данной работе разобраны все, используемые в ROS, локальные планировщики, а также же те, которые скоро будут добавлены.

В качестве среды для испытаний, использовалось ПО для симуляции пространства и робота Gazebo. Данная платформа крайне популярна в ROS сообществе, имеет поддержку физики и полноценно интегрирована в ROS. Она представляет собой виртуальный мир, в котором расположен робот. Интерфейс взаимодействия с предоставляется ROS будто это настоящий робот.

Цель данной работы: Провести оценку алгоритмов локального планирования траекторий для робота с дифференциальным приводом и ограниченной видимостью.

Задачи данной работы:

1. Изучение существующих алгоритмов локального планирования для мобильных роботов.
2. Создание симуляции для исследования алгоритмов.
3. Определение метрик для сравнения алгоритмов.
4. Сравнение алгоритмов локального планирования по заданным метрикам.

Объектом исследования являются алгоритмы локального планирования мобильных роботов

Предмет исследования — траектории движения робота, составленные алгоритмами локального планирования.

Практическая ценность работы: Заранее составленная траектория движения робота не позволяет полностью автоматизировать перемещение. Окружение и препятствия постоянно изменяются, может ухудшаться

видимость или поверхность передвижения, в таком случае необходимо прибегать к методам локального планирования.

Необходимо провести анализ существующих алгоритмов для определения их быстродействия и применимости к маленьким маневренным колесным роботам.

Помимо программных ограничений, также существуют и аппаратные, устанавливаемые сенсоры могут иметь плохое разрешение или шумы, стоит учитывать стоимость оборудования для использования того или иного алгоритма.

В дополнение, готовая реализация алгоритма на реальном роботе позволит более удобно проводить дальнейшие исследования построения глобальных маршрутов и SLAM-алгоритмов.

1 КЛАССИФИКАЦИЯ АЛГОРИТМОВ ПЛАНИРОВАНИЯ ТРАЕКТОРИЙ

Исследования в области планирования пути автономных мобильных роботов привлекли внимание с 1970-х годов, особенно после 1979 года, когда начали развиваться современные подходы к планирование движения, тогда Лозано-Перес и Уэсли представили концепцию конфигурационного пространства (C-space) [3].

В последние десятилетия задача планирования движения становится все более и более важной, поскольку увеличивается роль роботов в промышленности, а также в нашей повседневной жизни. По большей части, проблема навигации роботов развивалась для нахождения оптимальных решений, касающихся планирования пути [4] (т. е. расчета пути, по которому необходимо следовать, чтобы достичь заданной цели) и управления движением [5] (т. е. расчета контрольных сигналов, отправляемых роботу, для отслеживания пути, избегая препятствия) с учетом кинематических и динамических ограничений робота.

Что касается управления движением (motion control), было реализовано множество алгоритмов для следования заданному пути. В данной работе представлены некоторые из них.

Глобально, алгоритмы планирования движений делятся на две большие категории: традиционные и на основе машинного обучения [6], в соответствие с их принципами и эпохой в которую были изобретены.

Традиционные алгоритмы состоят из четырех групп:

- Алгоритмы поиска по графу
- Сэмплинг-методы
- Интерполяционные кривые
- Реактивные алгоритмы

Алгоритмы, основанные на машинном обучении опираются на три подхода: обучение с учителем (*Supervised Learning*), обучением с подкреплением на основе Optimal Value (*Optimal Value RL*), обучение с подкреплением на основе *Policy Gradient*. В данной работе алгоритмы на основе машинного обучения рассмотрены не будут. Категории алгоритмов представлены на рисунке 1.

Рисунок 1. Категории алгоритмов планирования движений.

1.1 Алгоритмы поиска по графу

Являются наиболее старыми представителями алгоритмов планирования траекторий и требуют значительных модификаций для того, чтобы быть использованными в качестве локального планирования путей.

Алгоритмы на основе графов делятся на два типа – поиск в глубину и поиск в ширину. Алгоритм поиска в глубину строит дерево поиска как можно глубже от начальной вершины, пока путь не будет найден. Алгоритм поиска в ширину основан на наиболее быстром расширении дерева поиска, пока не будет достигнут необходимый путь.

Также поиск на графе можно дополнить различными весами в вершинах и ребрах. В таком случае процесс поиска пути управляется расчетом значений дерева поиска, которые позволяют определить: стоит ли расширять дерево поиска

и какую ветвь расширять. Наиболее популярными являются алгоритмы *Дейкстры* [7] и *A** [8].

1.2 Сэмплинг-методы

Сэмплинг-методы случайным образом разбивают доступное видимое пространство на выборку возможных маршрутов, которые затем объединяются, образуя собой субоптимальный пути. *Rapidly-exploring Random Tree* (RRT) и *Probalistic Roadmap Method* (PRM) чаще всего используются при планировании движения.

Алгоритм *RRT* [9-10] более популярен и широко используется в коммерческих и промышленных целях. Он строит дерево, которое быстро и равномерно исследует доступное пространство посредством случайного поиска. *RRT* также может учитывать неголономные ограничения.

Алгоритм *PRM* [11] используется в статическом сценарии. Он разделен на две фазы: фаза обучения и фаза запроса. На этапе обучения доступное пространство преобразуется в дорожную карту и сохраняется в виде графа, на этапе запроса ищется путь, соединяющий исходный и целевой узлы.

1.3 Интерполяционные кривые

Алгоритм интерполяционной кривой определяется как набор математических правил для построения траекторий. Интерполяционные кривые преобразуют ломанный путь глобального планировщика в плавную линию, которая затем преобразуется в команды роботу. Типичные правила сглаживания траекторий включают в себя: линию и окружность, клотоиду, кривые Безье, сплайн кривые. Примером такого алгоритма является *путь Дубинса* [12], который строит траекторию из прямых и окружностей. Данные алгоритмы не предназначены для построения пути в динамическом пространстве, однако часто используются совместно с другими алгоритмами для сглаживания траекторий.

1.4 Реактивные алгоритмы

В отличие от алгоритмов поиска по графу, которые требуют больше времени для планирования, реактивные алгоритмы предназначены для быстрого выполнения реакции или планирования локального пути. Реактивные алгоритмы состоят из двух подсистем: восприятие и движение. Исполнение таких алгоритмов сопровождается постоянными парами: сканирование-действие (sense-act). Полученные данные сенсоров вызывают простую реакцию у робота: поворот, следование пути или экстренная остановка [13]. Наиболее популярны *Artificial Potential Field* (APF) [14], *Dynamic Window Approach* (DWA) [15], *Timed-Elastic-Band* (TEB) [16].

Алгоритм потенциальных полей (*APF*) – Метод основанный на принципе взаимодействия электростатических частиц, а именно: целевая точка притягивает робота, а препятствия отталкивают. Величина притягивающей или отталкивающей силы, которая влияет на направление движения робота, определяется расстоянием до препятствия или цели. Алгоритм *APF* – имеет ряд значительных недостатков. (1) Робот не может продолжать движение, если оказался в точке локального минимума. Если поле сходится к минимуму, который не является глобальным (целью), то робот останавливается в точке и не может продолжать движение. (2) Колебания движения робота при прохождении между близкими препятствиями.

Dynamic Window Approach (*DWA*) – алгоритм основанный на скорости движения, который вычисляет оптимальную скорость робота, избегая препятствия, необходимую для достижения цели. То есть значения (x, y) целевой точки и значения текущих скоростей (v, w) транслируются в команду скорости (v, w).

Основные задачи алгоритма: рассчитать правильное окно поиска скорости и выбор оптимальную скорость. Пространство поиска строится из скоростей,

обеспечивающих безопасную траекторию, т. е. позволяющую остановиться в случае препятствия. Из скоростей берется выборка с учетом динамики робота («*dynamic window*»). Оптимальной является команда скорости, максимизирующая расстояние до препятствий и скорость, минимизирующая курс к цели.

Timed-Elastic-Band (*TEB*) – Работает на схожих с *APF* принципах. Как и *APF*, *TEB* имитирует физические силы при построении локального плана. Построенный глобальным планировщиком путь сглаживается с помощью применения искусственных сил: сила внутреннего сжатия и внешняя отталкивающая сила. Сила внутреннего сжатия имитирует натяжение растянутой резинки. Для преодолевания препятствий, применяется внешняя отталкивающая сила, исходящая от препятствий. Таким образом упругая лента, словно натягивается на препятствия. При прохождении робота через несколько препятствий, возможно несколько способов расположить такую упругую ленту. Производится процесс многокритериальной оптимизации с учетом времени.

1. 5 Вывод

Среди традиционных подходов к планированию локальной траектории не все подходят для локального планирования в неизвестной среде (см. табл. 1). Наиболее подходящими являются реактивные алгоритмы, которые будут рассмотрены.

Таблица 1. Оценка классов традиционных алгоритмов

Класс алгоритмов	***Планирование***	***Обход препятствий***	***Скорость реакции***	***Необходимая видимость***
Поиск по графу	Глобальное	Да	Низкая	Всей местности
Сэмплинг методы	Глобальное/ Локальное	Да	Низкая	Во всех направлениях
Интерполяционные кривые	Локальное	Нет	Средняя	–
Реактивные	Локальное	Да	Высокая	В направлении движения

2 ОБЗОР АЛГОРИТМОВ ЛОКАЛЬНОГО ПЛАНИРОВАНИЯ

В данной работе рассмотрено 4 различных реактивных алгоритмов, реагирующих на окружающую среду в режиме реального времени, постоянно дополняя существующий маршрут.

2.1 Dynamic Windows Approach

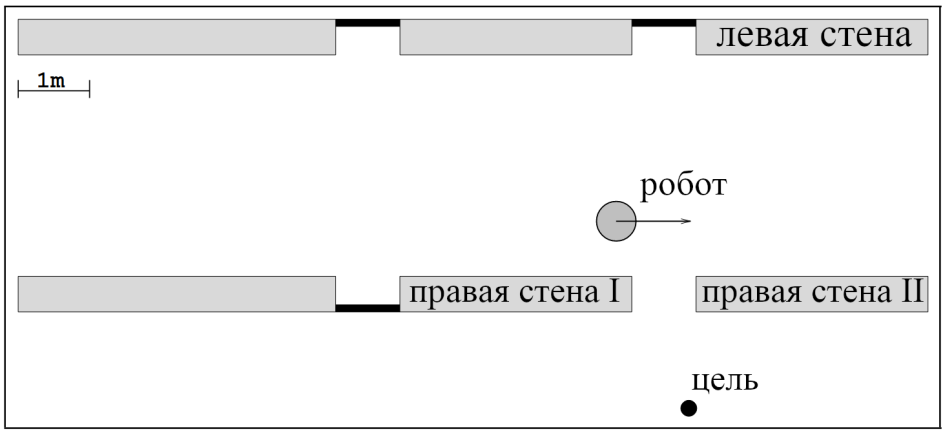

Рисунок 2. Пример расположения робота.

Выбор команды управления роботом осуществляется в пространстве скоростей. Динамика робота включена в метод за счет сокращения пространства поиска до тех скоростей, которые достижимы при динамических ограничениях. В дополнение к этому ограничению учитываются только скорости, безопасные по отношению к препятствиям. Это сокращение пространства поиска выполняется на первом этапе алгоритма. На втором этапе из оставшихся скоростей выбирается скорость, максимизирующая целевую функцию. Предположим робот, находится в ситуации, представленной на рисунке 2.

Для генерации траектории к заданной целевой точке для n интервалов времени, робот должен определить скорости (υ_i, ω_i), по одной для каждого из n интервалов между t_0 и t_n. Это должно быть сделано при условии, что результирующая траектория не пересекается с препятствием. Пространство поиска этих векторов экспоненциально по числу рассматриваемых интервалов.

Для реализации данного подхода, *Dynamic Window Approach* рассматривает только первый интервал времени, допуская то, что скорости в остальных временных интервалах постоянны, что эквивалентно нулевому расстоянию. Данное ограничение мотивировано тем, что: (1) пространство скоростей пересоздается через каждый промежуток времени; (2) если новые команды не даны (при недостатке аппаратных ресурсов), скорости остаются прежними.

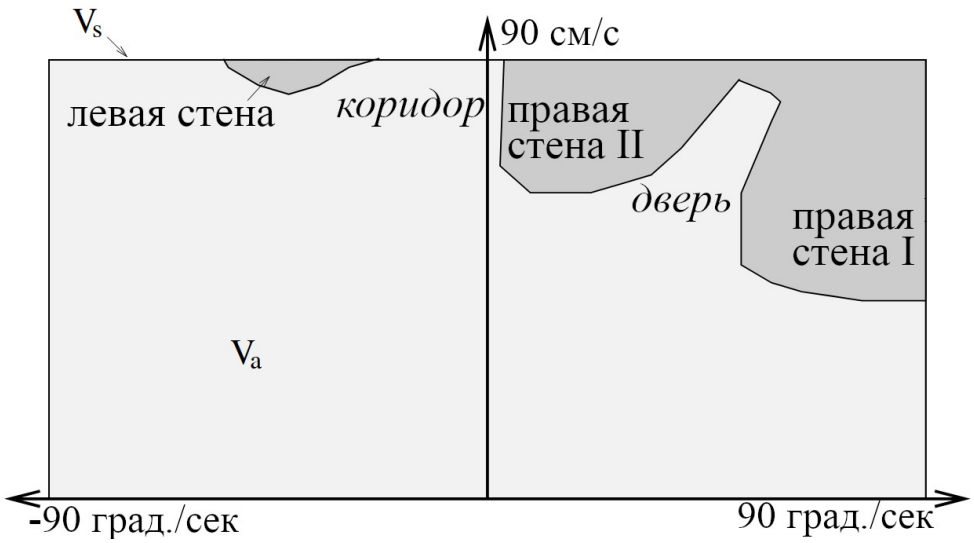

Рисунок 3. Пространство возможных скоростей.

Препятствия находящиеся в непосредственной близости с роботом накладывают ограничение на скорость движения и поворота (рис. 3). Максимальная допустимая скорость робота при движения по дуге ограничивается расстоянием до препятствия, находящегося на этой дуге. Примем для скорости (υ,ω), которую можно интерпретировать как дугу значение $dist(\upsilon,\omega)$ равное расстоянию до ближайшего препятствия на соответствующей дуге. Тогда скорость принимается достижимой, если робот может остановиться до того, как достигнет препятствия. Допустим $\dot{\upsilon}_b$ и $\dot{\omega}_b$ – ускорения торможения, в таком случае набор допустимых скоростей V_a определяется как:

$$V_a = \left\{ (\upsilon,\omega) \,|\, \upsilon \le \sqrt{2 \cdot dist(\upsilon,\omega) \cdot \dot{\upsilon}_b} \wedge \omega \le \sqrt{2 \cdot dist(\upsilon,\omega) \cdot \dot{\omega}_b} \right\}$$

Таким образом, V_a – набор скоростей (υ, ω), которые не допускают ситуации в которой робот не успеет остановиться перед препятствием.

Чтобы принять во внимание ограничения двигателей робота, пространство поиска скоростей сводится к динамическому окну (*Dynamic Window*), которое содержит только те скорости, которые могут быть достигнуты в течение следующего интервала времени. Пусть t будет интервалом времени, в течение которого будут приложены ускорения $\dot{\upsilon}$ и $\dot{\omega}$, а (υ_a, ω_a) – текущая скорость. В таком случае динамическое окно V_d определяется как (рис. 4):

$$V_d = \left\{ (\upsilon, \omega) \middle| \upsilon \in \left[\upsilon_a - \dot{\upsilon} \cdot t \, , \, \upsilon_a + \dot{\upsilon} \cdot t \right] \wedge \omega \in \left[\omega_a - \dot{\omega} \cdot t \, , \, \omega_a + \dot{\omega} \cdot t \right] \right\}$$

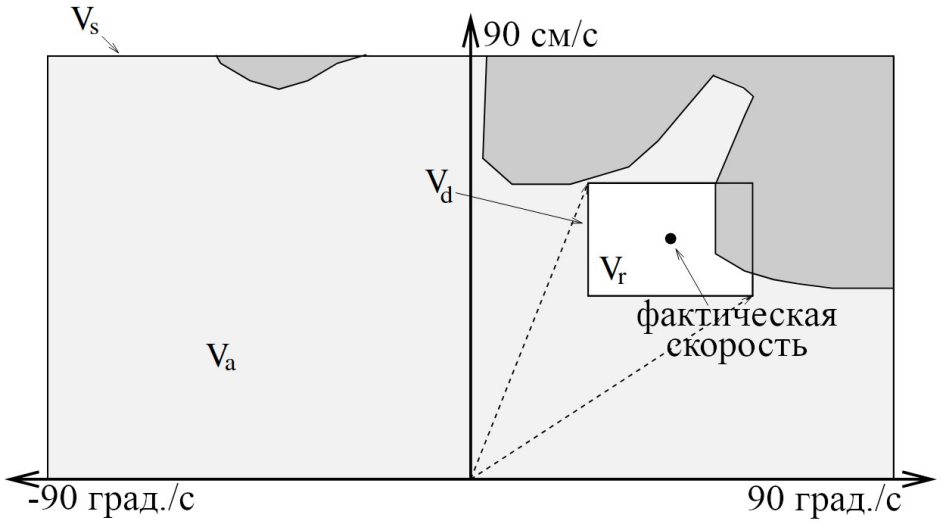

Рисунок 4. Динамическое окно

Динамическое окно сосредоточено вокруг текущей скорости, а его расширение зависит от возможных ускорений. Возможные траектории дуг за пределами динамического окна не могут быть достигнуты в течение следующего интервала времени и, следовательно, не учитываются при обходе препятствия.

Пересечение множеств V_a и V_d определяет результирующее пространство скоростей V_r (рис. 4), из которого выбирается команда скорости, направляемая роботу.

$$V_r = V_a \cap V_d$$

Для того, чтобы включить в критерий отбора подходящей скорости курс, препятствия и скорость, необходимо максимизировать функцию G над пространством V_r. Это выполняется благодаря дискретизации пространства V_r.

$$G(\upsilon,\omega)=\sigma(\alpha\cdot heading(\upsilon,\omega)+\beta\cdot dist(\upsilon,\omega)+\gamma\cdot velocity(\upsilon,\omega))$$

Направление (*heading*) является метрикой выравнивания робота относительно целевого направления. Определяется как $180-\theta$, где θ – угол между целевым направлением и направлением робота. Направление вычисляется относительно прогнозируемого расположения робота. Что обеспечивает плавный поворот к цели при ободе препятствия.

Расстояние до препятствия (*dist*) представляет собой расстояние до ближайшего препятствия, расположенного на траектории дуги. В случае отсутствия препятствия на такой траектории, значение устанавливается как большая константа.

Скорость (*velocity*) является оценкой продвижения робота по траектории и представляет собой υ.

Следует отметить, что все три компонента G — курс к цели, расстояние до препятствий и скорость — необходимы. Максимизируя только расстояние до препятствий и скорость, робот всегда будет перемещаться в свободное пространство, но у него не будет стимула двигаться к целевому местоположению. Исключительно максимизируя направление цели, робот быстро будет остановлен первым препятствием, преграждающим ему путь, и не сможет его обойти. Объединив все три компонента, робот максимально быстро избегает столкновений с учетом ограничений, перечисленных выше, и в то же время продвигается к своей цели.

2.2 Trajetory Rollout

В данном алгоритме применяется подход, отличающийся от таких алгоритмов как *DWA*. Для поиска подходящей команды скорости рассматривается не пространство траекторий (траектории дуг в *DWA*), а непосредственно пространство управляющих команд [17]. Робот управляется парой команд $(\dot{x}, \dot{\theta})$ – скорость перемещения и скорость поворота. Таким образом, имеется двухмерное пространство команд для рассмотрения.

Это пространство ограничено максимальными и минимальными значениями скоростей, отражающих возможности робота. Создается двумерная выборка из возможных скоростей, и для каждой команды определяется расположение робота в коротком промежутке времени. Такое моделирование предсказывает возможные траектории, как последовательность пятимерных состояний $(x, y, \theta, \dot{x}, \dot{\theta})$, с аппроксимацией динамики транспортного средства

Таким образом, смоделированые траектории, спроецированные на плоскость (x, y), представляют собой гладкие, непрерывные 2-мерные кривые, зависящие от пределов ускорения.

Каждая смоделированная траектория *t* оценивается следующей функцией стоимости:

$$C(t) = \alpha Obs + \beta Gdist + \gamma Pdist + \delta \frac{1}{\dot{x}^2}$$

Где, *Obs* – сумма стоимости ячеек сетки, через которые проходит траектория (учитывая те, в которые входят габариты робота); *Gdist* и *Pdist* – расстояния конченой точки траектории до целевой точки и начальной траекторией соответственно; $\dot{x}$ – поступательная составляющая команды скорости.

Выбирается траектория, минимизирующая *C(t)*. Тем самым наиболее предпочтительной является траектория, которая: (1) находится далеко от препятствий (*Obs*); (2) приближает к цели (*Gdist*); (3) остается вблизи глобаль-

ного пути (*Pdist*); (4) наибольшая допустимая скорость ($\dot{x}$). Траектории приводящие робота к столкновению сразу отбрасываются как недопустимые.

2.3 Timed Elastic Band

Данный алгоритм является расширением алгоритма «*Elastic Band*» [18].

Elastic-Band – алгоритм направленный на то, чтобы изменить изначальный глобальный маршрут в пределах робота таким образом, чтобы он плавно обходил препятсвтия. Основная идея «*Elastic Band*» состоит в том, чтобы деформировать изначально заданный путь, рассматривая его как эластичную упругую ленту, подверженную действию внутренних и внешних сил, которые уравновешивают друг друга в попытке сократит путь, обходя при этом препятствия.

«*Timed Elastic Band*» дополняет решение «*Elastic Band*» дополнительной информацией, позволяющей учитывать динамические ограничения робота и изменять траектории движения, вместо пути.

Классический «*Elastic Band*» описывается как последовательность из n промежуточных позиций робота $\mathbf{x_i}=(x_i, y_i, \beta_i)$, где x_i, y_i – точка расположения робота, а β_i – направление:

$$Q=\{\mathbf{x_i}\}_{i=0\ldots n} \quad n\in\mathbb{N}$$

«*Timed Elastic Band*» дополняется временными интервалами между двумя соседними конфигурациями, описывающимися последовательностью из n - 1 временных переходов ΔT_i: $\tau = \{\Delta T_i\}_{i=0\ldots n-1}$

Каждый временной переход означает, необходимое роботу для для перехода из одного состояние в другое (рис. 5). *TEB* описывается как набор векторов, состоящих из Q и τ:

$$B:=(Q,\tau)$$

Основной идея *TEB* заключается в одновременной оптимизации конфигураций и временных интервалов с помощью взвешенной многокритериальной оптимизации в реальном времени.

$$f(B)=\sum_k \gamma_k f_k(B)$$

$$B^* = \arg\min_B f(B)$$

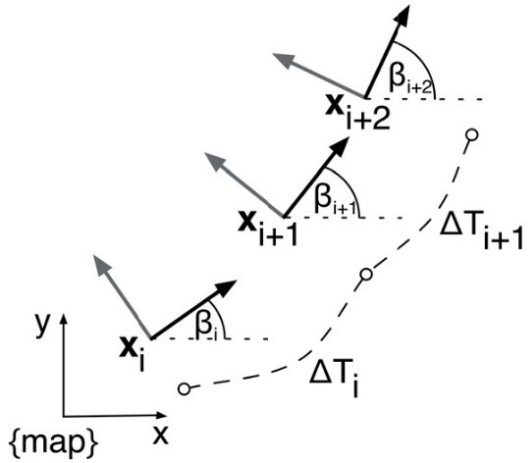

Рисунок 5. Состояния TEB.

Где B^* – оптимизированная *TEB*, $f(B)$ – целевая функция.

Большинство компонентов целевой функции локальны по отношению к B (оптимизированный «*Elastic Band*» маршрут), поскольку зависят от нескольких последовательных конфигураций, а не от всего пути. Такое свойство *TEB* позволяет представить систему как разреженную матрицу, для которых существуют эффективные методы численной оптимизации.

Целевые функции *TEB* относятся к двум типам: (1) ограничения, такие как пределы скорости и ускорения, представленные виде штрафных функций; (2) стремление к правильной траектории, предлагающей кратчайший (или быстрейший) путь, обтекающий препятствия.

TEB одновременно учитывает достижение промежуточных точек исходного пути и избежание статических или динамических препятствий. Обе целевые

функции аналогичны с той разницей, исходный путь притягивает «упругую ленту», а препятствия отталкивают ее.

Целевая функция *TEB* основана на минимальном расстоянии $d_{min,j}$ от генерируемой траектории до опорной точкой пути или препятствия $\mathbf{z}_j$. В случае опорных точек пути расстояние ограничено максимальным радиусом r_{pmax}, в случае препятствия ограничено минимальным расстоянием r_{omin}. Данные ограничения представляются следующими функциями:

$$f_{path} = e_{\Gamma}(d_{min,j}, r_{pmax}, \epsilon, S, n)$$

$$f_{ob} = e_{\Gamma}(-d_{min,j}, -r_{omin}, \epsilon, S, n)$$

Где e_{Γ} – кусочно-непрерывная дифференцируемая полиномиальная функция, аппроксимирующая границу:

$$e_{\Gamma}(x, x_r, \epsilon, S, n) \simeq \begin{cases} \left(\dfrac{x - (x_r - \epsilon)}{S} \right)^n & \text{if } x > x_r - \epsilon \\ 0 & \text{otherwise} \end{cases}$$

В которой x_r означает границу, а ϵ, S, n – определяют точность аппроксимации. А именно, S – масштабирование, n – полиномиальный порядок, ϵ – отклонение от аппроксимации.

Динамические ограничения на скорость и ускорение робота описываются похожими штрафными функциями как и в случае с геометрическими ограничениями (опорные точки маршрута и препятствия). Средние поступательная скорость и скорость поворота вычисляются в соответствие с евклидовым и угловым расстоянием между двумя соответствующими состояниями $\mathbf{x}_i$, $\mathbf{x}_{i+1}$ за временной промежуток ΔT_i.

$$v_i \simeq \frac{1}{\Delta T_i} \cdot \left\| \begin{matrix} x_{i+1} - x_i \\ y_{i+1} - y_i \end{matrix} \right\| \qquad \omega_i \simeq \frac{\beta_{i+1} - \beta_i}{\Delta T_i}$$

Несмотря на то, что робот движется по дуге, расчет расстояния как евклидова является достаточным приближением, поскольку конфигурации расположены

близко друг к другу. Для получения ускорения необходимо рассмотреть последовательные значения скоростей, следовательно рассматривается три последовательные конфигурации с двумя соответствующими временными интервалами:

$$a_i = \frac{2(\upsilon_{i+1} - \upsilon_i)}{\Delta T_i + \Delta T_{i+1}}$$

Ускорение поворота считается таким же образом, только вместо скорости поступательного движения используется скорость поворота.

Для мобильного робота с дифференциальным приводом связь между скоростями колес и парой поступательная-вращательная скорость центра робота устанавливается следующими выражениями:

$$\upsilon_{w_r,i} = \upsilon_i - L\,\omega_i$$
$$\upsilon_{w_l,i} = \upsilon_i - L\,\omega_i$$

Где L – половина расстояния между колесами.

Роботы с дифференциальным приводом обладают только двумя степенями свободы. Таким образом, они могут двигаться только в направлении текущего курса. Такое ограничение приводит к тому, что плавный путь строится из дуговых сегментов. Следовательно, два соседних состояния должны находиться на одной дуговой кривой (рис. 6). Угол ϑ_i между направлением состояния $\mathbf{x}_i$ и вектором $\mathbf{d}_{i,i+1}$, направленным от состояния $\mathbf{x}_i$ к состоянию $\mathbf{x}_{i+1}$, равен соответствующему углу ϑ_{i+1} между $\mathbf{x}_{i+1}$ и $\mathbf{d}_{i,i+1}$.

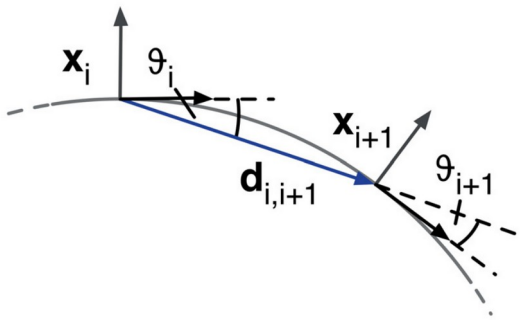

Рисунок 6. Взаимное расположение состояний при неголономном движении.

Если β_i – абсолютное направление, тогда:

$$\vartheta_i = \vartheta_{i+1} \Leftrightarrow$$

$$\Leftrightarrow \begin{pmatrix} \cos\beta_i \\ \sin\beta_i \\ 0 \end{pmatrix} \times \mathbf{d}_{i,i+1} = \mathbf{d}_{i,i+1} \times \begin{pmatrix} \cos\beta_{i+1} \\ \sin\beta_{i+1} \\ 0 \end{pmatrix}$$

Где вектор направления $\mathbf{d}_{i,i+1}$ равен

$$\mathbf{d}_{i,i+1} := \begin{pmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \\ 0 \end{pmatrix}$$

Тогда, целевая функция равна:

$$f_k(x_i, x_{i+1}) = \left\| \left[\begin{pmatrix} \cos\beta_i \\ \sin\beta_i \\ 0 \end{pmatrix} + \begin{pmatrix} \cos\beta_{i+1} \\ \sin\beta_{i+1} \\ 0 \end{pmatrix} \right] \times \mathbf{d}_{i,i+1} \right\|^2$$

Цель самого быстрого пути легко достигается путем минимизации квадрата суммы всех временных различий.

$$f_k = \left(\sum_{i=1}^{n} \Delta T_i\right)^2$$

Такая целевая функция позволяет добиться равномерного распределения состояний по времени, а не по расстоянию.

2.4 Model Predictive Control

Model Predictive Control (*MPC*) [19] – это продвинутый метод управления, работающий в дискретном времени. Из набора состояний относительно положения модели он оптимизирует проблему вокруг цели и дает последовательность управляющих сигналов в качестве выходных данных. Затем первый набор значений управляющих сигналов используется в качестве входных данных для управляемого объекта, и после короткого периода, установленного в качестве системного временного шага, измеряются новые значения состояния, и процесс повторяется.

Концепция *MPC* впервые была представлена в компании *Shell Oil* в 1979 году и называлась тогда «Динамический Матричный Контроль» (*DMC*, от англ. *Dynamic Matrix Control*) [20]. *DMC* – первая прогнозирующая система, используемая в промышленности. Идея заключалась в обработке многопараметрических управляющих линейных систем без каких либо ограничений для предсказания будущих значений. Алгоритм предсказывал необходимое поведение предприятия для плавного сближения к целевым заданным значениям.

MPC основан на постепенной оптимизации системы на конечном временном горизонте. В момент времени t выбирается текущее состояние системы и вычисляется стратегия минимизирующая затраты для относительно короткого временного горизонта H.

В задаче *MPC* можно выделить:

- Динамическую модель системы
- Историю предыдущих управляющих сигналов
- Целевая стоимостная функция

С помощью динамической модели системы мы можем предсказывать будущие состояния системы в дискретном времени.

$$x_{k+1} = A x_k + B u_k$$

Зная текущее состояние, возможно спрогнозировать контрольные сигналы на некотором временном промежутке H:

$$\begin{aligned} x_{k+1} &= A x_k + B u_k \\ x_{k+2} &= A x_{k+1} + B u_{k+1} \\ &\vdots \\ x_{k+H} &= A x_{k+H-1} + B u_{k+H-1} \end{aligned}$$

При рекурсивной подстановке одного уравнения в другое можно получить предсказание из начального состояния в состояние в конце временного горизонта.

$$
\begin{aligned}
x_{k+1} &= A\,x_k + B\,u_k \\
x_{k+2} &= A^2\,x_k + A\,B\,u_k + B\,u_{k+1} \\
&\vdots \\
x_{k+H} &= A^H\,x_k + A^{H-1}\,B\,u_k + A^{H-2}\,B u_{k+1} + \cdots + A\,B u_{k+H-2} + B\,u_{k+H-1}
\end{aligned}
$$

То есть предсказание в момент k можно представить в виде:

$$
\begin{bmatrix} x_{k+1|k} \\ x_{k+2|k} \\ \vdots \\ x_{k+H|k} \end{bmatrix} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^H \end{bmatrix} x_k + \begin{bmatrix} B & 0 & \cdots & 0 \\ A\,B & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{H-1}B & A^{H-2}B & \cdots & B \end{bmatrix} \begin{bmatrix} u_{k|k} \\ u_{k+1|k} \\ \vdots \\ u_{k+H-1|k} \end{bmatrix}
$$

Отсюда видно, что управляющие переменные $u_{k|k}, u_{k+1|k}, \ldots, u_{k+H-1|k}$ могут отвечать за команды, управляющие траекторией состояния, что как раз необходимо для для отслеживания пути.

Однако такая система предназначена для линейных систем. В любом случае, подобную процедуру можно провести и для нелинейных систем.

Такая задача представлена в виде задачи квадратичного программирования, которая формулируется следующим образом:

$$
\begin{aligned}
&\text{minimize} && \sum_{k=1}^{H} x^T Q\,x + \Delta u^T R\,\Delta u \\
&\text{subject} && x_{k+1} = A\,x_k + B\,u_k, k = 1, \ldots, H-1 \\
& && x \in \mathbb{R}^n, u \in \mathbb{R}^m \\
& && x_k \in X \\
& && u_k \in U
\end{aligned}
$$

Где целевая функция минимизируется по временному горизонту H с ограничениями возможных состояний X и ограничениями управляющих сигналов U. *MPC* работает поэтапно, и с помощью модели он предсказывает поведение робота на несколько шагов в будущем. Результатом оптимизации является последовательность управляющих сигналов и предсказаний.

2.5 Итог

Из рассмотренных рассмотренных реактивных алгоритмов можно выделить два подкласса: (1) алгоритмы на основе конфигурационного пространства,

(2) алгоритмы на основе многокритериальной оптимизации. Данные подходы лучше всего подходят для неголономных систем, в частности – колесных мобильных роботов. Отличия алгоритмов приведены в таблице 2.

Таблица 2. Рассмотренные алгоритмы локального планирования.

<table>
<tr><td colspan="4">Реактивные алгоритмы</td></tr>
<tr><td colspan="2">На основе конфигурационного пространства</td><td colspan="2">На основе многокритериальной оптимизиации</td></tr>
<tr><td colspan="2">Происходит выборка конфигурации (υ, ω) из рассматриваемого пространства</td><td colspan="2">Выбор оптимальной последовательности состояний с помощью набора ограничений, заданных целевыми функциями</td></tr>
<tr><td>DWA</td><td>TR</td><td>TEB</td><td>MPC</td></tr>
<tr><td>Выбирает конфигурацию исходя из того, что на всем промежутке времени конфигурация не изменится</td><td>Выбирает конфигурацию исходя из всего определенного временного горизонта</td><td>Применяется только к автомобилеподобным роботоам, поэтому целевые функции строго фиксированы, ограничения аппроксимированы полиномиальной функцией</td><td>Может применяться к любым колесным роботам, целевые функции модифицируемы, ограничения не аапроксимируются и могут принимать любой вид</td></tr>
</table>

3 СРЕДА ДЛЯ ИСПОЛНЕНИЯ АЛГОРИТМОВ

Большинство современных мобильных колесных роботов используют такой механизм управления, как дифференциальный привод. По этой причине, в ходе данной работы указанные алгоритмы были использованы на роботе с таким механизмом управления.

Робот с дифференциальным приводом состоит из двух ведущих колес, установленных на общей оси. Каждое колесо может управляться в обе стороны, т. е. двигаться вперед или назад.

Для управления роботом используется Robot Operating System (ROS). ROS – это гибкая платформа с обширным набором инструментов, которая используется в большом количестве роботизированных систем, будь то домашние роботы, коммерческие роботы на производстве, или роботы используемые в исследовательских целях. В связи с распространенностью данной системы, она легла в основу ПО для управления роботом в данной работе.

В качестве используемого робота выбран симулятор Gazebo. Это физически-корректная трехмерная среда, которая полностью заменяет реального робота. Gazebo – распространенный симулятор в сообществе робототехники, направленный на точную симуляцию реального мира, интегрированный в ROS.

3.1 Робот с дифференциальным приводом

Данный тип роботов принадлежит типу неголономных систем. Неголономные системы характеризуются уравнениями связывающими производные переменных конфигурации системы по времени. Как правило такое происходит, когда в системе меньше элементов управления, чем переменных конфигурации.

Робот с дифференциальным приводом [21] имеет два органа управления, которыми являются акселераторы левого и правого колеса, однако движется

робот в трехмерном пространстве (x, y, θ). Таким образом, не каждый путь в доступном пространстве является достижимым.

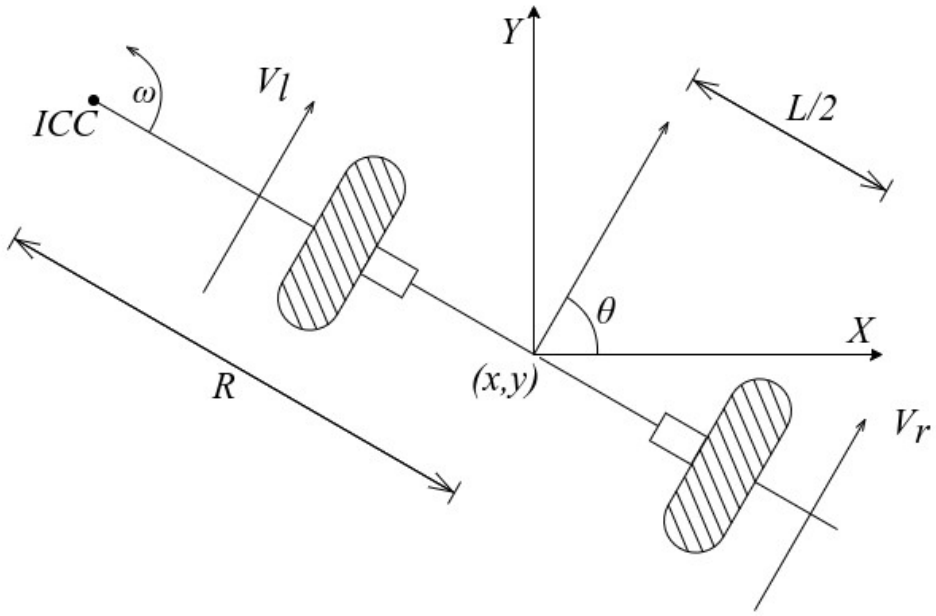

Рисунок 7. Кинематика робота с дифференциальным приводом.

Изменение скорости вращения колес приводит к повороту робота. Вращение происходит вокруг точки лежащей на общей с колесами оси. Точка вокруг которой вращается робот называется мгновенным центром скоростей (*ICC*, от англ. Instantaneous Center of Curvature), показанном на рисунке 7.

Изменение скорости вращения колес, приводит к изменению траектории движения робота. Поскольку скорость поворота робота ω вокруг *ICC* должна быть одинакова для обоих колес, верны следующие выражения:

$$\omega(R+l/2)=V_r$$

$$\omega(R-l/2)=V_l$$

Где l – расстояние между колесами, V_r, V_l – скорости движения колес вдоль земли. R – радиус поворота. Значения ω и R в любой момент времени могут быть получены следующим образом:

$$R=\frac{l}{2}\frac{V_l+V_r}{V_r-Vl}; \quad \omega=\frac{V_r-V_l}{l}$$

Рассмотрим три особых случая:

1. Если $V_l = V_r$, то робот движется вдоль прямой. R равно бесконечности, а ω равно нулю
2. Если $V_l = -V_r$, тогда R равно нулю, робот вращается вокруг своего центра.
3. Если $V_l = 0$, Робот вращается вокруг левого колеса, R равно $l/2$. Это также верно и для $V_r = 0$, только в таком случае вращение происходит вокруг правого колеса.

Стоит обратить внимание, что в реальных условиях робот не может двигаться вдоль прямой, поскольку всегда присутствует разность между скоростями колес. Робот с дифференциальным приводом крайне чувствителен к изменениям скорости колес, даже очень маленькое изменение скоростей влечет за собой изменение траектории. Также робот очень чувствителен к колебаниям поверхности и нуждается в дополнительных колесах для поддержки.

Манипулируя значениями V_l и V_r, можно изменять положение (x, y, θ) робота. Зная скорости V_l и V_r, можно рассчитать положение точки ICC.

$$ICC = (x - R\sin(\theta), y - R\cos(\theta))$$

Тогда, значение производных переменных конфигурации по времени $(\dot{x}, \dot{y}, \dot{\theta})$ равно:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\omega) & -\sin(\omega) & 0 \\ \sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega \end{bmatrix} = \begin{bmatrix} \upsilon \cdot \cos(\theta) \\ \upsilon \cdot \sin(\theta) \\ \omega \end{bmatrix}$$

Где $\upsilon = (V_l + V_r)/2$.

3.2 Robot Operating System

ROS разработана компанией *Willow Garage* в сотрудничестве со Стэнфордским университетом в рамках проекта *STAIR*, является промежуточным ПО для крупномасштабной разработки сложных роботизированных систем. Это мета-операционная система с открытым исходным кодом для роботов. ROS, как и

операционная система, предоставляет набор сервисов, связанных с аппаратной абстракцией, низкоуровневым управлением устройствами, передачей сообщений между процессами, управлением пакетов. В настоящее время ROS широко используется в сообществе робототехники. Его основная цель – упростить разработку многокомпонентных роботизированных систем, которые могут работать с различными роботах при небольших изменениях.

ROS основывается на трех уровнях концепций: (1) уровень файловой системы, (2) уровень вычислительного графа, (3) уровень сообщества. В рамках данной работы необходимо усвоить концепцию вычислительного графа (рис. 8), который представляет собой P2P сеть процессов ROS.

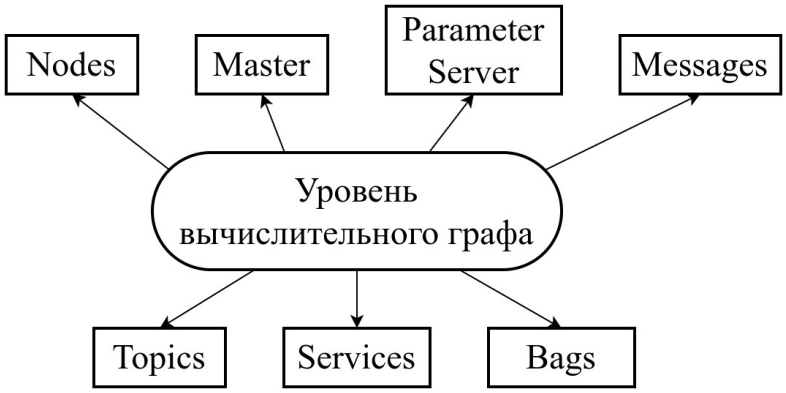

Рисунок 8. Уровень вычислительного графа ROS.

Базовые элементы из которых состоит графовая часть ROS:

- *Узлы (Nodes)*: Это процессы, выполняющие вычисления. ROS – модульная система состоящая из множества узлов, которые разделяют код и функциональные возможности, делая систему проще и более гибкой. Они реализуются с помощью библиотек ROS, таких как roscpp или rospy.
- *Главный узел (Master)*: Обеспечивает регистрацию имен и поиск других сервисов вычислительного графа. Роль главного узла состоит в том, чтобы позволить другим узлам находить друг друга, после чего узлы взаимодействуют в режиме P2P.

- *Сервер параметров (Parameter Server)*: Обеспечивает централизованное хранение данных. С помощью сервера параметров можно настраивать узлы во время их работы.
- *Сообщения (Messages)*: Узлы общаются друг с другом, передавая сообщения. Сообщение содержит данные, которые предоставляют информацию другим узлам. Это структура данных, состоящая из типизированных полей.
- *Топики (Topics)*: Сообщения направляются с помощью системы, работающей по системе издатель/подписчик. Узлы отправляют сообщения, публикуя их в определенный топик. Топик – это идентификатор для определенного типа сообщений. Узлы которым необходимы определенный данные, подпишутся на соответствующий топик. Как только топик получает сообщение, он вызывает обратные функции, которые оповещают соответствующие узлы.
- *Сервис (Service)*: Топики работают по принципу многие-ко-многим, что означает, что у топика может быть как несколько издателей, так и несколько подписчиков. Такая модель не подходит для взаимодействий типа запрос/ответ. Подобные взаимодействия осуществляются с помощью сервисов. Узел предоставляет сервис под определенным идентификатором, другие узлы используют этот сервис для отправления запросов и получения ответов, то есть по принципу один-ко-многим.
- *Записывающие хранилища (Bags)*: Основной механизм журналирования данных в ROS. Он позволяет записывать, визуализировать, маркировать и сохранять данные для дальнейшего использования. Данные такого хранилища можно воспроизводить и использовать так, будто их отправляет узел ROS.

На рисунке 9 показан процесс передачи данных между узлами. Узел издатель публикует сообщение в топик. Другой узел должен подписаться на соответствующий узел, чтобы получить сообщение.

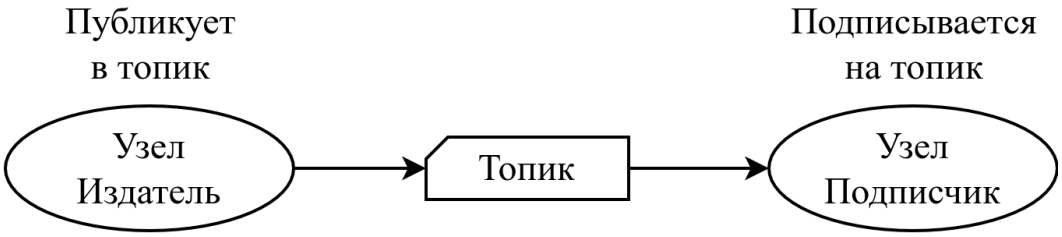

Рисунок 9. Передача данных между узлами ROS.

В ROS также предусмотрены таймеры. Они позволяют запланировать обратный вызов с определенной частотой. Чтобы использовать таймер, его необходимо запустить (например в обратной функции топика). Как только таймер запущен, обратная функция будет исполнена.

Еще один важный аспект ROS заключается в том, что он поставляется вместе с 3D-визуализатором Rviz (ROS Visualization) [22]. Идея Rviz заключается в предоставлении графического отображения различных сообщений ROS. Rviz довольно легок в использовании, пользователю необходимо выбрать отображение либо с помощью типа сообщения, либо по доступному топику. Что касается задачи навигации, то полезными для отображения являются сообщения типа «*Map*», «*Path*», «*Pose*», «*Odometry*», и т. д. Каждое отображение можно настроить выбрав цвет, топик или другие, специфичные для данного сообщения свойства. Rviz также позволяет сохранить конфигурацию отображаемых данных для конкретной роботизированной системы, так что нет нужды настраивать его каждый раз. На рисунке 10 показан пример настроенного Rviz

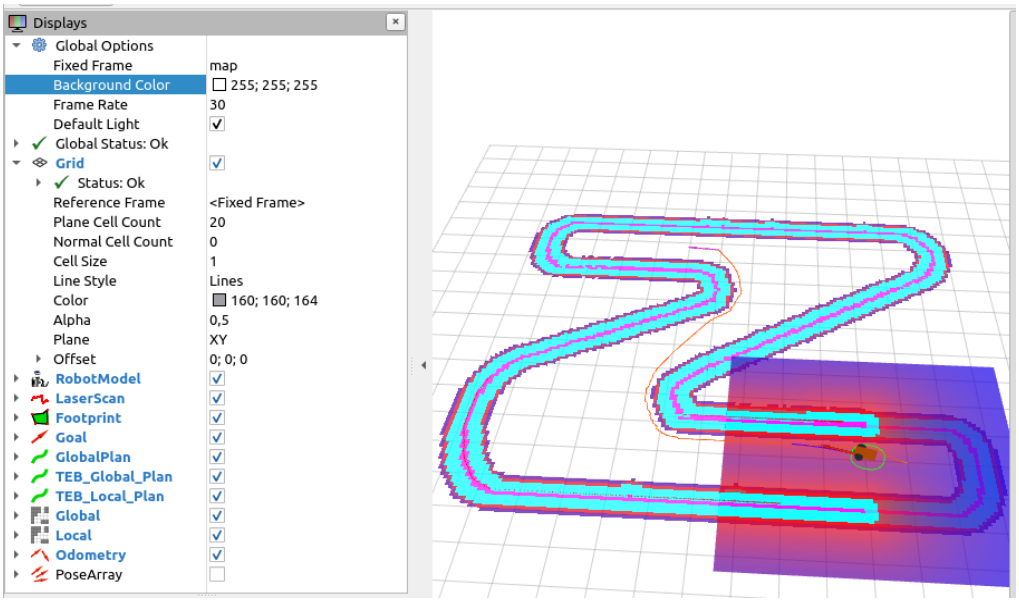

Рисунок 10. Визуализатор Rviz.

3.3 Gazebo

Симулятор Gazebo спроектирован так, чтобы точно воспроизводить динамическую среду, с которой может столкнуться робот [23]. Все смоделированные объекты обладают массой, скоростью, трением и множеством других характеристик, которые позволяют этим объектам вести себя реалистично. Пример окна Gazebo приведен на рисунке 11.

Gazebo – не единственный симулятор 3D-динамики, однако один из немногих, способный создавать реалистичные миры для роботов. По мере разработки Gazebo, качество симуляции совершенствуется и грань с реальностью становится все меньше.

Симуляция окружения – это, по сути, набор моделей и датчиков. Некоторые объекты, такие как поверхность передвижения являются стационарными моделями и не могут быть передвинуты во время симуляции, тогда как роботы и другие объекты – динамическими. Датчики – отделены от моделей и только собирают или выдают данные.

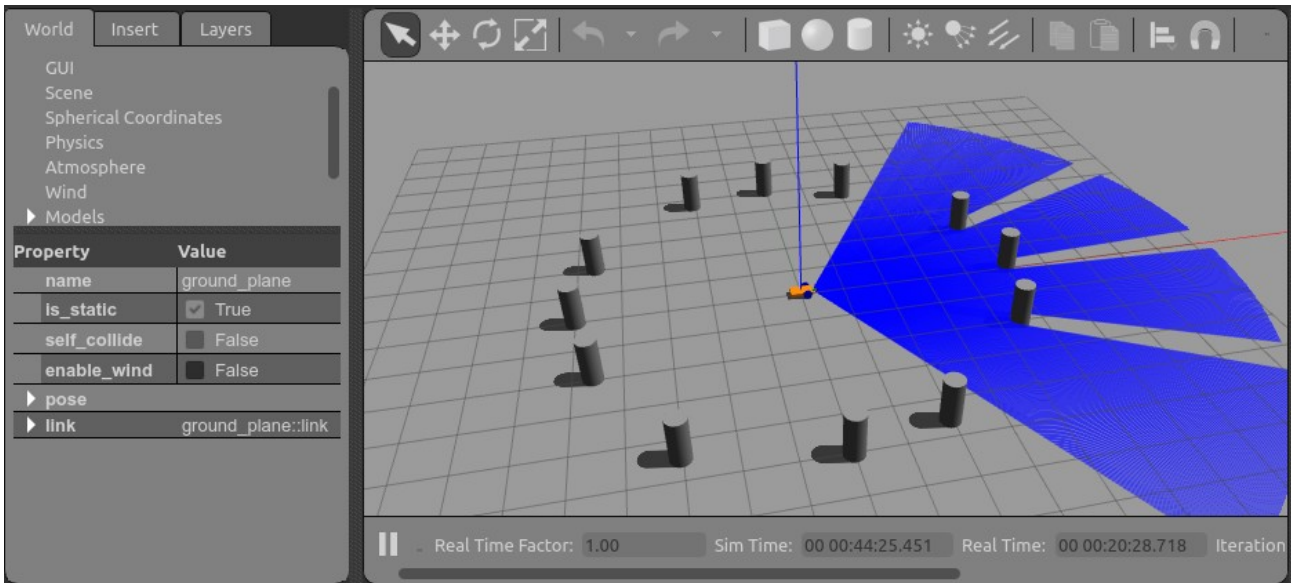

Рисунок 11. Симулятор Gazebo.

Основные компоненты симулятора:

- *Модели, тела, соединения*: Модель (Model) – любой объект, который имеет физическое представление. Модели включают в себя все от простой геометрии до сложных роботов. Модель состоит как минимум из одного твердого тела, нуля или более соединений или датчиков.

  Тела (Body) представляют собой «строительные блоки» для создания моделей. Они могут принимать форму таких примитивов как параллелепипед, шар, цилиндр, плоскость или линия, кроме того, тела могут быть созданы как полигональная модель. Каждое тело имеет массу, трение, коэффициент отскока и различные свойства отрисовки.

  Соединения (Joint) обеспечивают механизм соединения тел для создания кинематических и динамических отношений. Доступны различные соединения: шарнирные, для вращения вокруг одной или двух осей, скользящие и универсальные. Соединения также могут выступать в роли двигателей.

- Сенсоры: Роботы не могут самостоятельно совершать какие либо действия, если они не имеют сенсоров. Сенсор в Gazebo – абстрактное устройство без физического представления, включающееся в модель.
  В Gazebo доступно три типа сенсоров: одометр, лучевой датчик, камера.
- Внешние интерфейсы: Gazebo активно используется в ROS. Сервер ROS с помощью специального плагина «*ros-gazebo*» рассматривает Gazebo как реальное устройство, способное отправлять или получать данные.
  Библиотека Gazebo предоставляет интерфейс для установки скорости вращения колес, считывания данных с лазерных датчиков, получения изображения с камеры.

3.4 Навигация (Navigation Stack)

Для навигации мобильного робота решить три основные задачи: построение карты, локализация и планирование пути. ROS имеет набор полезных инструментов для навигации робота по известной, частично известной или неизвестной среде; используя их, робот может планировать и отслеживать путь, пока он отклоняется от препятствий, которые появляются на его пути на протяжении всего курса. Эти инструменты включает в себя стек навигации ROS [24].

Устройство стека навигации показано на рисунке 12. Основным узлом является *«move_base»*. Этот узел подписывается на одометрию, данные с датчиков, и положение точки назначения. Обрабатывая полученные данные, *«move_base»* предоставляет сигналы управления роботом (команды скорости вращения колес). Также move_base может подписаться на на карту – статичное изображение местности, позволяющее генерировать глобальные маршруты. Источником одометрии является симулятор Gazebo, аналогом в реальных условиях может служить Wi-Fi позиционирование или отслеживание с помощью камер. Однако зачастую используется алгоритм адаптивной локализации

Монте-Карло (AMCL), поскольку это не требует дополнительных устройств, а только какое-то знание окружающей местности.

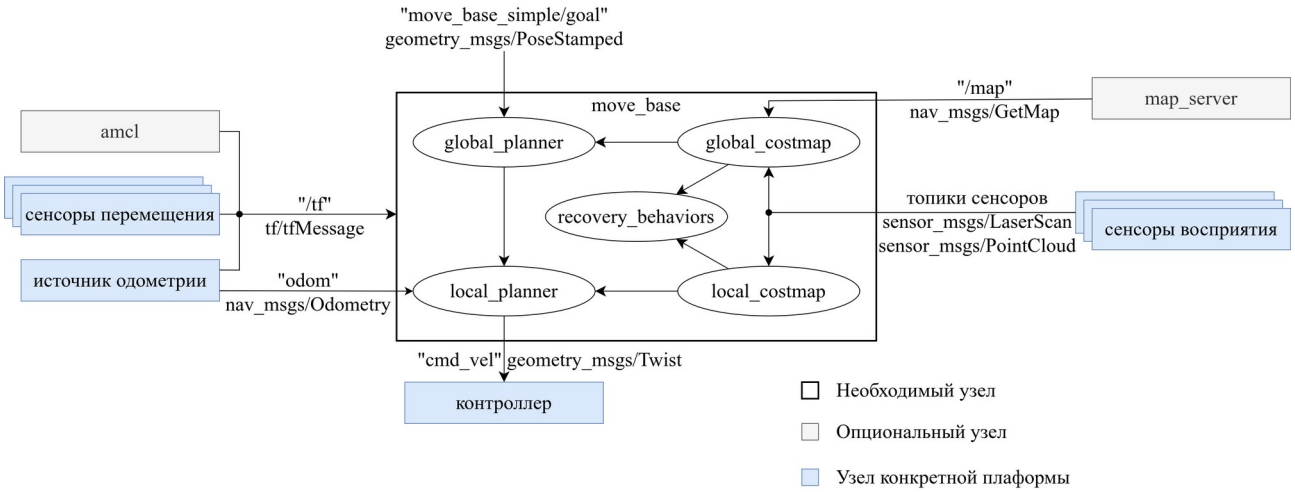

Рисунок 12. Схема узлов в ROS Navigation Stack.

«move_base» состоит из двух модулей: глобальное и локальное планирование. Работа этих модулей обеспечивается глобальной и локальной картой для соответствующих планировщиков. Карты содержат в себе информацию о препятствиях в окружающей среде в виде сетки. Глобальная карта инициализируется статической картой, если такая доступна, после чего обновляется с помощью локальной карты. Локальная карта в свою очередь инициализируется глобальной картой и постоянно обновляется с помощью датчиков в небольшой области около робота.

Для того, чтобы использовать какой-либо глобальный или локальный планировщик в «move_base», необходимо, чтобы он соответствовал определенному интерфейсу, который определен непосредственно в «move_base». Глобальный и локальный планировщики должен соответствовать интерфейсам «nav_core::BaseGlobalPlanner» и «nav_core::BaseLocalPlanner» соответственно. Также они должны быть добавлены в ROS как плагины, чтобы они могли работать вместе со стеком навигации.

4 СРАВНЕНИЕ АЛГОРИТМОВ

Эксперимента проводились в симуляционной среде. Для этого была создана элементарная модель робота с дифференцированным двигателем с установленным на него лазерным сканером с ограниченной видимостью.

Для созданного робота созданы три различные среды:

- Коридор с резкими поворотами и полностью известная карта (заранее известная среда)
- Коридор с поворотом на прямой угол, в котором расставлены различные препятствия, о которых не известно робота (частично известная среда)
- Открытая среда, к которой не предоставлено никакой карты и на которой расставлено множество препятствия (полностью неизвестная среда)

Все алгоритмы запускались с одинаковыми конфигурациями. Динамические ограничения и генераторы стоимостных карт одинаковы.

Конфигурация:

- Максимальная линейная скорость: 2 м/с
- Максимальная скорость поворота: 1 рад/с
- Максимальное линейное ускорение: 0.5 м/с^2
- Максимальное ускорение поворота: 0.5 рад/с^2
- Размер локальной карты 7×7 м
- Дальность видимости сканера 4 м
- Расстояние расчета локальной карты от препятствий: 1 м
- Максимальная скорость заднего хода (TR, TEB, MPC): 0.5 м/с
- Динамические препятствия (TEB, MPC): включены
- Радиус габаритов робота: 0.35 м
- Допустимое отклонение: 0.20 м, 0.10 рад.

4.1 Модель робота

Основной корпус робота представляет собой параллелепипед длиной, шириной и высотой 50 см, 30 см и 7 см соответственно. Робот направлен вдоль длины этого параллелепипеда.

Колеса робота расположены по бокам корпуса. Центр колес закреплен следующим образом: вдоль OZ по середине корпуса, вдоль OX на расстоянии 0.25 м от центра корпуса. Диаметр колес 20 см, ширина колес 4 см.

В передней части робота, по центру грани корпуса располагается лазерный сканер. Габариты лазера 5×5×5 см. Угол обзора сканера 120°. Сканер выпускает 720 лучей. Частота обновления сканера: 40 Гц.

В задней части робота находится опорный ролик.

Схема модели робота представлена на рисунке 13.

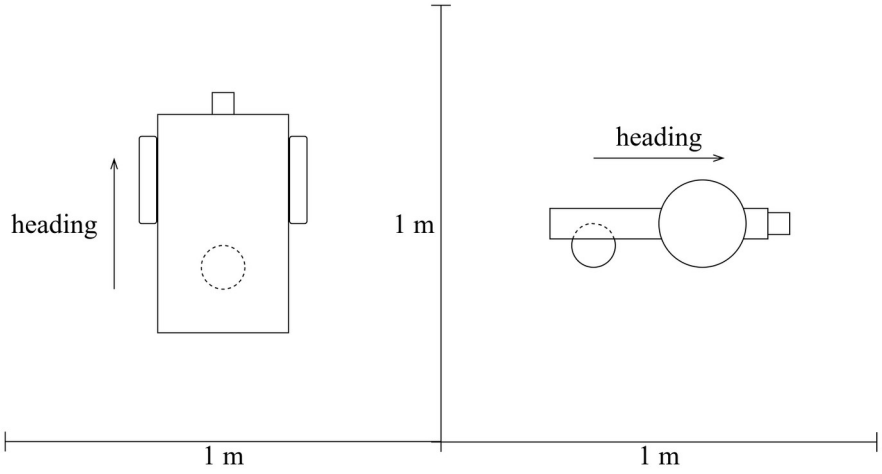

Рисунок 13. Схема модели робота.

По данной схеме создана модель, описанная в формате URDF. Корпус имеет массу 2 кг. Колеса массой 200 грамм, задний ролик – 50 грамм.

Для управления колесами использовался плагин «*libgazebo_ros_diff_drive*». Данный плагин необходим для того, чтобы робот мог публиковать данные об одометрии в топик «*/odom*» и подписываться на топик «*/robot/cmd_vel*». Для ра-

боты лазерного сенсора использовался плагин «*libgazebo_ros_laser*». Данный плагин публикует данные, полученные сканером в топик «*/robot/laser/scan*».

Полученная модель представлена на рисунке 14.

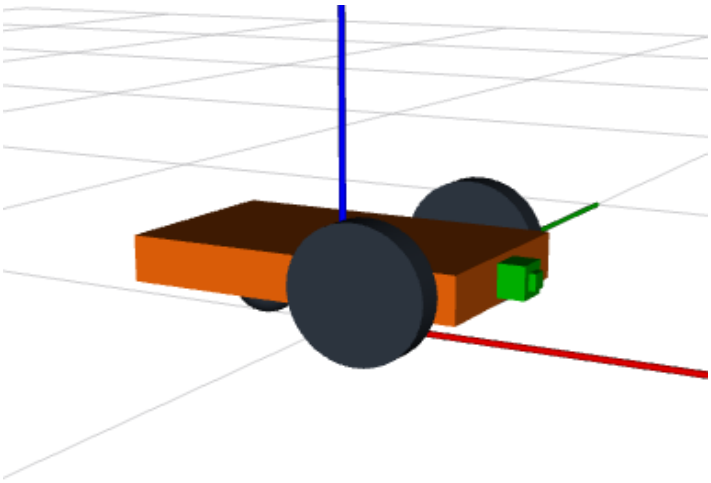

Рисунок 14. Модель робота из URDF.

4.2 Метрики оценки

Скорость генерации маршрута будет оценена с помощью двух параметров:

- Частота публикации сообщений в топик cmd_vel
- Время движения робота

Данные метрики напрямую берутся из данных *rosbag*. Частота публикации рассчитывается из количества сообщений в «*cmd_vel*» и промежутка времени, за которое было опубликованы эти сообщения. Время движения робота рассчитывается из времени публикации первого и последнего сообщения в «*cmd_vel*». Последняя команда, равная (0, 0) означает, что робот достиг пункта назначения.

Качество построенного маршрута оценивается с помощью расчетных характеристик:

- Гладкость построенного маршрута
- Гладкость (колебание) скоростей, во время следования пути

- Отклонение от маршрута

Гладкость (колебания) построенного маршрута: СКО нормализованных линейных скоростей показывает насколько сильно колеблется маршрут.

$$\sqrt{D\left[\frac{\|dx, dy\|}{dt}\right]}$$

```
1.  smooth (t, x, y):
2.      normalize(t)
3.      normalize(x)
4.      normalize(y)
5.      return std(√(d(x)^2+d(y)^2) / d(t))
```

Гладкость (колебание) скоростей: СКО отношения нормализованных команд скоростей к нормализованному времени (нормализованная производная) показывает насколько сильно колеблется маршрут.

$$\text{smoothness}_{linear}=\sqrt{D\left[\frac{\text{cmd\_vel}_{linear}}{time}\right]} \qquad \text{smoothness}_{angular}=\sqrt{D\left[\frac{\text{cmd\_vel}_{angular}}{time}\right]}$$

```
1.  function(t, v):
2.      normalize(t)
3.      normalize(v)
4.      return std(d(v) / d(t))
```

Отклонение от маршрута: для каждой точки локальной траектории выбирается минимальная дистанция до глобального маршрута в каждый момент времени.

$$|\text{global\_route}(t)-\text{local\_route}(t)|$$

Метрикой отклонения является интеграл данной функции по времени:

$$\text{deviation}=\int|\text{global\_route}(t)-\text{local\_route}(t)|dt$$

Нормирование метрики происходит по длине локального пути:

$$\text{normal deviation}=\frac{\int|\text{global\_route}(t)-\text{local\_route}(t)|dt}{\int\sqrt{\text{local\_route}_x{}'^2(t)+\text{local\_route}_y{}'^2(t)}\,dt}$$

4.3 Заранее известная среда

В симуляторе Gazebo создана среда, представляющая собой коридор с резкими поворотами. Коридор представлен в виде высоких стен, которые не может преодолеть робот. Никаких дополнительных препятствий в коридоре нет. Трехмерное изображение сцены представлено на рисунке 15. Карта, которой располагает робот представлена на рисунке 16.

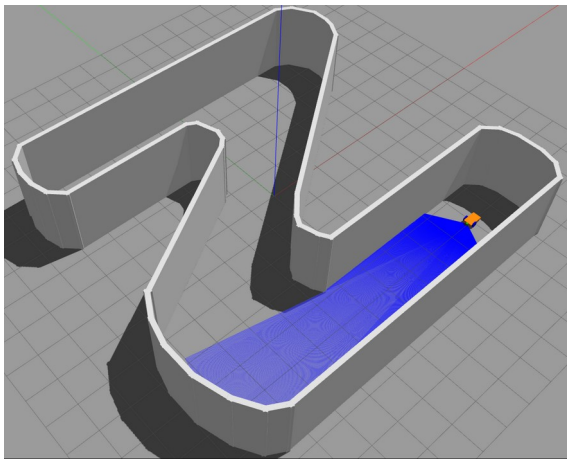

Рисунок 15. Полностью известная среда.

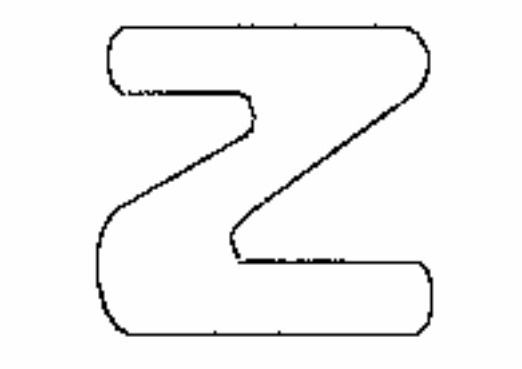

Рисунок 16. Карта местности полностью известной среды.

Данные о прохождении маршрута с помощью *rosbag* были записаны во время выполнения алгоритмов. Результаты работы алгоритмов (траектории движения) показаны на рисунке 17.

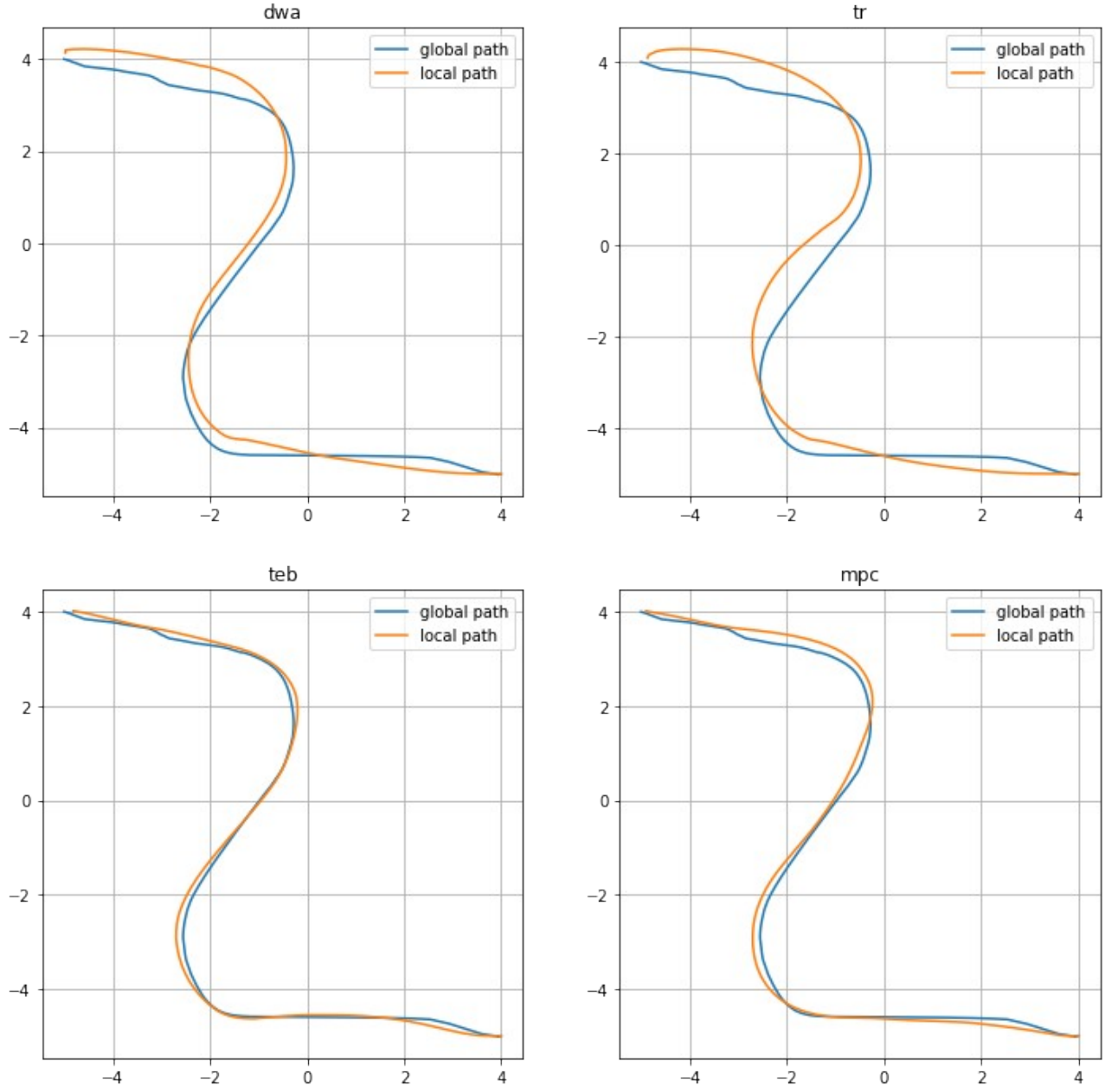

Рисунок 17. Маршруты в полностью известной среде.

Данные графики уже позволяют оценить гладкость маршрутов и их отклонение о глобального пути, однако для полной картины также получены графики зависимости команды скорости от времени (рис. 18)

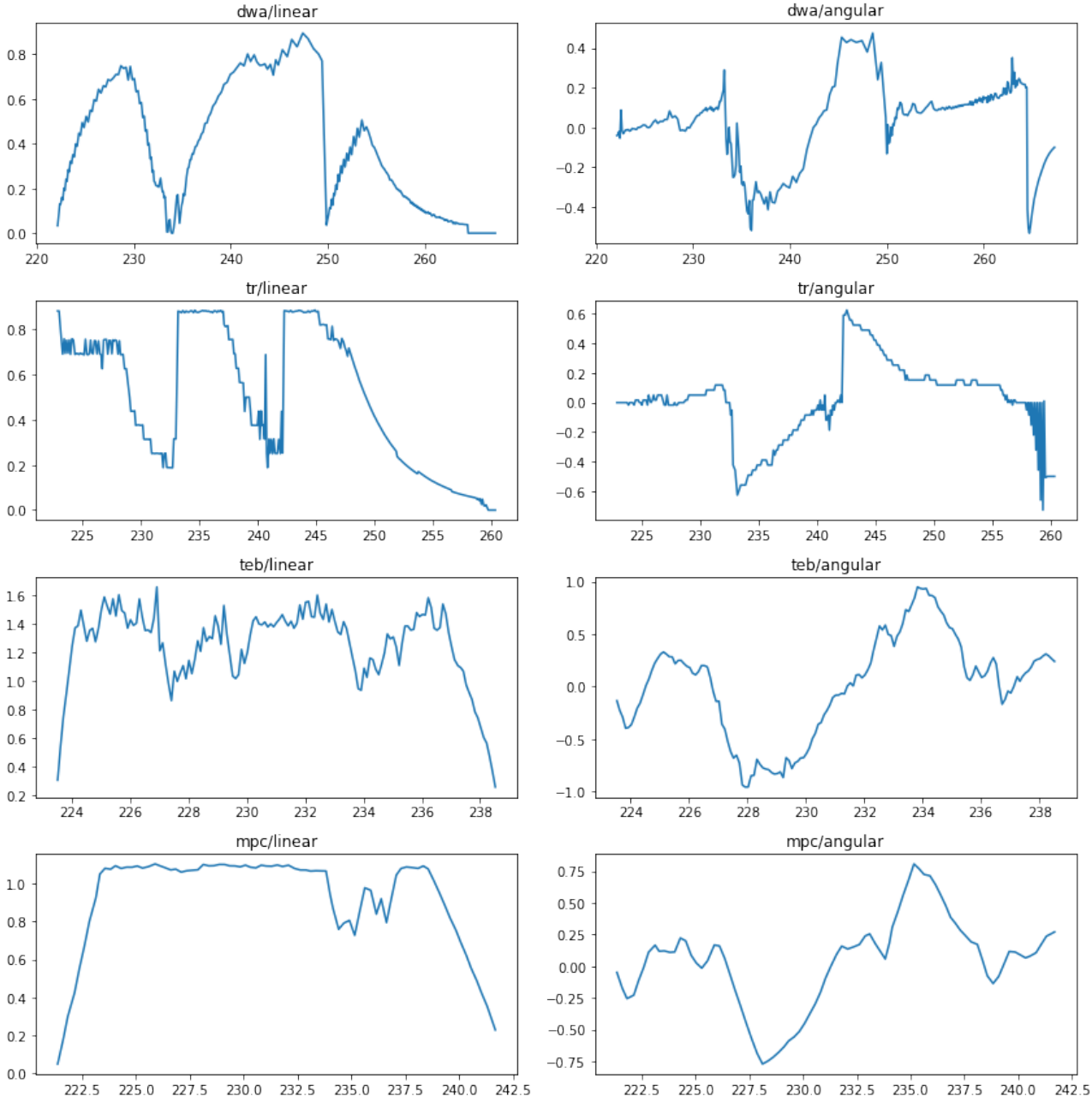

Рисунок 18. Колебания скоростей в полностью известной среде.

Данные графики позволяют увидеть как изменяются команды скорости во время движения вдоль изгибающейся кривой глобального маршрута. Для оценки отклонения построен график отклонения от глобального маршрута (рис. 19).

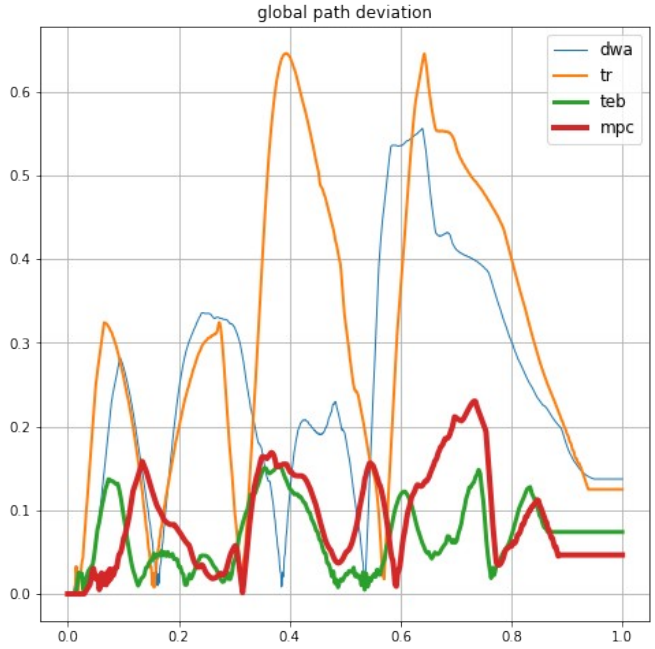

Рисунок 19. Отклонение траектории движения в полностью известной среде.

Результат оценивания гладкости (колебания) скоростей представлен в таблице 3. Поскольку скорость представлена двух типах: линейная и угловая, то внимание будет также взято общее значение колебаний, выраженное суммой.

Таблица 3. Колебания скоростей в полностью известной среде.

	Линейная	Угловая	Общая
DWA	11.195	30.090	41.285
TR	23.316	28.333	51.649
TEB	11.498	5.541	17.038
MPC	5.646	4.139	9.785

Все значения метрик оценивания алгоритмов показаны в таблице 4. За характеристику «колебания скоростей» взято общее колебание. Частота запуска TEB строго ограничена 10 Гц, поскольку все остальные методы не превышают данного значения.

Таблица 4. Оценки алгоритмов в полностью известной среде.

	Колеб. маршр.	Колеб. скоростей	Средн. скорость м/с	Макс. скорость м/с	Отклон. машр.	Длит. движ. с	Частота запуска Гц
DWA	1.5304	41.2853	0.2847	0.8943	0.6450	45.181	9.3458
TR	1.3794	51.6489	0.4811	0.8868	0.6753	37.468	9.6154
TEB	1.0102	17.0384	1.2526	1.6568	0.0652	14.994	10.0000
MPC	1.0270	9.7850	0.9225	1.1170	0.1171	20.582	8.7719

Для подсчета результирующей оценки метода выбора траектории все значения нормализуются. Гладкость маршрута и скорости, отклонение от глобального пути, а также длительность движения нормализуются по минимуму и берутся обратными. Значения скоростей и частота запуска нормализуются по максимуму. Нормализованные оценки приведены в таблице 5.

Таблица 5. Нормализованные оценки алгоритмов в полностью известной среде.

	Колеб. маршр.	Колеб. скоростей	Средн. скорость	Макс. скорость	Отклон. машр.	Длит. движ.	Частота запуска
DWA	0.6601	0.2370	0.2273	0.5398	0.1010	0.3319	0.9346
TR	0.7323	0.1895	0.3841	0.5353	0.0965	0.4002	0.9615
TEB	1.0000	0.5743	1.0000	1.0000	1.0000	1.0000	1.0000
MPC	0.9836	1.0000	0.7365	0.6742	0.5563	0.7285	0.8772

Просуммировав все значения из табл. 5 можно получить оценку алгоритма для полностью известной среды. Оценка показана в таблице 6.

Таблица 6. Оценка каждого алгоритма для полностью известной среды.

DWA	***TR***	***TEB***	***MPC***
3.0316	3.2993	6.5743	5.5563

4.4 Частично известная среда

В симуляторе Gazebo создана среда, представляющая собой обширный коридор, ограниченный стенами, с поворотом под прямым углом, заполненный различными препятствиями, о которых неизвестно роботу. Трехмерное изображение сцены представлено на рисунке 20. Карта местности, известная роботу, представлена на рисунке 21.

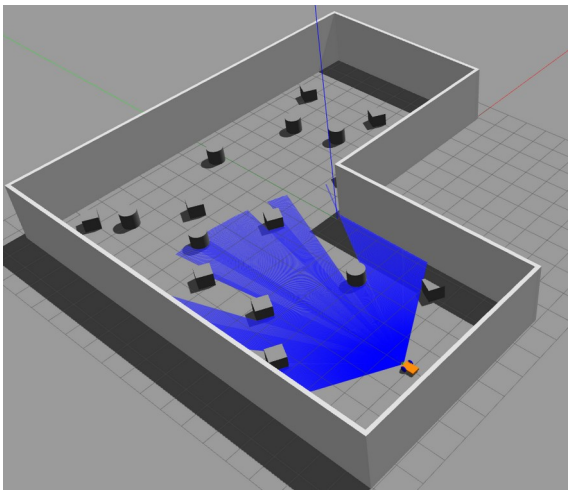

Рисунок 20. Частично известная среда.

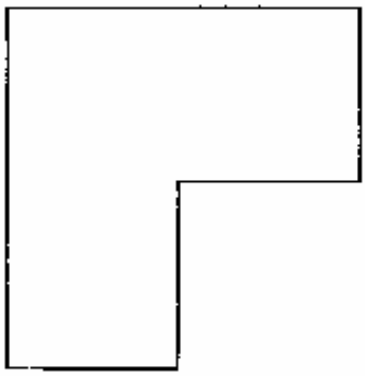

Рисунок 21. Карта местности частично известной среды.

Данные о прохождении маршрута с помощью *rosbag* были записаны во время выполнения алгоритмов. Результаты работы алгоритмов (траектории движения) показаны на рисунке 22.

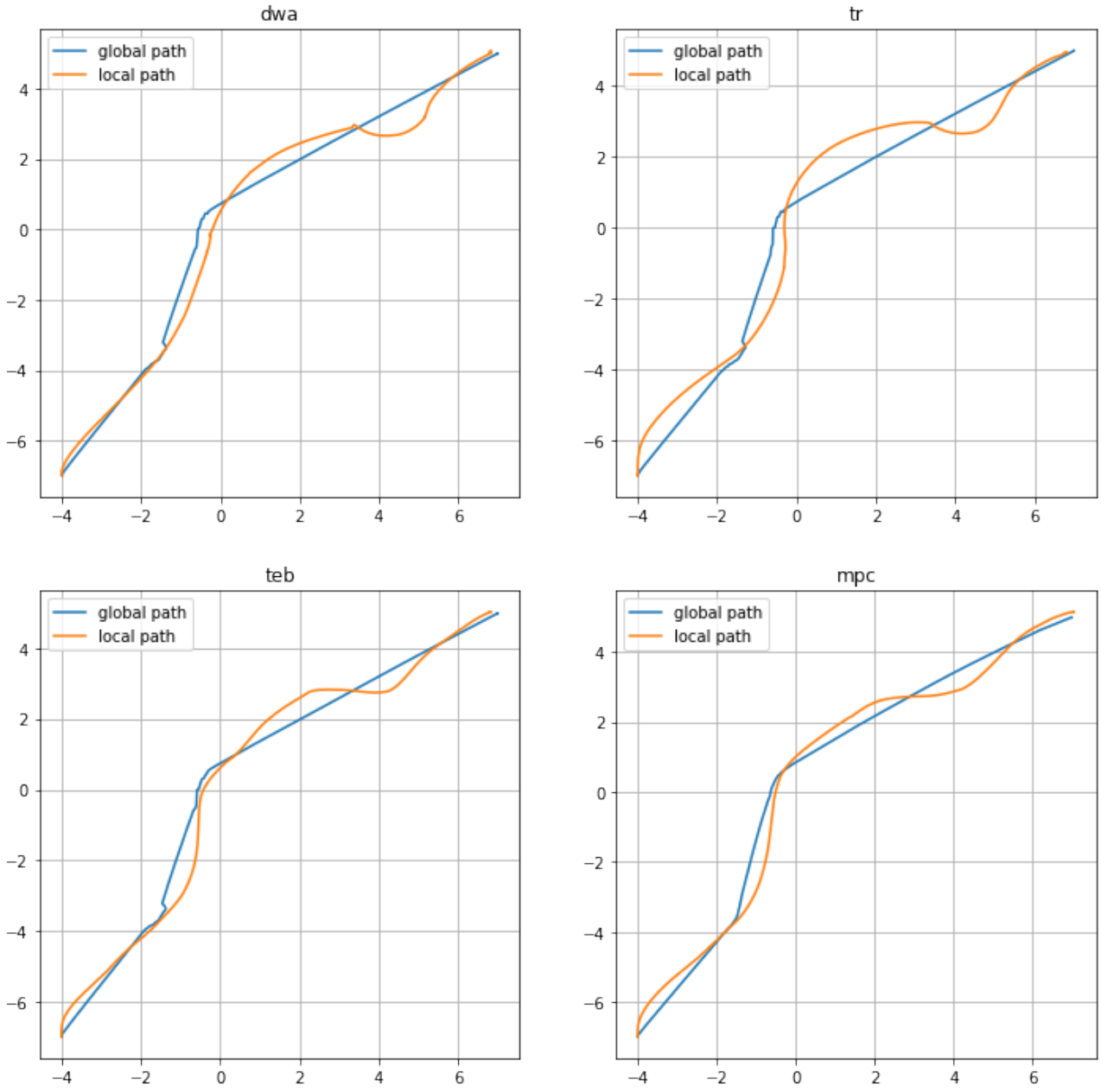

Рисунок 22. Маршруты в частично известной среде.

На данных графиках можно обратить внимание на сильные отклонения от глобального маршрута в одинаковых местах для каждого планировщика. Это обусловлено препятствиями, которые не учитывались при построении глобального маршрута. Графики зависимости команды скорости от времени представлены на рисунке 23.

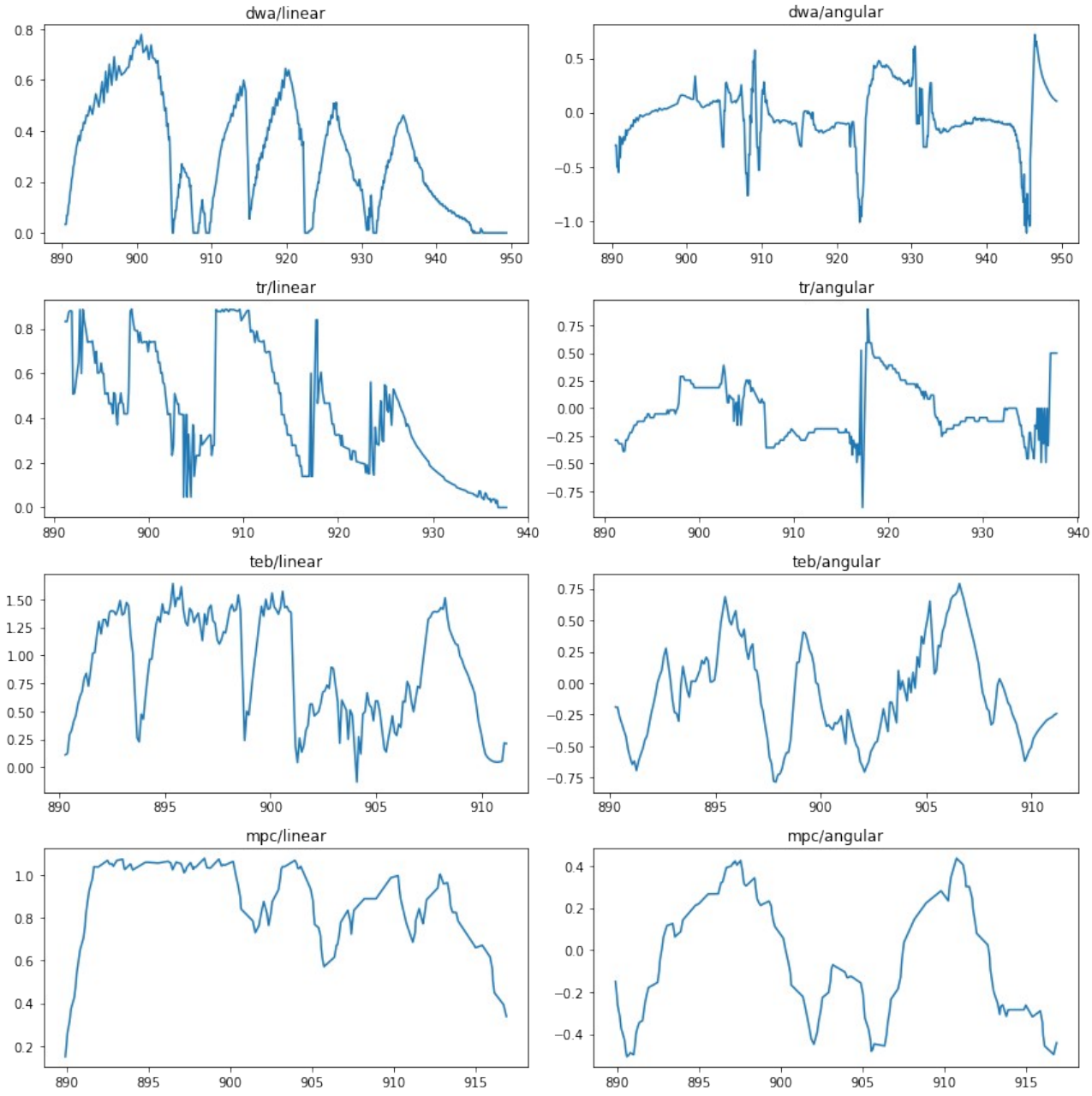

Рисунок 23. Колебания скоростей в частично известной среде.

По резким спадам скорости на графиках можно видеть, в какой момент происходило обнаружение неизвестного препятствия. Для оценки отклонения построен график отклонения от глобального маршрута (рис. 24).

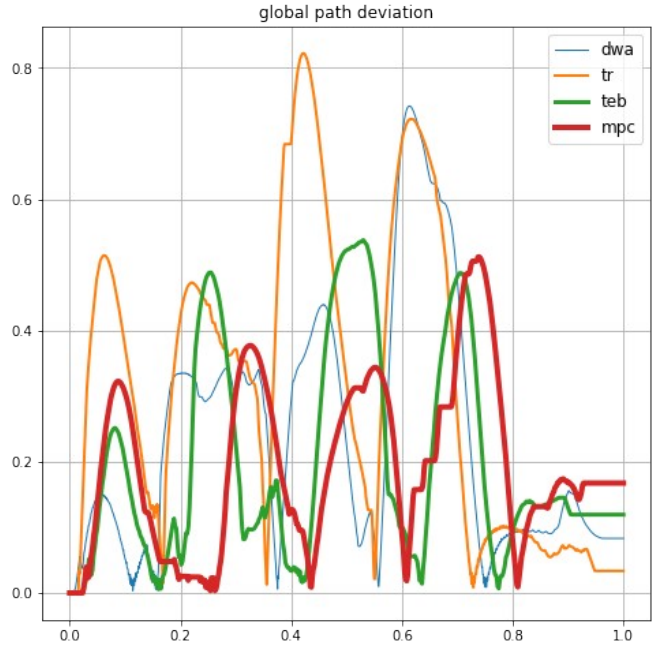

Рисунок 24. Отклонение траектории движения в частично известной среде.

Оценки колебания скоростей представлены в таблице 7.

Таблица 7. Колебания скоростей в частично известной среде.

	Линейная	Угловая	Общая
DWA	17.327	30.627	47.955
TR	76.043	40.786	116.829
TEB	16.464	13.241	29.705
MPC	13.516	14.476	27.992

Оценки по заданным критериям показаны в таблице 8.

Таблица 8. Оценки алгоритмов в частично известной среде.

	Колеб. маршр.	Колеб. скоростей	Средн. скорость м/с	Макс. скорость м/с	Отклон. машр.	Длит. движ. с	Частота запуска Гц
DWA	1.1917	60.7445	0.2402	0.7787	0.8192	58.832	9.5238
TR	1.2043	120.2568	0.4029	0.8873	0.8592	46.575	7.8740
TEB	1.1317	38.2347	0.8849	1.6392	0.3027	20.893	10.0000
MPC	1.0392	47.6574	0.8597	1.0779	0.3265	26.998	8.5470

Нормализованные оценки приведены в таблице 9.

Таблица 9. Нормализованные оценки алгоритмов в частично известной среде.

	Колеб. маршр.	Колеб. скоростей	Средн. скорость	Макс. скорость	Отклон. машр.	Длит. движ.	Частота запуска
DWA	0.8721	0.6294	0.2715	0.4750	0.3695	0.3551	0.9524
TR	0.8629	0.3179	0.4553	0.5413	0.3524	0.4486	0.7874
TEB	0.9183	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
MPC	1.0000	0.8023	0.9715	0.6576	0.9273	0.7739	0.8547

Суммарная оценка каждого алгоритма приведена в таблице 10.

Таблица 10. Оценка каждого алгоритма для частично известной среды.

DWA	***TR***	***TEB***	***MPC***
3.9251	3.7658	6.9183	5.9872

4.5 Полностью неизвестная среда

В симуляторе Gazebo создана среда, представляющая собой открытое, неограниченное пространство, заполненное различными препятствиями, о которых неизвестно роботу. Трехмерное представление сцены показано на рисунке 25. Роботу неизвестна карта заранее.

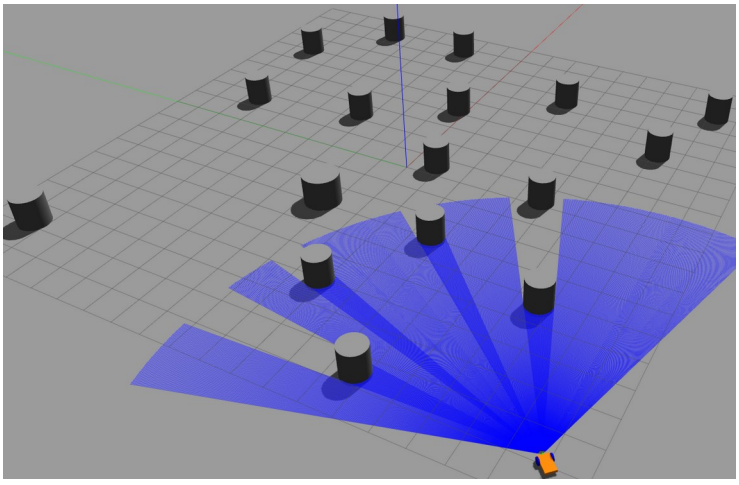

Рисунок 25. Полностью неизвестная среда.

Данные о прохождении маршрута с помощью *rosbag* были записаны во время выполнения алгоритмов. Результаты работы алгоритмов (траектории движения) показаны на рисунке 1.

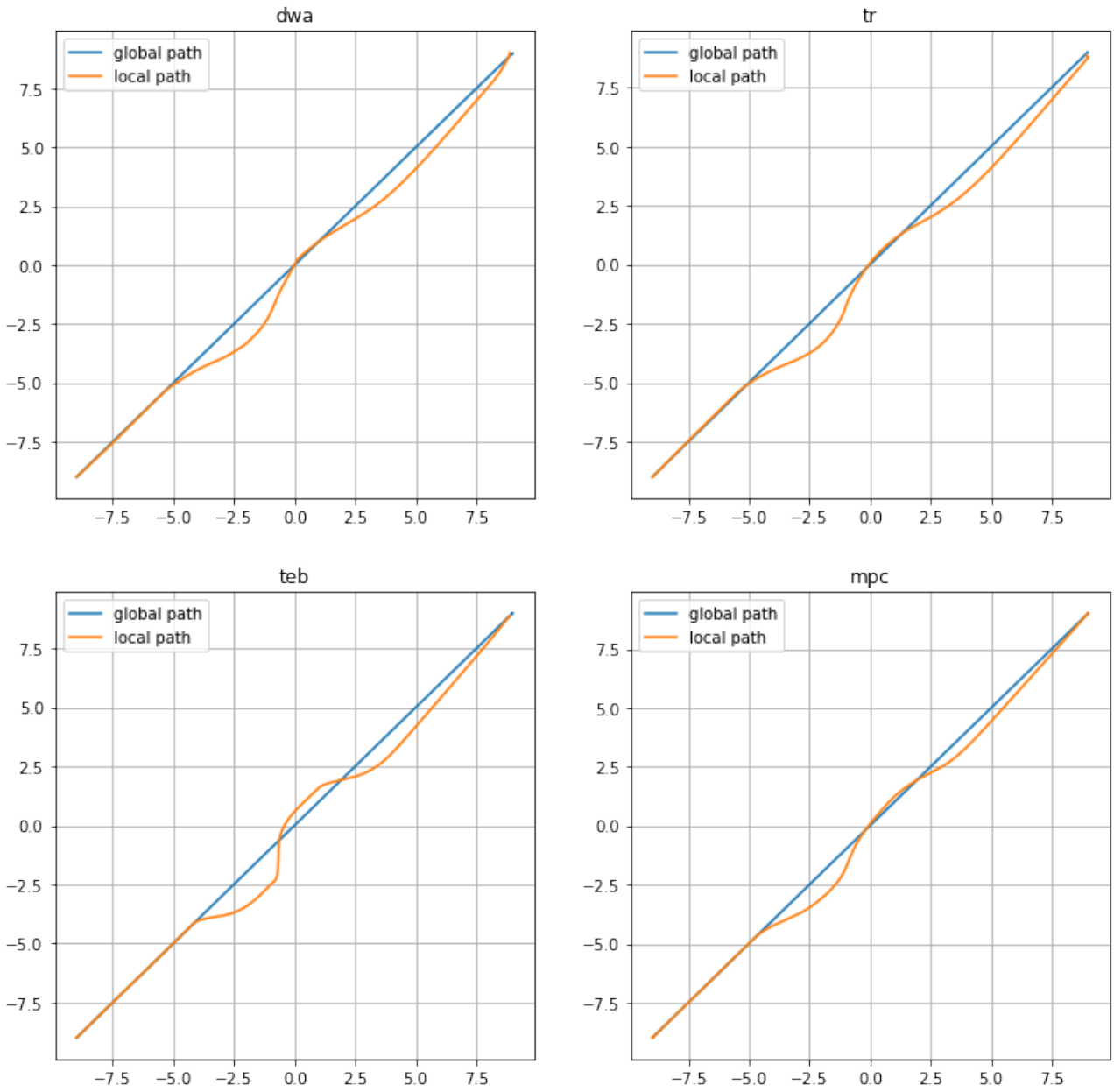

Рисунок 26. Маршруты в полностью неизвестной среде.

Любые отклонения от глобального маршрута обусловлены только препятствиями, встреченными на пути. Отклонение от маршрута показывает, насколько оптимально алгоритм обходит препятствия.

Графики зависимости команд скорости от времени отображены на рис. 27.

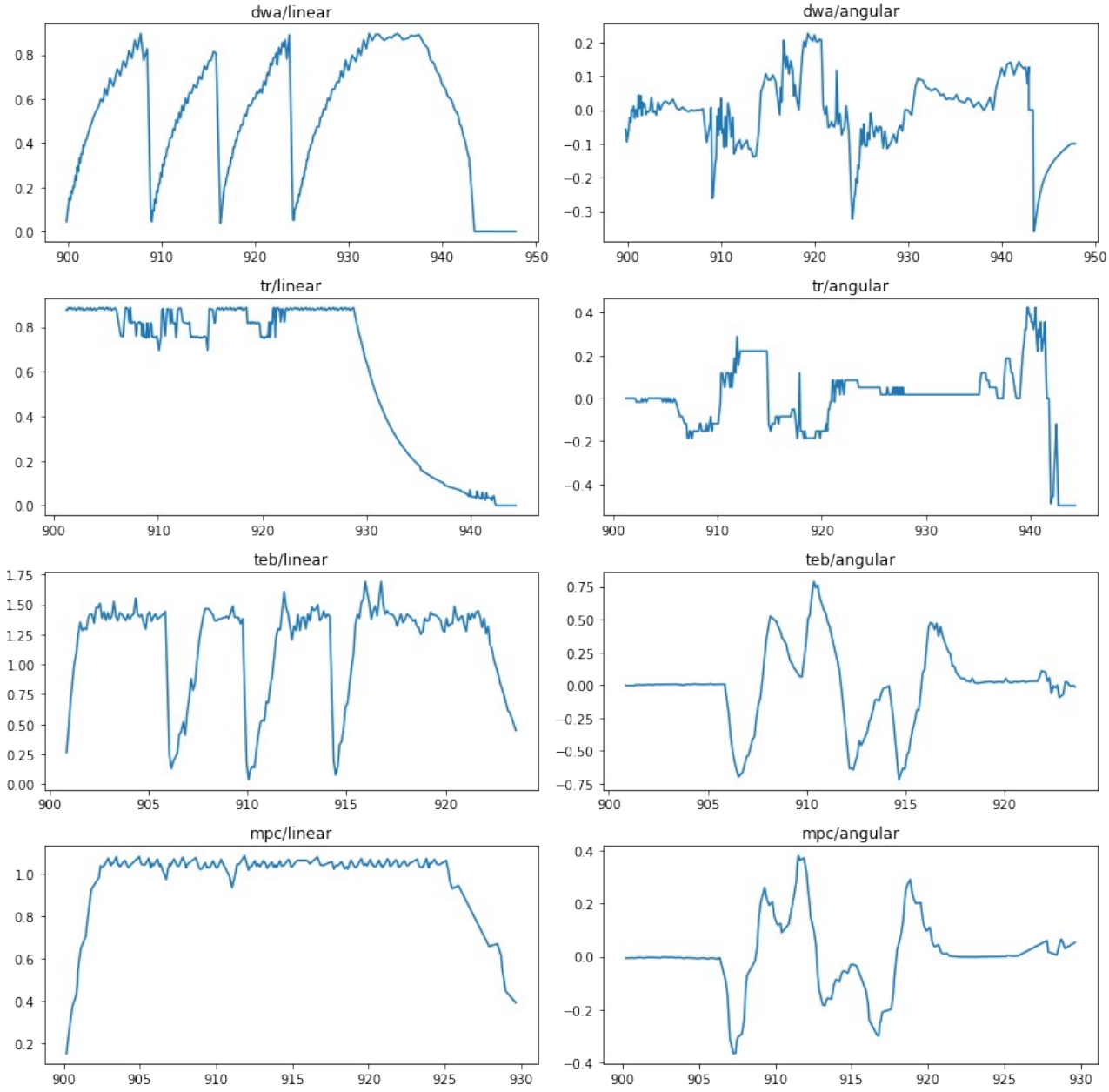

Рисунок 27. Колебания скоростей в полностью неизвестной среде.

В случае полностью открытого пространства, хорошо видно, как различные алгоритмы реагируют на обнаруженные препятствия (напр.: *DWA* полностью останавливается, а *TR* пытается сохранять скорость).

График отклонения от глобального маршрута показан на рисунке 28.

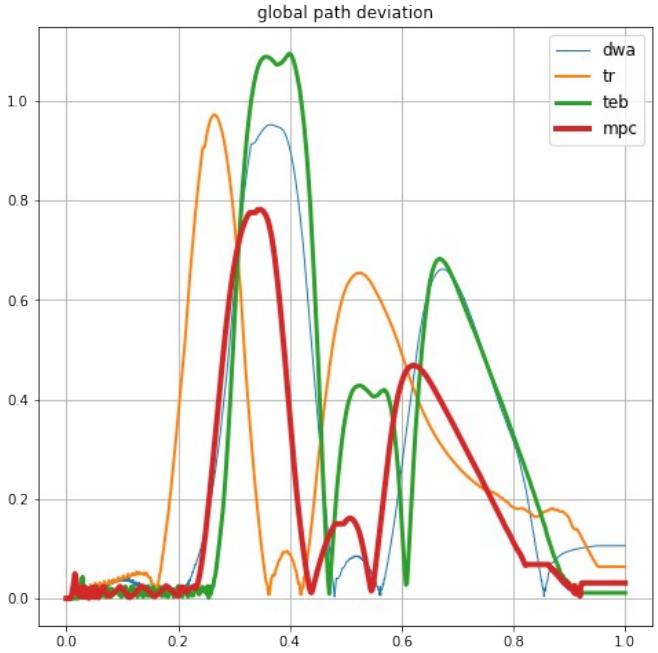

Рисунок 28. Отклонение траектории движения в полностью неизвестной среде.

Оценки колебания скоростей представлены в таблице 11.

Таблица 11. Колебания скоростей в полностью неизвестной среде.

	Линейная	Угловая	Общая
DWA	15.770	33.423	49.192
TR	11.532	20.797	32.329
TEB	15.201	9.145	24.346
MPC	9.051	13.544	22.595

Оценки по заданным критериям показаны в таблице 12.

Таблица 12. Оценки алгоритмов в полностью неизвестной среде.

	Колеб. маршр.	Колеб. скоростей	Средн. скорость м/с	Макс. скорость м/с	Отклон. машр.	Длит. движ. с	Частота запуска Гц
DWA	0.8289	62.8137	0.4267	0.8957	0.5615	47.980	6.6225
TR	0.9114	44.9675	0.5987	0.8873	0.5151	43.102	9.9010
TEB	0.7459	34.4234	1.1770	1.6921	0.3108	22.691	10.0000
MPC	0.6979	33.4482	0.9859	1.0873	0.2495	29.457	9.9010

Нормализованные оценки приведены в таблице 13.

Таблица 13.

Нормализованные оценки алгоритмов в полностью известной среде.

	Колеб. маршр.	Колеб. скоростей	Средн. скорость	Макс. скорость	Отклон. машр.	Длит. движ.	Частота запуска
DWA	0.8419	0.5325	0.3625	0.5294	0.4443	0.4729	0.6623
TR	0.7657	0.7438	0.5087	0.5244	0.4843	0.5264	0.9901
TEB	0.9355	0.9717	1.0000	1.0000	0.8027	1.0000	1.0000
MPC	1.0000	1.0000	0.8376	0.6426	1.0000	0.7703	0.9901

Суммарная оценка каждого алгоритма приведена в таблице 14.

DWA	***TR***	***TEB***	***MPC***
3.8458	4.5434	6.7099	6.2406

Таблица 14. Оценка каждого алгоритма для полностью неизвестной среды.

4.6 Выводы

Для исследуемых алгоритмов составлена оценка их поведения в трех разных средах: (1) Полностью известная среда; (2) Частично известная среда; (3) Неизвестная среда. Результаты оценки представлены в таблице 1.

Таблица 15. Результаты оценки алгоритмов.

	Известная среда	*Частично известная среда*	*Неизвестная среда*
DWA	3.0316	3.9251	3.8458
TR	3.2993	3.7658	4.5434
TEB	6.5743	6.9183	6.7099
MPC	5.5563	5.9872	6.2406

Для робота с дифференциальным двигателем лучшим выбором является *Timed-Elastic-Band*. Он предоставляет траекторию движения оптимальную по времени, используя при этом все возможности кинематики и динамики для такого робота.

Каждый из алгоритмов имеет свои ярко-выраженные особенности, которые подходят для той или иной среды.

DWA: Медленно набирает скорость и плавно останавливается в точке назначения. Подход *DWA* не требует высоких аппаратных требований, однако его реализация в ROS не предусматривает многопоточное выполнение. *DWA* – удачный выбор для тяжелых маломобильных роботов в известной среде.

TR: Посылает в контроллер команду наибольшей линейной скорости, согласно пространственным ограничениям (так чтобы робот мог вовремя остановиться). Производительность на уровне *DWA*. Является хорошим выбором для роботов с низкой производительностью, способных наиболее быстро разгоняться до своей максимальной скорости.

TEB: Алгоритм в основе которого лежит решение системы уравнений, выраженной в виде задачи многокритериальной оптимизации. Из этого следует, что аппаратные требования значительно увеличиваются, однако реализация алгоритма очень хорошо оптимизирована, благодаря фреймворку g2o, оптими-

зация матрицы происходит в многопоточном режиме. Команды для контроллера не являются гладкими, однако алгоритм направлен на то, чтобы как можно быстрее достигнуть точки назначения. TEB подходит для высокомобильных роботов с дифференциальным или передним приводом.

MPC: Усложненный алгоритм основанный на многкритериальной оптимизации. Его особенностью является то, что он справляется с любой динамической системой и поддерживает жесткие ограничения (ограничения в TEB аппроксимируются полиномиальной функцией). Аппаратные ограничения значительно выше. MPC используется в случаях когда необходимо строго следовать заданному маршруту или когда динамика сложнее автомобилеподобных роботов, например велосипед.

5 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

Основной задачей данной работы являлось исследование поведения робота при применении различных алгоритмов локального робота в симмуляционной среде. В настоящее время одним из важнейших этапов разработки ПО в робототехнике является проведение экспериментов в симуляторе, поэтому необходимо уделить внимание эргономике интерфейсов взаимодействия в с ПО для навигации и симуляции пространства.

В связи с широким развитием области информационных технологий и масштабом производимой продукции, к программным средствам также применяются стандарты, требующие создания наиболее удобных пользовательских интерфейсов. От пользовательского интерфейса зависят качество и скорость проводимых экспериментов.

5.1 Основные положения об эргономике

Требования по разработке интерфейса человек – система изложены в ГОСТ Р ИСО 9241-100 [26]. Этот документ необходимо использовать при оценке эргономики ПО.

Согласно ГОСТ, эргономика – это наука рассматривающая вопросы взаимодействия человека с другими элементами интерактивной системы.

Основной задачей ГОСТ является введение принципов, рекомендаций и требований для пользовательских интерфейсов, обеспечивающих диалог между пользователем и системой, рассматриваемый как последовательность действий (ввод) и ответных реакций системы (вывод), направленный на достижение цели и выполнение производственной задачи.

Стандарты по эргономике ПО применимы ко всем компонентам системы, включая:

- прикладные программы

- операционные системы
- встроенное ПО
- программные средства проектирования
- вспомогательные технологии

Стоит принять во внимание, что стандарты по эргономике носят больше рекомендательный характер, т. к. описываемые требования к пользовательским интерфейсам не обязательны. Однако соблюдение этих требований позволяет предотвратить возникновение будущих проблем с пригодностью ПО к использованию.

В ГОСТ Р ИСО 9241-100 приведены семь высокоуровневых эргономических принципов для разработки диалога между пользователем и системой:

1. Пригодность для выполнения задачи
2. Информативность
3. Соответствие ожиданиям пользователя
4. Пригодность для обучения
5. Управляемость
6. Устойчивость к ошибкам
7. Пригодность для индивидуализации

Данные принципы направлены на понимание требований в области эргономики и нет строгой необходимости в проверке их выполнения.

5.2 Требования к эргономике используемого ПО

Для разработки ПО, использующего различные алгоритмы планирования локальных траекторий колесных роботов и создания среды в симуляторе использовалась среда разработки Visual Studio Code (VSCode). В данной среде разработки соблюдаются все основные принципы диалога:

1. ***Пригодность для выполнения задачи***: для *VSCode* существует множество встроенных расширений, позволяющих проводить разработку, от-

ладку и сборку как отдельных модулей так и всего проекта. *VSCode* также поддерживает систему контроля версий *Git*, что ускоряет процесс разработки.

2. ***Информативность***: все заголовки и подзаголовки меню имеют понятное название и полностью предсказуемое поведение. Около каждого возможного действия из меню указаны горячие клавиши, вызывающее это действие. Все элементы связанные с организацией проекта имеют собственные иконки и всплывающие подсказки. Иерархия файлов проекта представлена в виде закрывающегося дерева, все популярные типы файлов имеют собственные иконки и могу быть дополнены расширениями. *VSCode* также отображает состояние всех исполняемых процессов в статусной панели.
3. ***Соответствие ожиданиям пользователя***: процесс сборки сопровождается оповещениями о текущем состоянии. Все ошибки замеченные во время написания кода отслеживаются и *VSCode* сразу указывает строки, файлы, и директории, которые содержат ошибки синтаксиса.
4. ***Пригодность для обучения***: *VSCode* содержит обширную документацию по всем компонентам среды, также предусмотрены руководства по использованию и разработке собственных расширений или аугментаций.
5. ***Управляемость***: *VSCode* имеет собственный интегрированный терминал, что позволяет сохранять историю всех запущенных процессов. Каждый процесс может быть запущен и остановлен в любое время. Все действия *VSCode* могут выполняться как мышью, так и горячими клавишами.
6. ***Устойчивость к ошибкам***: проект каждый промежуток времени кэшируется, что позволяет не беспокоиться за сохранность данных в случае отказа приложения. Удаление файлов требует подтверждения, а контроль версий позволит восстанавливать случайные изменения в проекте.

7. ***Пригодность для индивидуализации***: *VSCode* имеет полностью настраиваемую цветовую палитру. Все используемые шрифты могут быть изменены пользователем. Для каждого типа файлов может быть установлено пользовательское форматирование. Библиотека расширений *VSCode* имеет множество инструментов, позволяющих настроить среду для абсолютно любой задачи.

Следовательно, VSCode имеет эргономичный интерфейс и соответствует всем высокоуровневым принципам стандартов в ГОСТ Р ИСО 9241-100.

5.3 Оценка эргономики разработанной модели

Для оценки эргономики ПО, обеспечивающего проведение экспериментов необходимо определить из каких компонентов состоит система:

- Симулятор (Gazebo)
- Бортовое ПО (ROS)
- Визуализатор данных (Rviz)

Далее разобран каждый из компонентов.

Симулятор:

Предоставляет физически корректную среду, состоящую как из робота, так и окружающих его объектов. Состояние каждого объекта можно просмотреть в меню навигации. Модель робота и окружающие его препятствия могут иметь любую форму и характеристики, задаваемые пользователем. Время в симуляторе идет параллельно, но не всегда совпадает с реальным, например в моменты высокой нагрузки на систему. Сам симулятор широко поддерживается сообществом имеет множество плагинов и руководств. Так же симулятор поставляется вместе с контроллером, эмулирующим работу реального робота.

Бортовое ПО:

ROS не имеет графического интерфейса поскольку предназначен для исполнения на борту робота. Однако предоставлен широкий набор инструментов

для управления из командной строки. Архитектура ROS – это набор узлов, общающихся между собой, предоставляя тем самым гибкое управление устройством. Важным элементом является низкоуровневая реализация управления контроллерами робота, что решает основную задачу робототехники: связь ПО и аппаратной части робота. При отказе какого либо узла ROS, он по умолчанию перезапускается, что обеспечивает отказоустойчивость. Текущее состояние узлов, можно посмотреть с помощью *rqt_graph*, пример конфигурации, используемой в данной работе представлен на рисунке 29.

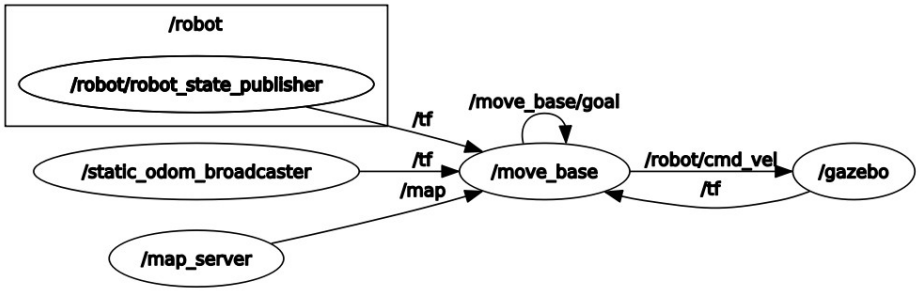

Рисунок 29. Пример конфигурации ROS (*rqt_graph*) для наземной навигации.

ROS также предоставляет инструментарий для регулировки узлов в реальном времени – *rqt_reconfigure*, пример настроек для локального планировщика показан на рисунке 30.

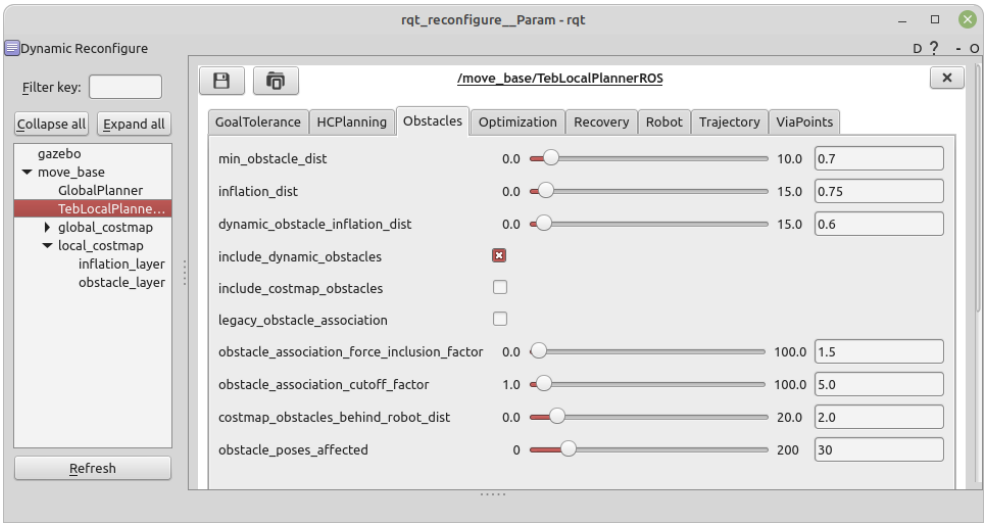

Рисунок 30. Окно настроек *rqt_reconfigure*.

Визуализатор: инструмент предназначенный для графической интерпретации данных, которыми оперируют узлы ROS. Данные можно выбрать как по типу сообщения так и с помощью топика, в который эти сообщения публикуются. Различные данные отображаются по разному: траектория в виде линии, позиция в виде стрелки с началом в указанной точке, окружение в виде тепловой карты и т. д. Если вдруг из каких-то топиков не отвечает, то его данные не отображаются, не нарушая при этом работу визуализатора. Пример визуализации представлен на рисунке 31.

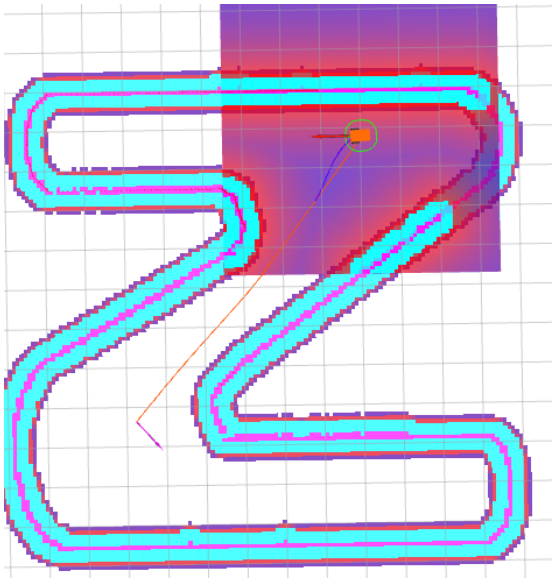

Рисунок 31. Пример визуализации данных в Rviz.

5.4 Вывод

Разработанное ПО для проведения экспериментов по оценке алгоритмов локального планирования построено на базе рассмотренного ПО, следовательно удовлетворяет принципам диалога. Запуск каждого сервиса сопровождается сообщениями, описывающими текущий статус запуска, а затем статусом работы. Запущенный комплекс программ, ожидает публикации сообщения в топик «*goal*» ROS средствами командной строки или инструментом Rviz. Инструкция по запуску приложена в приложении А.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было исследовано четыре алгоритма локального планирования траекторий колесных роботов. Выявлен лучший алгоритм, предоставляющий роботу способность справить с любой плоской средой.

В процессе работы были выполнены следующие поставленные задачи:

- Изучены современные используемые алгоритмы локального планирования траекторий.
- Разработано ПО, применяющее алгоритмы в симуляции.
- Определены метрики, позволяющие оценить алгоритмы по разным критериям.
- Определена эффективность применения алгоритмов в неизвестной среде.

На текущий день существует множество подходов для следования вдоль глобального маршрута. Однако для использования алгоритма на колесном роботе, он должен удовлетворять требованиям кинематики. Исследования алгоритмов проводились на роботе с дифференциальным приводом. Несмотря на то, что у такого робота ограничения ниже, чем у робота с передним или задним приводом, оно все еще может двигаться только вдоль дуг окружности. Данному ограничению соответствуют классические алгоритмы *Dynamic Window Approach*, *Trajectory Rollout*, *Timed-Elastic-Band*, *Model Predictive Control* и различные подходы, основанные на машинном обучении, которые пока еще широко не исследованы и не имеют открытых реализаций.

Разработано ПО на базе ROS, позволяющее запустить симуляцию робота и управляющий контроллер. Симуляция робота может быть запущена в разных средах. Данная симуляция предоставляет систему управления, аналогичную той, что используется на реальных роботах. Управляющий контроллер построен на базе ROS Navigation Stack, позволяющий указать точку назначения, который затем направляет команды роботу для достижения этой точки.

Контроллер может запускаться с различными планировщиками и картами местности.

Определен набор метрик для оценки эффективности алгоритма:

- Колебание построенного маршрута
- Колебание команд скорости
- Отклонение от глобального маршрута
- Средняя скорость движения
- Максимальная скорость движения
- Время достижения точки назначения
- Частота запуска алгоритма

В реальных условиях каждая из данных метрик оценивается в соответствие со средой и условиями движения робота. В данной работе каждая из метрик является равнозначной.

Для оценки алгоритмов было написано ПО для узла графа *ROS*, который указывает точку назначения робота, после чего начинает запись движения робота с помощью *rosbag*. По достижения точки назначения, запись прекращается. После записи всех экспериментов. Была проведена оценка их работы с помощью библиотек *Numpy*, *SciPy* и *Pandas* для *Python*. *Python* был выбран, поскольку имеет удобный инструментарий для чтения *rosbag* записей.

В ходе оценки алгоритмов выявлено, что лучшим алгоритмом построения траектории для робота с дифференциальным приводом и ограниченным зрением, является *Timed-Elastic-Band*. Алгоритм ставит перед собой задачу наиболее быстрого движения вдоль контрольных точек глобального маршрута. В сегодняшние дни алгоритм имеет множество оптимизаций, для него создан фреймворк *g2o*, быстр оптимизирующий разреженные матрицы в многопоточном режиме.

ЛИТЕРАТУРА

1. Karur K. et al. A Survey of Path Planning Algorithms for Mobile Robots // Vehicles. – 2021. – С. 3. – №. 3. – P. 448-468.
2. Stachniss C. Robotic mapping and exploration. – Springer, 2009. – Т. 55.
3. Lozano-Pérez T., Wesley M. A. An algorithm for planning collision-free paths among polyhedral obstacles //Communications of the ACM. – 1979. – V. 22. – P. 560-570.
4. Alajlan M. et al. Global path planning for mobile robots in large-scale grid environments using genetic algorithms //2013 International Conference on Individual and Collective Behaviors in Robotics (ICBR). – IEEE, 2013. – P. 1-8.
5. Paden B. et al. A survey of motion planning and control techniques for self-driving urban vehicles //IEEE Transactions on intelligent vehicles. – 2016. – V. 1. – P. 33-55.
6. Zhou C., Huang B., Fränti P. A review of motion planning algorithms for intelligent robots //Journal of Intelligent Manufacturing. – 2021. – P. 1-38.
7. Dijkstra E. W. et al. A note on two problems in connexion with graphs // Numerische mathematik. – 1959. – V. 1. – P. 269-271.
8. Hart P. E., Nilsson N. J., Raphael B. A formal basis for the heuristic determination of minimum cost paths //IEEE transactions on Systems Science and Cybernetics. – 1968. – V. 4. – P. 100-107.
9. LaValle S. M., Kuffner Jr J. J. Randomized kinodynamic planning //The international journal of robotics research. – 2001. – V. 20. – P. 378-400.
10. González D. et al. A review of motion planning techniques for automated vehicles //IEEE Transactions on intelligent transportation systems. – 2015. – V. 17. – P. 1135-1145.
11. Kavraki L. E. et al. Probabilistic roadmaps for path planning in high-

dimensional configuration spaces //IEEE transactions on Robotics and Automation. – 1996. – V. 12. – P. 566-580.

12. Dubins L. E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents // American Journal of mathematics. – 1957. – V. 79. – P. 497-516.
13. Murphy R. R. Introduction to AI robotics. – MIT press, 2019.
14. Khatib O. Real-time obstacle avoidance for manipulators and mobile robots // Autonomous robot vehicles. – Springer, New York, NY, 1986. – P. 396-404.
15. Fox D., Burgard W., Thrun S. The dynamic window approach to collision avoidance //IEEE Robotics & Automation Magazine. – 1997. – V. 4. – P. 23-33.
16. Rösmann C. et al. Trajectory modification considering dynamic constraints of autonomous robots //ROBOTIK 2012; 7th German Conference on Robotics. – VDE, 2012. – P. 1-6.
17. Gerkey B. P., Konolige K. Planning and control in unstructured terrain //ICRA workshop on path planning on costmaps. – 2008.
18. Quinlan S., Khatib O. Elastic bands: Connecting path planning and control // [1993] Proceedings IEEE International Conference on Robotics and Automation. – IEEE, 1993. – P. 802-807.
19. Rösmann C., Makarow A., Bertram T. Online motion planning based on nonlinear model predictive control with non-Euclidean rotation groups //2021 European Control Conference (ECC). – IEEE, 2021. – P. 1583-1590.
20. Cutler C. R., Ramaker B. L. Dynamic matrix control?? a computer control algorithm //joint automatic control conference. – 1980. – P. 72.
21. Dudek G., Jenkin M. Computational principles of mobile robotics. – Cambridge university press, 2010.

22. ROS/Concepts - ROS Wiki [Электронный ресурс]. URL: http://wiki.ros.org/ROS/Concepts
23. rviz - ROS Wiki [Электронный ресурс]. URL: http://wiki.ros.org/rviz
24. Koenig N., Howard A. Design and use paradigms for gazebo, an open-source multi-robot simulator //2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566). – IEEE, 2004. – V. 3. – P. 2149-2154.
25. navigation/Tutorials/RobotSetup - ROS Wiki [Электронный ресурс]. URL: http://wiki.ros.org/navigation/Tutorials/RobotSetup
26. ГОСТ Р ИСО 9241-110-2016 Эргономика взаимодействия человек-система. Часть 110. Принципы организации диалога. М.: Стандартинформ, 2016.

ПРИЛОЖЕНИЕ А

Инструкция по запуску

Установка ROS

```
5.  sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-lat-
est.list'
6.  sudo apt install curl # if you haven't already installed
curl
7.  curl -s https://raw.githubusercontent.com/ros/rosdistro/
master/ros.asc | sudo apt-key add -
8.  sudo apt install ros-noetic-desktop-full
9.  echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
10. source ~/.bashrc
11. sudo apt install python3-rosdep python3-rosinstall
python3-rosinstall-generator python3-wstool build-essential
12. sudo apt install python3-rosdep
13. sudo rosdep init
14. rosdep update
```

Установка Gazebo

```
15. sudo sh -c 'echo "deb http://packages.osrfoundation.org/
gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/
sources.list.d/gazebo-stable.list'
16. wget https://packages.osrfoundation.org/gazebo.key -O - |
sudo apt-key add -
17. sudo apt-get update
18. sudo apt-get install gazebo11
```

Запуск симулятора

```
19. :local_planner_research $ cd simulation_ws
20. :local_planner_research $ catkin_make
21. :local_planner_research/simulation_ws $ source devel/set-
up.sh
22. # map can be <empty, zig-zag, corner, outdoor>
23. :local_planner_research/simulation_ws $ roslaunch
robot_gazebo robot_gazebo map:=empty
```

Запуск системы навигации

```
24. # after simulation run
25. :local_planner_research $ cd catkin_ws
26. :local_planner_research/catkin_make $ catkin_make
27. :local_planner_research/catkin_make $ source devel/set-
up.sh
28.
29. # availble arguments (can be undefined): map, local_plan-
ner, autogoal
30. # map -- map which used to build global plan. Available
<zig-zag, corner, outdoor>
31. # local_planner -- approach to build motion trajectory.
Available <dwa, tr, teb, mpc>
32. # auto_goal -- flag for automatically set goal pose and
start recording. Available <false, true>
33. :local_planner_research/catkin_make $ roslaunch nav_2d
robot_navigation
```