Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Направление подготовки: 09.04.04 – Программная инженерия

Магистерская программа: Робототехника

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА АЛГОРИТМА ИССЛЕДОВАНИЯ И КАРТОГРАФИРОВАНИЯ ОКРУЖАЮЩЕЙ СРЕДЫ РОБОТОМ TURTLEBOT В УСЛОВИЯХ ОГРАНИЧЕННОГО РЕСУРСА БАТАРЕИ

Обучающийся 2 курса

группы 11-031 Маврин И.А.

Научный руководитель

PhD (технические науки), доцент,

профессор кафедры

интеллектуальной робототехники Магид Е.А.

Директор ИТИС КФУ

канд. техн. наук Абрамский М.М.

Казань – 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Робототехника, как наука, существует уже достаточно длительное время [1]. Однако именно в последние десятилетия она переживает существенный подъем. Научные исследования расширяют возможности роботов, тем самым расширяя области их практического применения. Продажи роботов растут из года в год [2], при этом одним из основных драйверов этого роста является мобильная робототехника [3].

Мобильные роботы широко применяются во многих областях. Например, таких, как складское и логистическое использование [4], поисково-спасательные операции [5], медицина [6], военная робототехника [7] и сфера обслуживания [8]. Промышленные и военные роботы, как правило, проектируются с учетом их использования в опасных или недоступных для человека территориях.

Существенным драйвером роста робототехники является значительный научно-технический прогресс в последние десятилетия. Аппаратная составляющая роботов с каждым годом улучшается. Например, совершенствуются технологии производства аккумуляторов, что значительно увеличивает их емкость и скорость зарядки [9]. Разрабатываются более продвинутые контроллеры процессов заряда и разряда батареи [10]. Также прилагаются значительные попытки снизить расход заряда батареи, например, путем использования более эффективных электромоторов [11].

Тем не менее, фундаментальная проблема остается не решенной – емкость бортовой батареи является ограниченной. Несмотря на то, что эта проблема не всегда является критически важной, существует следующая рекомендация: если какой-либо ресурс является ограниченным, имеет смысл использовать его рационально. Разумеется, существуют и другие способы решения проблемы ограниченной емкости бортовой батареи, например, замена разряженной батареи на заряженную при помощи другого робота или

автоматической станции [12]. Тем не менее, при использовании других способов также имеет смысл рационально использовать заряд батареи, поскольку придется реже производить её замену.

Одной из самых часто встречающихся задач для мобильных роботов является задача исследования неизвестной среды. Зачастую исследуемые среды могут быть опасны или недоступны для человека. Примерами такой задачи могут служить обследование разрушенных зданий во время чрезвычайных происшествий [13] или составление карты радиационных загрязнений территории [14].

Целью данной работы является улучшение существующего алгоритма исследования неизвестной среды E^3 [**15**] для того, чтобы убрать самое существенное его ограничение – невозможность исследования неизвестных сред с препятствиями. Далее необходимо провести экспериментальное сравнение улучшенной версии алгоритма E^3 с несколькими уже существующими алгоритмами по энергоэффективности. **Актуальность работы** состоит в том, что обозначенное ограничение делает алгоритм E^3 практически неприменимым для реальных задач, поскольку реальные задачи, как правило, подразумевают исследование сред с препятствиями. **Новизна работы** состоит в том, что на данный момент не существует работ, которые были бы посвящены исправлению указанного ограничения с целью обеспечения практической применимости теоретического алгоритма E^3.

В качестве робота, на котором реализуется алгоритм и проводится экспериментальное сравнение, решено было использовать Turtlebot 3 Burger [16]. Детальное описание робота содержится в главе 3.3.

1 ОБЗОР ЛИТЕРАТУРЫ

В данной главе приводится краткое описание работ, посвященных задаче исследования неизвестной среды и нескольким смежным темам, которые необходимы для понимания предметной области. Кроме того, приведено несколько работ другой тематики, но полезных для выполнения настоящей работы.

Статья [17], является одной из самых известных и наиболее часто цитируемых работ по теме исследования неизвестной среды. В ней описывается алгоритм Frontier exploration, в котором был применен принципиально новый подход к исследованию среды, основанный на поиске границ между исследованными и неисследованными территориями. В работе описывается способ представления карты исследуемой территории в памяти компьютера и способ поиска границ в этом представлении. Авторами было проведено тестирование алгоритма Frontier exploration на реальном роботе, которое показало его способность исследовать как большие и открытые пространства, так и узкие пространства, загроможденные препятствиями. Несмотря на то, что статья была опубликована в 1997 году, алгоритм Frontier exploration всё еще актуален, поэтому он выбран в настоящей работе в качестве алгоритма для сравнения.

Статья [18], в которой авторы проводят сравнительный анализ наиболее эффективных подходов для поиска границ. Помимо этого, они предлагают собственный подход под названием Frontier-Tracing Frontier Detection (FTFD). Предложенный подход тестируется как в робототехническом симуляторе Gazebo [19], так и на реальном роботе. По результатам сравнения авторы приходят к выводу, что на данный момент не существует абсолютного лидера среди всех подходов, который бы показывал наилучший результат во всех экспериментальных тестовых средах. Вместо этого ими предлагается идея о динамическом выборе подхода, наиболее подходящего для исследуемой

среды. Подходы для поиска пограничных областей, перечисленные в этой статье, в настоящей работе не используются. Однако это не делает сравнение, проведенное в настоящей работе, некорректным, поскольку использование более эффективного подхода способно повлиять только на время полного исследования неизвестной среды, но не на общее пройденное расстояние. Данная статья ценна для настоящей работы тем, что она показывает, что алгоритм Frontier exploration в настоящее время является актуальным и широко используемым.

Работа [20] описывает решение задачи поиска скрытого объекта в неизвестной среде при помощи алгоритмов машинного обучения. Эта тема является актуальной, поскольку такая задача очень часто возникает при использовании мобильных роботов для поисково-спасательных операций. В этом случае время, затраченное на поиск объекта, является критически важным, поэтому авторы предлагают собственный алгоритм для оптимального планирования маршрута движения робота при поиске объекта. Предложенный авторами алгоритм может работать в двух режимах: поиск объекта по излучаемому им радиосигналу и поиск без дополнительной информации о расположении объекта путем прочесывания местности. При этом следует отметить, что алгоритмы машинного обучения (а именно обучение с подкреплением) используются только в режиме поиска объекта по излучаемому радиосигналу.

В статье [21] авторы описывают разработанный ими алгоритм для навигации в неизвестной среде для робота, у которого отсутствует сенсор, способный измерять точные расстояния до препятствий. По условиям задачи, поставленной авторами, робот имеет всего один сенсор, который может определять только в каком направлении происходит резкое изменение расстояния до ближайшего препятствия, то есть способен находить грани препятствий. Поскольку робот не способен замерять точные расстояния до препятствий, авторы не используют системы координат. Также невозможна и

локализация робота на карте местности. При этом, робот способен запоминать карту среды в виде древовидной структуры данных, которая позволяет ему осуществлять навигацию и строить локально оптимальные маршруты. Из-за того, что для такого робота отсутствует понятие расстояния, построенные им маршруты не являются глобально оптимальными. Авторы работы приходят к выводу, что глобально оптимальная навигация теоретически невозможна для такого типа роботов.

Авторы работы [22] изучают задачу автономного исследования неизвестной среды. Решение этой задачи путем создания множества условных правил (то есть конструкций *if…then…else*) является невозможным, поскольку невозможно покрыть условиями огромное количество комбинаций различных неизвестных сред и сенсоров, устанавливаемых на мобильных роботах. Таким образом, авторы приходят к выводу, что гораздо правильнее будет решать эту задачу при помощи алгоритмов машинного обучения. В своей работе они представляют разработанный ими фреймворк для решения задачи исследования неизвестной среды путем разделения этой задачи на три этапа: принятие решения, планирование и картографирование. Также авторы предлагают алгоритм глубокого обучения с подкреплением, который при помощи глубокой нейронной сети способен выучить стратегию исследования неизвестной среды, изучая частичную карту данной среды.

Статья [23] содержит в себе обзор подходов и алгоритмов планирования маршрутов движения для мобильных роботов в неизвестной среде. Авторы изучили наиболее заметные работы по теме оптимизации алгоритмов для планирования маршрутов движения, систематизировали их и предложили следующую классификацию. Все алгоритмы планирования маршрутов разделяются на две большие группы. Первая группа содержит в себе алгоритмы, которые используют классический подход к решению задачи планирования (например, алгоритм Дейкстры или A*). Вторая группа алгоритмов использует эволюционный и гибридный подходы к решению

задачи планирования. Алгоритмы этой группы используют машинное обучение, нейронные сети, генетические алгоритмы, обучение с подкреплением и системы на основе нечеткой логики. При этом такие методы могут применяться как самостоятельно, так и в комбинации с классическими алгоритмами. Однако стоит отметить, что в данной работе авторы концентрируют свое внимание на алгоритмах планирования маршрута, не затрагивая при этом тему непосредственно исследования неизвестной среды.

В публикации [24] авторы изучают проблему оптимального исследования неизвестной территории. В своей работе они приводят описание собственного алгоритма Environmental Ergodic Exploration (E^3), который предназначен для оптимального исследования неизвестной территории, регионы которой имеют различную степень важности. Это алгоритм оптимального контроля, который пытается оптимизировать траекторию движения робота по нескольким критериям. Во-первых, минимизируется метрика так называемой *эргодичности* траектории, то есть разница между средневременным поведением системы на указанной траектории и распределением количества информации на этой траектории. Во-вторых, минимизируются затрачиваемые усилия на выполнение данной траектории. Основной научной новизной алгоритма E^3 является функция для количественной оценки информации в точках траектории, которая используется при оптимизации траектории. В экспериментах, проведенных авторами на реальном роботе, успешно получилось исследовать двумерный участок территории. Больше информации по алгоритму E^3 содержится в докторской диссертации [25].

Работа [26] является дальнейшим развитием некоторых концепций из алгоритма E^3. Авторы утверждают, что расчет эргодичности траектории при помощи базисных функций Фурье, как в оригинальном алгоритме E^3, имеет некоторые недостатки. Они предлагают новый принцип расчета эргодичности, который не имеет указанных недостатков. Помимо нового принципа расчета

эргодичности, авторы приводят описание собственного алгоритма исследования среды, основанного на решении транспортной задачи. Также авторы провели тестирование собственного алгоритма в тестовых средах с равномерным и неравномерным распределением информации. Результаты экспериментов показали, что практическая эффективность предложенного алгоритма лучше, чем аналитически найденная верхняя граница для данной задачи.

В статье [27] авторы расширяют предыдущие наработки для решения задачи исследования неизвестной среды при помощи нескольких мобильных роботов. Они описывают концепцию децентрализованной модели исследования неизвестной среды, где не требуется единственный руководитель, который будет распределять задачи. В предложенном авторами алгоритме предполагается, что роботы будут обмениваться информацией об уже исследованных областях при появлении друг друга в зоне видимости. Помимо этого, авторы предлагают метод оценки производительности исследования, который обладает невысокой вычислительной сложностью. Авторы экспериментально подтверждают работоспособность собственного алгоритма тестированием в симуляторе. Следует отметить, что в обеих научных работах имеется следующий недостаток: авторы полностью концентрируются на математической теории и не уделяют внимание практическим деталям реализации. Например, в обеих работах не учитывается, что в исследуемых средах могут находиться препятствия.

В статье [28] авторы работают над задачей исследования частично неизвестной среды. Они предлагают новый алгоритм исследования помещений, который спроектирован так, чтобы учитывать ограничения области видимости сенсоров робота и предсказуемую структуру подобных помещений. Кроме того, авторы приводят описание жадного алгоритма исследования неизвестной среды и детали реализации обоих алгоритмов с использованием фреймворка Robot Operating System (ROS [29]). Тестирование

алгоритмов в симуляторе и на реальном роботе показало, что предложенный авторами алгоритм демонстрирует лучшие результаты. Следует отметить, что авторы вместо лазерного лидара использовали Kinect [30]. Нам не удалось запустить предложенный авторами алгоритм на роботе с лазерным лидаром, поскольку он предполагает, что сенсоры робота должны работать некоторым особым образом. При этом данные с лидара в ROS обрабатываются совсем не так, как ожидает этого предложенный авторами алгоритм. Более подробно это описано в главе 3.1. В жадном алгоритме подобные допущения отсутствуют, поэтому он используется в настоящей работе в качестве алгоритма для сравнения.

В работе [31] авторы полагают, что при исследовании необходимой среды необходима память для того, чтобы иметь возможность различать похожие области в исследуемой среде. Предложенный авторами алгоритм использует глубокую нейронную сеть для того, чтобы запоминать свой опыт и обучаться на нем. При этом, робот обучается таким образом не только оптимально планировать маршрут движения, но и избегать столкновений с препятствиями. Авторы провели тестирование собственного алгоритма в Gazebo и установили, что робот под управлением данного алгоритма способен автономно обучаться в реальном времени. Кроме того, после обучения в одной тестовой среде, он способен адаптироваться к работе в другой тестовой среде.

В [32] авторы предлагают собственную энергетическую модель типового мобильного робота. Данная модель позволяет подсчитывать и предсказывать количество электроэнергии, которое будет затрачено на выполнение того или иного действия. Использование энергетической модели позволит ранжировать предполагаемые варианты действий с целью того, чтобы затрачивать минимально возможное количество энергии для достижения какой-либо цели, что способствует повышению энергоэффективности действий робота. Авторы выделяют три основных составляющих собственной модели: систему сенсоров, систему управления и

систему передвижения. Для настоящей работы эта публикация полезна тем, что в ней авторы проводят сравнительный анализ потребления электроэнергии каждой из трех составляющих своей модели для мобильного робота типовой конструкции. Сравнительный анализ показывает, что система передвижения, то есть электромоторы, являются основными потребителями энергии. Именно на основании результатов сравнительного анализа составлялись критерии оценки энергоэффективности в настоящей работе.

Для того, чтобы провести экспериментальное сравнение алгоритмов исследования неизвестной среды, необходимо создать некоторое количество тестовых сред, однако их ручное создание занимает длительное время. В статье [33] авторы представляют разработанную ими утилиту для автоматического создания тестовых сред, подходящих для использования в симуляторе Gazebo. Данная утилита позволяет создавать 3D модели на основе данных с лазерного лидара, представленных в виде 2D изображения, или же на основе любого другого черно-белого 2D изображения. При этом важно отметить, что созданные 3D модели являются достаточно оптимальными и не снижают значительным образом производительность симулятора Gazebo. В настоящей работе данная утилита используется для создания тестовых сред, о чем подробно написано в главе 6.1.

2 ЗАДАЧА ИССЛЕДОВАНИЯ НЕИЗВЕСТНОЙ СРЕДЫ

2.1 МОТИВАЦИЯ И ПОСТАНОВКА ЗАДАЧИ

Задача исследования неизвестной среды подразумевает картографирование, то есть построение карты данной среды. В дальнейшем карта среды может быть использована как человеком (например, для проведения анализа и заключения каких-либо выводов), так и роботом (например, для построения кратчайшего из одной точки среды в другую).

Помимо задачи картографирования, во время исследования неизвестной среды необходимо решить задачу локализации, т.е. определить положение робота в данный момент на карте среды. Задачи картографирования и локализации тесно связаны друг с другом. Для решения задачи локализации требуется карта среды, а для картографирования необходимы данные о текущем положении робота. Поэтому в робототехнике эти две задачи часто решаются одновременно, при помощи алгоритмов семейства SLAM [34] (Simultaneous Localization And Mapping).

Задача картографирования неизвестной среды требует от мобильного робота перемещаться по территории данной среды. Почти в любом механизме передвижения, которые используются в мобильных роботах, перемещение в пространстве осуществляется при помощи моторов. При этом исследования показывают [32], что электромоторы являются самыми большими потребителями энергии среди стандартного оборудования мобильного робота. В процентном соотношении потребление энергии электромоторами может достигать 95% от общего потребления. Количество энергии в бортовой батарее робота ограничено, поэтому необходимо передвигаться по исследуемой территории наиболее рациональным образом. То есть алгоритм исследования неизвестной среды должен быть энергоэффективным.

Отдельно стоит отметить, что в робототехнике существует еще несколько достаточно популярных задач, которые обладают высокой

степенью сходства с задачей исследования неизвестной среды. Например, очень похожа на неё задача поиска в неизвестной среде [20, 35, 36]. Основной разницей является то, что поиск прекращается после нахождения объекта. Однако в случае, если требуется найти неизвестное количество объектов, задача поиска полностью сводится к задаче исследования всей неизвестной среды. Примерами таких задач служат поиск мин и неразорвавшихся снарядов после военных действий или поиск локальных возгораний в лесном массиве.

Задача патрулирования [37, 38, 39] также обладает множеством сходств с задачами поиска и исследования неизвестной территории. Задача патрулирования предполагает бесконечный процесс исследования территории с целью обнаружения каких-либо объектов. Основным отличием от задачи исследования является то, что при патрулировании обычно имеется карта среды. В случае, если точная и актуальная карта среды отсутствует и ограничена только область патрулирования, обе этих задачи становятся очень похожи. Примером такой задачи может служить патрулирование автомобильной парковки, на которой количество и расположение автомобилей каждый раз меняется.

2.2 КРИТЕРИИ ОЦЕНКИ ЭНЕРГОЭФФЕКТИВНОСТИ АЛГОРИТМОВ ИССЛЕДОВАНИЯ НЕИЗВЕСТНОЙ СРЕДЫ

Энергоэффективность – это рациональное использование энергетических ресурсов. Понятие энергоэффективности фактически является аналогом коэффициента полезного действия (КПД) [40], который показывает, насколько эффективно работает тот или иной механизм. КПД вычисляется следующим образом (2.2.1).

$$\eta = \frac{A}{Q} \tag{2.2.1}$$

В данной формуле A – объём полезной работы, а Q – количество энергии, затраченное при выполнении этой работы. Объем полезной работы для

полного исследования неизвестной среды является фиксированным, поэтому энергоэффективность алгоритма исследования полностью зависит от количества энергии, которое будет потрачено на исследование. Более энергоэффективный алгоритм будет расходовать как можно меньше энергии из бортовой батареи робота на исследование неизвестной среды.

Задача подсчета количества энергии, потраченной из бортовой батареи робота, не является тривиальной. Только по напряжению на клеммах батареи точно оценивать расход энергии не получится, поскольку зависимость напряжения от количества заряда в батарее нелинейная [41]. Для точного подсчета требуется устанавливать в разрыв электрической цепи специальную микросхему-счетчик [42], которая будет интегрировать по времени силу тока I_t, проходящего по этому участку электрической цепи (2.2.2).

$$Q = \int_{t_0}^{t_F} I_t \, dt \approx \sum_{t_0}^{t_F} I_t \, \Delta t \qquad (2.2.2)$$

Далеко не все роботы оборудованы соответствующим аппаратным обеспечением, поэтому точный подсчет расхода заряда бортовой батареи не всегда возможен. С другой стороны, в робототехнических симуляторах (например, в Gazebo) зачастую вообще отсутствует симуляция бортовой батареи робота. Поэтому для того, чтобы была возможность тестировать алгоритмы исследования неизвестной среды в симуляторе, необходимо было найти способ косвенной оценки расхода заряда бортовой батареи мобильного робота.

В статье [32] авторы оценивали расход энергии каждым элементом из стандартного набора оборудования, обычно устанавливаемого на робота. В результате они установили, что основными потребителями энергии являются электромоторы, на долю которых приходится до 95% потребляемой энергии. При этом абсолютные значения потребления энергии электромоторами полностью зависят от того, движется робот или стоит.

С другой стороны, остальные 5% от общей потребляемой энергии приходятся на различную электронику. Например, на электропитание вычислительных устройств и сенсоров робота. В абсолютном выражении количество энергии, потребляемое различной электроникой робота, всегда остается более-менее постоянным, независимо от того, движется робот или стоит. Таким образом, количество энергии, потребляемое электроникой робота, не зависит от оптимальности действий, выполняемых роботом во время исследования неизвестной среды. Эта постоянная составляющая будет одинаковой для каждого алгоритма исследования неизвестной среды, поэтому при оценке энергоэффективности одного алгоритма по сравнению с другим, её можно не учитывать.

В симуляторе Gazebo не реализована полная симуляция физических процессов, происходящих в электромоторах. Робот приводится в движение путем вращения 3D модели колеса относительно 3D модели робота с некоторой скоростью. Из этого следует, что единственным показателем, по которому можно каким-либо образом судить о потреблении энергии электромотором, является количество оборотов колеса робота. При этом не важно в каком направлении вращается колесо, потому что электромотор потребляет одинаковое количество энергии как в режиме движения вперед, так и в режиме заднего хода.

Для подсчета количества оборотов каждого из колес робота был реализован обработчик ROS. Этот обработчик вычислял и интегрировал абсолютное значение угла поворота колеса робота через малые промежутки времени, то есть фактически вычислял одометрию робота. В процессе тестирования было установлено, что суммарное количество оборотов всех колес робота, с точностью до умножения на константу, соответствует общему пройденному расстоянию. При этом значение данной константы является одинаковым для всех протестированных алгоритмов. Это позволяет сделать вывод, что в симуляторе, в отсутствии проскальзываний колес или

продолжения скольжения мобильного робота по подстилающей поверхности по инерции при полной блокировке колес (юз), метрики суммарного количества оборотов колес r и общего пройденного расстояния L являются эквивалентными (2.2.3), поэтому в таблицах с результатами сравнения в главе 6.2 для каждого алгоритма указано общее пройденное расстояние, как более наглядная и интуитивно понятная метрика.

$$L = 2\pi r \tag{2.2.3}$$

Отдельно следует отметить, что время, затраченное на полное исследование среды, не является подходящей метрикой для оценки энергоэффективности. Во-первых, в формуле расчета КПД не используется время, затраченное на выполнение полезной работы, то есть КПД не зависит от времени. Таким образом, более энергоэффективный алгоритм может исследовать неизвестную среду дольше менее энергоэффективного, однако при этом тратить на исследование меньше энергии. Во-вторых, метрика времени, затраченного на полное исследование среды, никак не учитывает какой процент этого времени робот двигался и какой процент – был неподвижен. В этом случае основной потребитель энергии бортовой батареи, то есть электромоторы, никак не учитывается.

Подводя итоги, можно сделать вывод, что использование времени, затраченного на полное исследование среды, в качестве метрики для сравнения алгоритмов по энергоэффективности, даст неправильные результаты сравнения и приведет к заключению неправильных выводов. Гораздо более правильным будет использовать для сравнения метрику общего пройденного расстояния.

3 ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ И ИНСТРУМЕНТОВ

3.1 ROBOT OPERATING SYSTEM (ROS)

Robot Operating System [29] (ROS) – это набор библиотек и инструментов для разработки программного обеспечения для роботов. ROS не является операционной системой в полном смысле этого термина, однако выполняет некоторые стандартные задачи операционной системы, такие как низкоуровневый контроль устройств, обеспечение аппаратной абстракции, реализацию часто используемых функций, передачу данных между процессами и механизм управления зависимостями и пакетами. ROS является мета-операционной системой и работает как надстройка над классической операционной системой, в качестве которой чаще всего используется Ubuntu или Debian. Также существуют экспериментальные версии ROS для Windows и Arch Linux.

ROS имеет графовую архитектуру, в которой обработка данных происходит в узлах (node или нода), которые способны обмениваться данными по ребрам графа (topic или топик). Соединение узлов происходит по механизму Publisher – Subscriber, который подразумевает, что ноды могут подписываться на топики, получать из них информацию, которую после обработки могут отправлять в другие топики.

В составе ROS имеется набор пакетов под общим названием ROS Navigation Stack (стек навигации ROS), который содержит в себе реализации различных алгоритмов, необходимых мобильным роботам для навигации в пространстве. Стек навигации ROS предполагает использование пакета *move_base* для указания мобильному роботу целевой точки, которой он должен достигнуть. Целевая точка указывается через ROS Action API – стандартного интерфейса для постановки задач, контроля прогресса их выполнения или отмены. Пакет *move_base* реализует интерфейс, который

принимает на вход целевую точку на карте, строит маршрут до неё и в автоматическом режиме управляет роботом во время движения по маршруту. Фактически, стек навигации позволяет управлять роботом, не вникая в детали его устройства, предоставляя для этого пакет *move_base*.

Однако необходимо отметить, что для робота должна быть реализация интерфейса, которая преобразует высокоуровневые команды поступательной и вращательной скоростей в низкоуровневые команды для управления непосредственно сервоприводами. Робот Turtlebot 3 Burger, который используется в данной работе для тестирования алгоритмов, отличается от других подобных роботов тем, что все программное обеспечение, необходимое для работы стека навигации, разрабатывается и поддерживается непосредственно производителем робота. Подробнее об этом написано в главе 3.3.

Задача исследования неизвестной среды подразумевает построение карты этой среды. Для преобразования данных с лазерного лидара робота в карту помещения используются алгоритмы семейства SLAM [34]. ROS содержит несколько различных алгоритмов SLAM, которые имеют одинаковый интерфейс – они публикуют созданную карту среды в топик */map*. По умолчанию, для робота Turtlebot 3 Burger используется алгоритм SLAM Gmapping [44].

Стоит отметить, что в реализациях алгоритмов SLAM для ROS есть одна особенность, которая не соответствует интуитивному пониманию принципов работы лазерного лидара. Лидар измеряет расстояния до препятствия с определенным шагом в пределах своих минимального и максимального углов зрения. Таким образом, набор этих расстояний представлен в виде массива, длина которого равна количеству шагов между минимальным и максимальным углами зрения. В тех случаях, когда расстояние до препятствия превышает радиус действия лидара, по умолчанию возвращается значение $+\infty$.

В алгоритмах SLAM для ROS есть встроенный фильтр, который отбрасывает показания лидара, которые больше его радиуса действия. Фактически, области карты помечаются на карте как свободные только тогда, когда они находятся между роботом и каким-либо препятствием. При этом, расстояние до этого препятствия должно быть меньше радиуса действия лидара робота. Если расстояние до препятствия будет больше радиуса действия лидара робота, то такие области останутся помеченными как неизвестные несмотря на то, что они свободны.

На практике это обозначает, что если робот, использующий такие алгоритмы, приедет на границу между исследованной и неисследованной областями, то не гарантируется, что робот сможет исследовать эту область. Из-за этого наблюдаются проблемы при исследовании больших открытых пространств, на которых отсутствуют препятствия. Эта проблема усугубляется для маломощных лидаров с малым радиусом действия, например, для лидара робота Turtlebot 3 Burger, радиус действия которого составляет 3 метра. Для роботов, оснащенных профессиональными лидарами, радиус действия которых составляет более 100 метров, данная проблема не имеет практического значения в типовых замкнутых помещениях, но может проявляться на открытых пространствах.

Данное ограничение на самом деле является вполне логичным, потому что существуют препятствия, которые могут поглощать лазерные лучи или отражать их в случайном направлении. Кроме того, у лидаров имеются большие проблемы с распознаванием стеклянных объектов.

Все три алгоритма исследования неизвестной среды, рассмотренные в моей дипломной работе, взаимодействуют с фреймворком ROS одинаковым образом:

1) Получают актуальную карту среды из топика */map*.

2) Принимают решение, какую область необходимо исследовать следующей.
3) При помощи стека навигации направляют робота в указанную область.

Для того, чтобы сравнение алгоритмов было объективным, все параметры стека навигации устанавливаются одинаковыми для всех трех алгоритмов. В частности, при тестировании всех трех алгоритмов используются одинаковые глобальный и локальный планировщики. В качестве глобального планировщика решено было использовать *NavFn* [45], в качестве локального – *DWA Local Planner* [46].

3.2 СИМУЛЯТОР GAZEBO

Использование робототехнических симуляторов на всех этапах разработки и внедрения робототехнических комплексов позволяет значительно сократить расходы и ускорить процессы. На самых ранних этапах моделирование робота в симуляторе позволяет быстро создать и протестировать прототип конструкции без создания дорогостоящего физического образца робота. На этапе разработки программного обеспечения для робота симуляторы способны предоставить безопасную и полностью контролируемую среду для испытаний и проверки гипотез [47].

В настоящей работе использование симулятора необходимо, чтобы ликвидировать влияние случайных факторов, таких как шум датчиков, а также обеспечить полностью детерминированную тестовую среду для повторяемости результатов экспериментов. Кроме того, использование симулятора позволяет создавать достаточно большие тестовые помещения с любым количеством и расположением препятствий, что в реальном мире не всегда возможно. Запуск экспериментов в симуляторе позволяет значительно сократить время на подготовку и выполнение эксперимента, что также очень важно, поскольку для программирования и отладки алгоритмов требуются большое количество тестовых запусков.

Для симуляции решено было использовать симулятор Gazebo [19]. В настоящее время Gazebo – это набор библиотек с открытым исходным кодом, созданных для того, чтобы упростить разработку высокопроизводительных приложений. Основной аудиторией Gazebo являются разработчики и инженеры-робототехники. Каждая библиотека Gazebo требует минимальных зависимостей, позволяя решать самые разные задачи: от решения математических трансформаций, до физической симуляции роботов.

Gazebo позволяет использовать для симуляции различные физические движки, такие как Open Dynamics Engine (ODE) [51], Bullet [52], Dynamic Animation and Robotics Toolkit (DART) [53] и другие. Для 3D-рендеринга используется движок с открытым исходным кодом OGRE [54], способный работать в реальном времени. В настоящее время Gazebo используется в качестве стандартной среды для симуляции (Рисунок 3.1) во многих крупных соревнованиях по робототехнике, в том числе проводимых агентствами DARPA [55] и NASA [56].

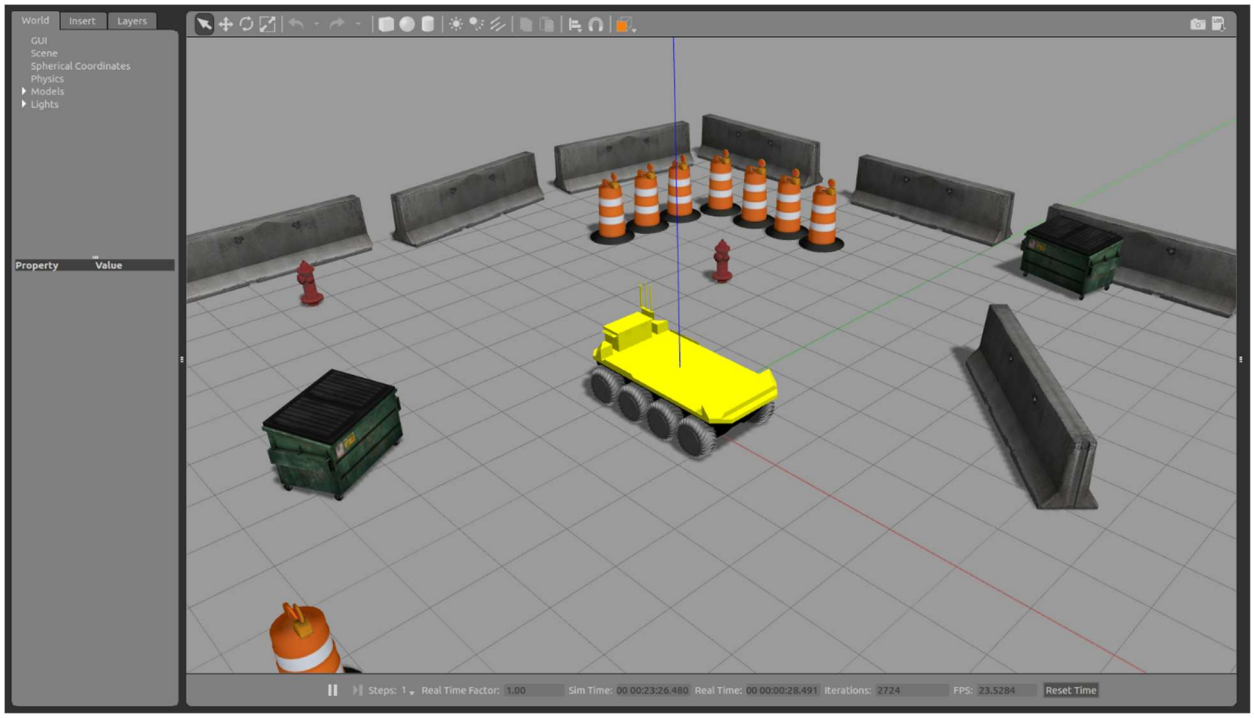

Рисунок 3.1. Основное окно робототехнического симулятора Gazebo[1].

[1] https://www.clearpathrobotics.com/assets/guides/kinetic/moose/_images/moose_simulator_gazebo.png

3.3 МОБИЛЬНЫЙ РОБОТ TURTLEBOT 3 BURGER

Все рассмотренные в настоящей работе алгоритмы исследования неизвестной среды требуют от аппаратной составляющей мобильного робота только наличия лидара. При этом для работы этих алгоритмов фактически безразлично каким конкретно механизмом передвижения обладает мобильный робот, потому что фреймворк ROS предоставляет единый интерфейс управления роботом с использованием пакета *move_base* [57]. Таким образом, при выборе робота, на котором будут тестироваться алгоритмы исследования неизвестной среды, в первую очередь следует учитывать не аппаратную, а программную составляющую, то есть полную поддержку фреймворка ROS и наличие хорошо работающей модели данного робота для симулятора Gazebo. После изучения наиболее популярных в научном сообществе мобильных роботов, было решено, что робот Turtlebot 3 Burger лучше всего подходит для данной задачи.

Turtlebot 3 Burger – это компактный, модульный, настраиваемый мобильный робот от компании ROBOTIS [58]. При его проектировании разработчики поставили себе целью создать компактный и доступный по цене мобильный робот, который будет обладать функционалом, схожим с функционалом существенно более дорогих профессиональных мобильных роботов. Данный робот имеет модульную структуру (Рисунок 3.2), которая позволяет существенно расширять его возможности путем установки дополнительных модулей. Также следует отметить, что робот поддерживает установку всех сенсоров и исполнительных механизмов, которые производятся компанией ROBOTIS.

Этот робот имеет достаточно простой механизм движения Differential Drive и отличается от других подобных роботов тем, что для него существует хорошая реализация стека навигации, которая поддерживается совместно силами разработчиков ROS и компании ROBOTIS. Кроме того, для него

существует плагин для Gazebo от компании-производителя, который позволяет использовать его в симуляции.

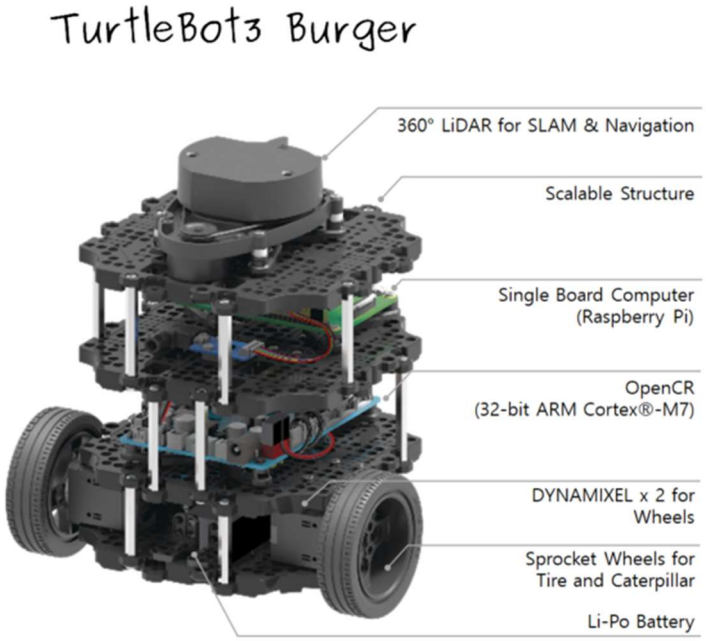

Рисунок 3.2. Описание элементов конструкции робота Turtlebot 3 Burger[2].

Робот Turtlebot 3 Burger имеет следующие отличительные особенности:

- **Полная поддержка фреймворка ROS.** Всё необходимое программное обеспечение, которое требуется для поддержки фреймворка ROS разрабатывается и поддерживается непосредственно производителем робота.
- **Доступная цена.** Turtlebot является доступной по цене платформой для обучения и научных исследований.
- **Компактность.** Робот достаточно легкий, чтобы его можно было переносить в рюкзаке (Рисунок 3.3).
- **Возможности для усовершенствования.** Для робота доступно множество дополнительных модулей.
- **Модульные сервоприводы.** Сервоприводы легко установить или снять для проведения сервисного обслуживания.

[2] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#features

- **Открытое программное обеспечение.** Весь исходный код программного обеспечения, необходимого для функционирования робота, находится в открытом доступе.
- **Открытое аппаратное обеспечение.** Дизайн печатных плат, список деталей и 3D модели компонентов находятся в открытом доступе.
- **Богатый набор сенсоров.** Робот оснащен 360° лазерным дальномером (LiDAR), высокоточными энкодерами и 9-осевым гиростабилизатором (IMU).

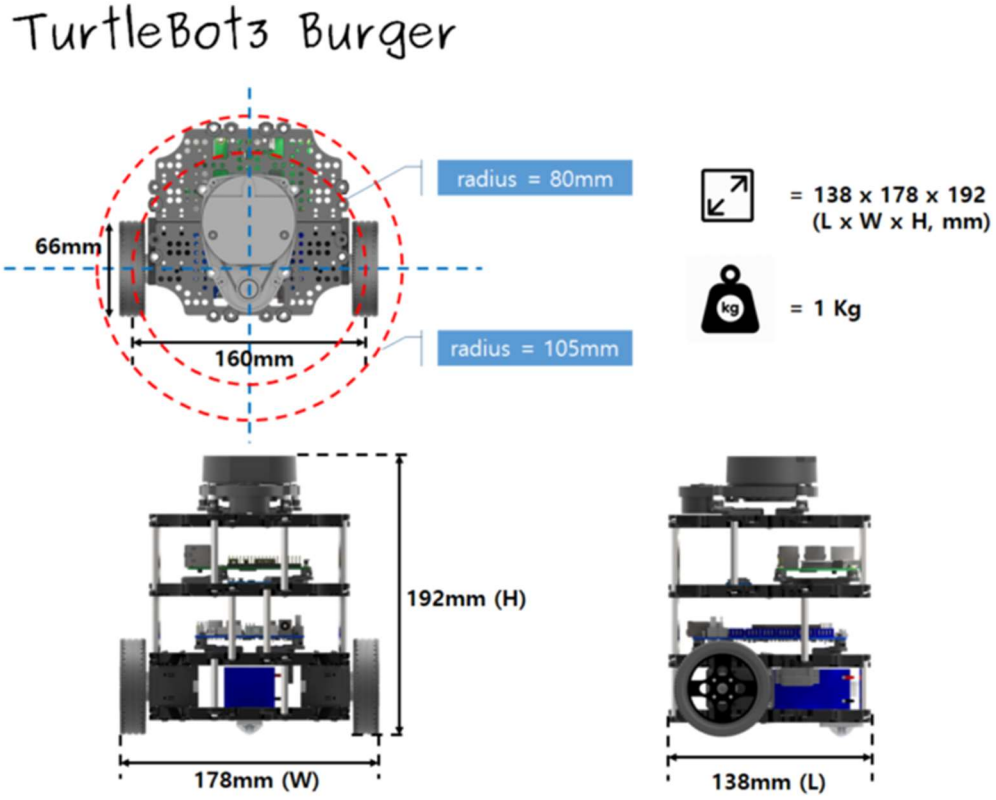

Рисунок 3.3. Физические размеры робота Turtlebot 3 Burger[3].

Основная идея при проектировании робота Turtlebot 3 Burger состояла в том, чтобы создать робота для обучения мобильной робототехнике, который был бы доступен для бюджета любой образовательной организации. Робот оборудован лидаром, что позволяет использовать его для задач картографирования, исследования неизвестного окружения и навигации.

[3] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#features

Благодаря вышеперечисленным факторам данный робот очень популярен в научном сообществе. Имеется большое количество научных работ, в которых используется этот робот. Характеристики робота Turtlebot 3 Burger приведены в таблице (Таблица 3.1).

Таблица 3.1. Характеристики робота Turtlebot 3 Burger.

Характеристика	**Значение**
Максимальная скорость прямолинейного движения	0,22 м/с
Максимальная скорость вращения на месте	1,82 рад/с (104,27 град/с)
Размеры (Д х Ш х В)	138 мм х 178 мм х 192 мм
Масса (в сборе)	1 кг
Максимальная высота преодолеваемого препятствия	10 мм
Расчетное время работы от батареи	2 ч 30 мин
Расчетное время заряда батареи	2 ч 30 мин
Ёмкость бортовой батареи	1800 мАч
Дальность действия лазерного дальномера	3 м

4 ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

4.1 АЛГОРИТМ FRONTIER EXPLORATION

Алгоритм Frontier Exploration был изобретен в 1997 году в Лаборатории военно-морских исследований ВМФ США [59]. Его автором является Brian Yamauchi. В его публикации [17] подробно описаны детали тестирования алгоритма на роботе Nomad 200 [60] в реальных помещениях.

В алгоритме используется термин *frontier* (англ. frontier – граница, рубеж; далее обозначается как «фронтир»), который обозначает границу между исследованной и неисследованной областями (Рисунок 4.1). Алгоритм Frontier exploration построен на гипотезе, что постоянное движение к ближайшей границе между исследованной и неисследованной областями – лучший способ исследования неизвестной среды.

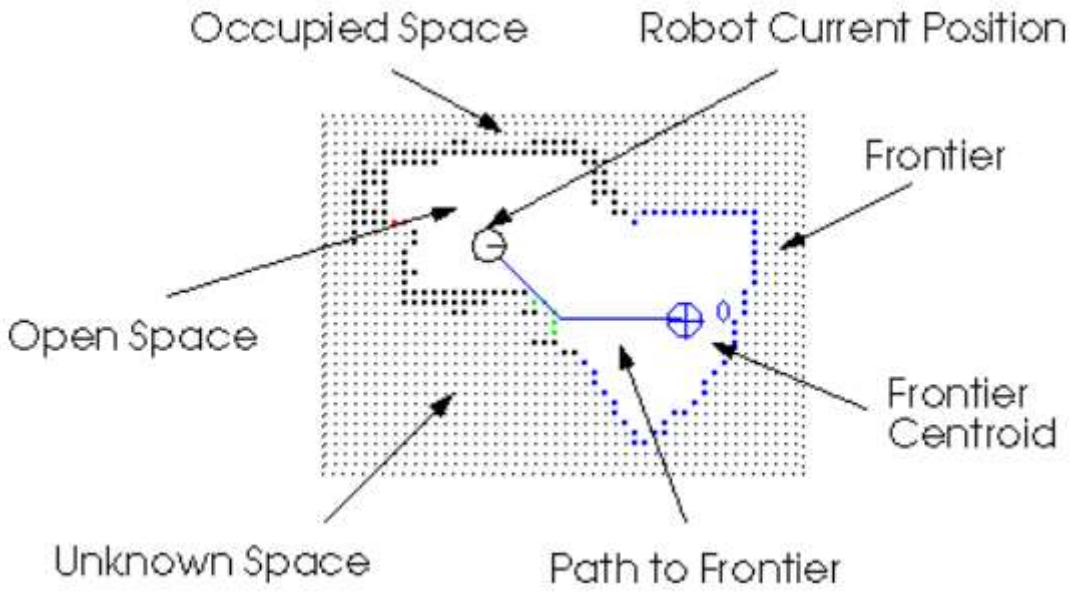

Рисунок 4.1. Визуальное описание терминов, используемых в алгоритме Frontier exploration [62].

Достигнув такой границы, робот получает возможность исследовать своими сенсорами те области, которые ранее были для него неизвестными. После исследования робот наносит эти области на свою карту, таким образом отодвигая границу между известными областями и неизвестными.

Постоянной двигаясь к этой границе, робот увеличивает размеры известной ему территории.

Если робот, имея полностью известную карту местности, может построить маршрут из своего текущего местоположения до определенной точки на карте, то такая точка считается *доступной*. Пространство доступных точек непрерывно, потому что по определению должен существовать маршрут из каждой доступной точки в любую другую доступную точку. Начальная позиция робота на карте также является доступной точкой. Построим маршрут из начальной позиции робота до любой из доступных точек, которая не лежит в пределах действия сенсоров робота. Начало построенного маршрута будет находиться на известной территории, а конец – на неизвестной. Из этого следует, что робот, двигаясь по этому маршруту, обязательно пересечет фронтир. Достигнув этого фронтира, робот нанесет на карту новые территории. После этого ситуация повторится: робот будет находиться на известной территории, а конец маршрута – на неизвестной, следовательно робот должен будет снова пересечь еще один фронтир. Таким образом, робот постепенно исследует всё пространство доступных точек, то есть составит точную карту среды за конечное количество времени.

Алгоритм Frontier Exploration использует occupancy grid map (OGM) [61] в качестве пространственного представления карты помещения в памяти робота. Карта помещения представлена в виде матрицы, в каждой ячейке которой хранится значение вероятности, что данная ячейка занята препятствием. Для уменьшения вычислительной сложности конкретное числовое значение вероятности обычно не вычисляется, и ячейки имеют только три состояния:

- Ячейка свободна.
- Ячейка занята.
- Состояние неизвестно.

Представление карты помещения в виде OGM является широко распространенным подходом под названием *дискретизация*, который применяется при цифровой обработке непрерывного сигнала. Суть дискретизации заключается в представлении непрерывной функции дискретной совокупностью её значений при разных наборах аргументов. В данном случае непрерывное трехмерное пространство реального мира разделяется на ячейки равного размера, для каждой из которых указывается наличие или отсутствие препятствий. В качестве аргументов выступают координаты x и y.

Задача поиска фронтиров сводится к задаче поиска в матрице ячеек, состояние которых известно, смежных с ячейками, состояние которых неизвестно. В оригинальной работе поиск таких ячеек был реализован при помощи алгоритма поиска в ширину (Breadth First Search (BFS) [63]), который в данном случае имеет временную сложность $O(rows * cols)$, где $rows$ – количество рядов, $cols$ – количество колонок в матрице.

В настоящее время существует множество других работ, авторы которых предлагают более продвинутые алгоритмы для поиска фронтиров, например, такие как Expanding-Wavefront Frontier Detection (EWFD) [64] или Frontier-Tracing Frontier Detection (FTFD) [18]).

Алгоритм Frontier exploration работает следующим образом:

1) Найти все фронтиры на глобальной карте.
2) Выбрать ближайший к роботу доступный фронтир.
3) Установить центроид (т. е. точку, каждая координата которой равна среднему значению соответствующей координаты у всех точек фронтира) выбранного фронтира в качестве цели.
4) Построить маршрут к цели и приступить к его выполнению.
5) Во время движения по маршруту добавлять новую информацию на карту.

6) После достижения цели перейти к шагу 1, пока местность не будет полностью исследована (то есть на карте не останется фронтиров).

Алгоритм Frontier exploration является одним из самых часто используемых алгоритмов для решения задачи исследования неизвестной среды. На его основе создано множество алгоритмов для исследования неизвестной среды как при помощи одного робота [65], так и при помощи команды из нескольких роботов [66].

Фреймворк ROS имеет несколько пакетов с различными реализациями алгоритма Frontier exploration. Пакет *frontier_exploration* [67] имеет хорошую документацию, используется во множестве различных работ, но, на данный момент, его разработка и поддержка приостановлены[4]. Релизы пакета *frontier_exploration* существуют только для устаревших версий ROS, уже достигших статуса EOL (End of life), например, таких как ROS Jade или ROS Kinetic.

Для актуальной версии ROS Noetic существует пакет *explore_lite* [68]. Именно он используется в данной дипломной работе в качестве реализации алгоритма Frontier exploration. В документации к пакету указано, что в нём используется такой же алгоритм поиска фронтиров, как в пакете *frontier_exploration*, поэтому с точки зрения вычислительной сложности нет разницы какой конкретно пакет используется.

4.2 ЖАДНЫЙ АЛГОРИТМ ИССЛЕДОВАНИЯ СРЕДЫ

Жадный алгоритм исследования неизвестной среды также использует концепцию фронтиров. При этом одно из основных отличий от алгоритма Frontier exploration состоит в том, каким образом жадный алгоритм выбирает наиболее подходящий фронтир.

Жадный алгоритм принимает в расчет область действия сенсоров, в частности радиус действия лазерного лидара, установленного на роботе. Эта

[4] Последнее обновление пакета *frontier_exploration* было 26.10.2016

информация позволяет алгоритму проанализировать сколько информации о неизвестной области робот способен собрать из какой-либо точке на карте.

Как и предыдущий алгоритм, жадный алгоритм представляет карту помещения в памяти в виде Occupancy Grid Map. Каждая ячейка OGM является представлением какой-либо области на местности в реальном мире, при этом размеры такой области строго фиксированы. Например, стандартные алгоритмы из состава ROS оперируют квадратными областями размером 5см x 5см на одну ячейку. Таким образом, зная радиус действия лазерного лидара, который, как правило, указывается производителем в характеристиках, можно подсчитать, сколько ячеек OGM способен исследовать робот, который находится в какой-либо конкретной ячейке (Рисунок 4.2).

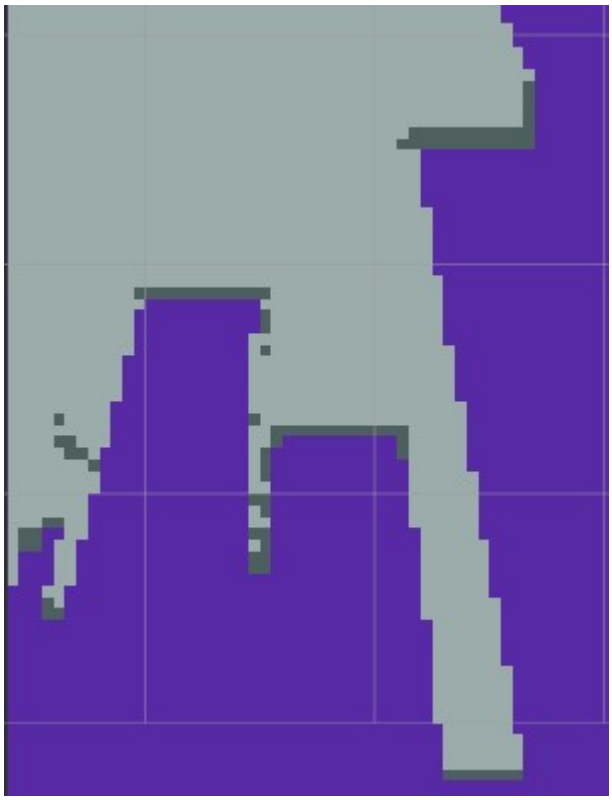

Рисунок 4.2. Участок глобальной карты в виде OGM. Серым выделены свободные области, темно-серым – препятствия, фиолетовым – неизвестные области.

Сравнивая по этому критерию ячейки, которые находятся на рубежной области, между собой, можно выбрать наиболее «полезную» ячейку, которую будет рациональнее всего указать роботу в качестве цели.

При этом необходимо отметить, что некоторые неизвестные области могут находиться за препятствиями, например за стенами. В этом случае лазерный лидар не даст никакой информации о том, что находится в этих областях. Таким образом, что простой подсчет количества неизвестных ячеек, которые находятся внутри круга радиусом в область действия лидара, даст совершенно неправильные результаты в оценке «полезности» той или иной ячейки. Если заранее известно, что в области видимости лидара находятся препятствия, то следует установить, насколько они перекрывают область видимости лидара.

Авторы жадного алгоритма используют для решения этой проблемы алгоритм Брезенхэма [69], который позволяет определить в какие ячейки двумерной матрицы попадает прямая линия между двумя заданными точками. Проверив наличие препятствий в каждой из этих ячеек, можно определить находятся ли две точки в прямой видимости друг от друга.

Кроме того, жадный алгоритм отличается от алгоритма Frontier exploration тем, что принимает во внимание угол зрения и радиус действия лидара, установленного на роботе, а также размеры самого робота. На основе этих данных алгоритм строит конфигурационное пространство, которое используется для того, чтобы обеспечить безопасное движение робота на местности.

Жадный алгоритм исследования неизвестной среды работает следующим образом:

1) Найти все фронтиры на глобальной карте.
2) Полным перебором всех точек всех фронтиров найти точку на любом из фронтиров, из которой можно будет исследовать

наибольшее количество неизвестных ячеек OGM в зоне действия лидара робота. Выбрать эту точку кандидатом на цель.

3) Если кандидат на цель находится в конфигурационном пространстве, выбрать её в качестве цели.
4) В противном случае выбрать ближайшую точку из конфигурационного пространства, которая будет в прямой видимости от точки-кандидата на цель. Выбрать её в качестве цели.
5) Построить маршрут к цели и приступить к его выполнению.
6) Во время движения по маршруту добавлять новую информацию на карту.
7) После достижения цели перейти к шагу 1, пока местность не будет полностью исследована (то есть на карте не останется фронтиров).

Использование конфигурационного пространства позволяет избежать ситуаций, когда цель расположена слишком близко к препятствию, которое не дает роботу достигнуть цели из-за размеров робота.

Жадный алгоритм и алгоритм Frontiers exploration имеют некоторые сходства по своей структуре. Фактически, оба алгоритма можно считать жадными, поскольку они на каждом шаге пытаются принять локально оптимальное решение. При этом, у них совершенно разные критерии оценки оптимальности решений. Алгоритм Frontiers exploration всегда выбирает ближайший фронтир, пытаясь таким образом минимизировать расстояние, которое должен проехать робот. Для жадного алгоритма исследования неизвестной среды расстояние до выбранного фронтира значения не имеет, он пытается на каждом шаге выбрать фронтир, который содержит наибольшее количество полезной информации.

Жадный алгоритм будет проигрывать алгоритму Frontier exploration по метрике общего пройденного расстояния хотя бы потому, что не ставит себе

целью оптимизировать пройденное расстояние. По этой причине жадный алгоритм представляет интерес только с исследовательской точки зрения. Для практического использования он интереса не представляет, поэтому среди общедоступных пакетов для фреймворка ROS отсутствует. Для тестирования используется реализация жадного алгоритма из пакета *unknown_exploration* [28], который был разработан в нашей лаборатории.

4.3 АЛОГРИТМ ERGODIC ENVIRONMENTAL EXPLORATION (E^3)

Алгоритм Ergodic Environmental Exploration (E^3) является частью докторской диссертации [25] «A Control Theoretic Perspective On Learning In Robotics», автором которой является Rowland O'Flaherty. Тема оптимального исследования среды является лишь частью диссертации, которой посвящено несколько глав. Основной темой является автономность робототехнических систем, которая подразумевает самостоятельное принятие решений.

В задаче автономного исследования неизвестной среды роботу приходится принимать множество самостоятельных решений. Эти решения подразумевают как быстрое решение множества низкоуровневых задач (например, движение по заданной траектории), так и медленное решение нескольких высокоуровневых задач (например, планирование пути на карте). При этом, зачастую, граница между этими классами задач является достаточно размытой.

Алгоритм E^3 способен работать в многомерном пространстве. Например, его можно использовать для исследования трехмерного пространства при помощи беспилотного летательного аппарата. Авторы протестировали алгоритм E^3 на наземном роботе для исследования двумерного пространства и на квадрокоптере для исследования трехмерного пространства. В данной главе моей работы описывается только вариант алгоритма для двумерного пространства, в связи с чем теория подается в несколько упрощенном виде.

Значительным отличием алгоритма E^3 от двух предыдущих алгоритмов является то, что он не использует концепцию фронтиров. Вместо этого он вводит функцию для оценки количества информации в каждой точке исследуемой среды. На основе оценки количества информации в точке x исследуемой среды и времени её последнего сканирования сенсорами робота вычисляется значение функции $H(x)$. Функция $H(x)$ позволяет численно выразить важность той или иной области для исследования, и отдавать предпочтение более важным областям в процессе исследования. Процессы оценки количества информации в каждой точке среды, актуализации этой оценки с течением времени и вычисления функции $H(x)$ подробно описаны в главе 4.3.1. Для визуального восприятия этой оценки далее приведен рисунок (Рисунок 4.3).

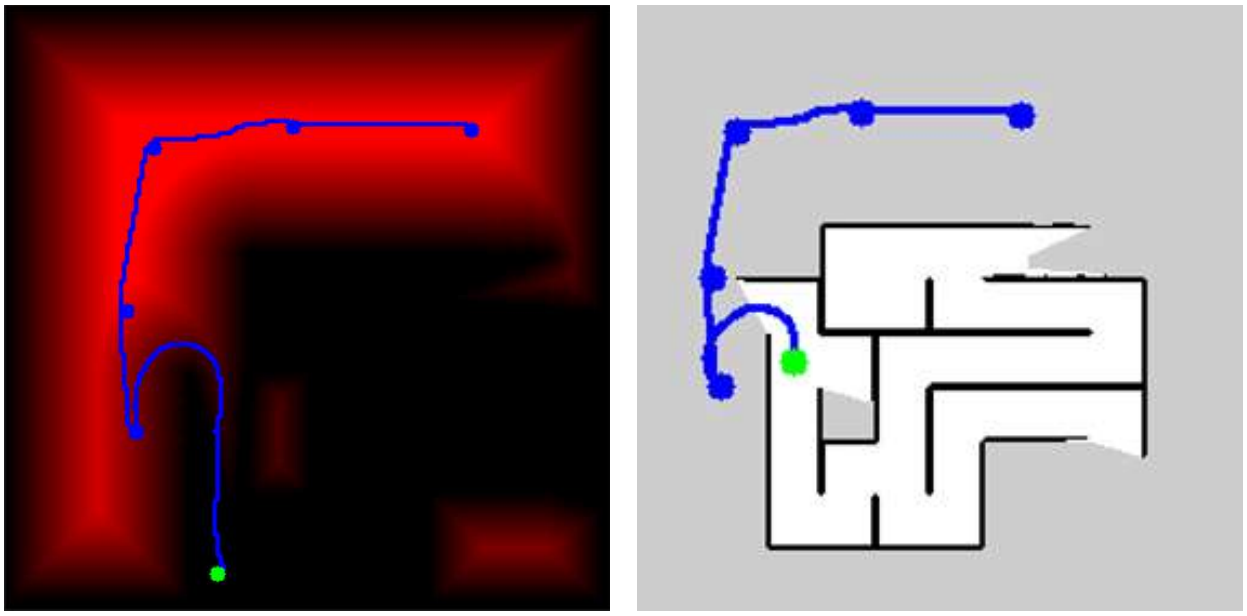

Рисунок 4.3. Слева – оценка количества информации в виде градиента от черного (мин.) до красного (макс.); справа – глобальная карта среды. Синяя линия – построенный маршрут для исследования, зеленая точка – робот.

На основе значений функции $H(x)$, вычисленных для всех точек исследуемой среды составляется двумерная карта данной среды. С использованием этой карты планируется кратчайший маршрут, который проходит по наиболее полезным для исследования областям среды.

Составление двумерной карты среды и процедура планирования маршрута описаны в главе 4.3.2.

Составленный маршрут представляет собой ломанную линию, которая проходит по нескольким точкам среды, которые имеют наибольшую оценку количества информации. Авторы алгоритма E^3 вводят две метрики – количество затраченных усилий и эргодичность маршрута. Описание этих метрик представлено в главе 4.3.3. Далее, выполняется процедура оптимизации маршрута по двум этим метрикам при помощи линейно-квадратичного регулятора. Процедура оптимизации также описана в главе 4.3.3. Целью процедуры оптимизации является нахождение разумного компромисса, который позволит минимизировать значения обеих метрик. После того, как будет найден оптимальный маршрут, робот выполняет его, попутно исследуя окружающие территории при помощи своих сенсоров. После достижения последней точки маршрута, вышеописанная последовательность действий повторяется.

Алгоритм E^3 работает следующим образом:

1) Вычислить оценку важности каждой точки для исследования $H(x)$, основанную на оценке количества информации в этой точке и времени её последнего сканирования сенсорами робота.
2) Найти l точек, которые имеют самые высокие значение $H(x)$.
3) Построить кратчайший маршрут, проходящий по всем l точкам, с началом в текущей позиции робота.
4) Оптимизировать этот маршрут при помощи линейно-квадратичного регулятора с целью минимизации метрики эргодичности маршрута и метрики затраченных усилий.
5) Приступить к выполнению полученного маршрута.
6) Во время движения по маршруту добавлять новую информацию на карту.

7) После достижения цели перейти к шагу 1, пока алгоритм не будет завершен.

Алгоритм E^3 основан на алгоритме Ergodic Exploration for Distributed Information (EEDI) [70], который был разработан в Лаборатории нейронауки и робототехники (NxR Lab [71]) университета Northwestern [72]. Научная новизна алгоритма E^3 заключается во введении функции оценки количества информации, которая позволяет хранить эту оценку для каждой области среды в виде OGM.

Алгоритм EEDI предполагает, что расположение информации в неизвестной среде может быть описано при помощи функции распределения, параметризованной m неизвестными параметрами. Для нахождения точных значений этих неизвестных параметров требуется решить матрицу Фишера размера $m \times m$. Использование алгоритма EEDI в двумерном пространстве, представленном в виде OGM размера $c \times c$, потребует использования $m = c^2$ неизвестных параметров и решения матрицы Фишера, в которой будет c^4 элементов. Таким образом, количество элементов в матрице Фишера растёт очень быстро, что делает использование алгоритма EEDI для исследования больших территорий практически неосуществимым.

Авторы алгоритма E^3 использовали для экспериментов площадку размерами $5\text{м} \times 3{,}75\text{м}$ ($18{,}75\text{м}^2$), представленную в виде OGM размерами 36×36. Даже использование такой сравнительно небольшой площадки требует решения матрицы Фишера из более, чем 1,68 миллионов элементов, для чего необходимо большое количество вычислительных ресурсов. Для сравнения, в ROS по умолчанию используется OGM размерами 384×384.

По причинам, указанным выше, авторы алгоритма E^3 в своей работе проводили сравнение следующего набора алгоритмов:

- Алгоритм E^3.

- Алгоритм E^3 без оптимизации маршрута по эргодичности и затраченным усилиям.
- Алгоритм, генерирующий маршруты случайным образом.

Ожидаемо, что алгоритм E^3 оказался лучшим по итогам сравнения. Версия алгоритма E^3 без оптимизации маршрута по эргодичности и затраченным ресурсам показала результат примерно на 10% хуже.

Следует обратить внимание, что алгоритм E^3 помимо своих очевидных достоинств, имеет несколько серьезных недостатков. Во-первых, алгоритм E^3 подходит только для исследования неизвестных сред фиксированного размера. При этом размер исследуемой среды необходимо указать перед началом работы алгоритма.

Во-вторых, он не учитывает, что в исследуемой среде могут быть препятствия, и авторы тестировали алгоритм только в среде без препятствий. Среды, в которых полностью отсутствуют препятствия, встречаются в реальных задачах робототехники очень редко, что сильно ограничивает возможности практического использования алгоритма E^3.

Для того, чтобы алгоритм E^3 мог работать в среде с препятствиями необходимы серьезные доработки. Многие ключевые части алгоритма, такие как оценка количество информации или оптимизация маршрута по метрике эргодичности, должны быть значительно переработаны. В данной работе нами предлагаются несколько модификаций и улучшений алгоритма E^3, которые позволяют ему успешно работать в среде с препятствиями.

4.3.1 ОЦЕНКА КОЛИЧЕСТВА ИНФОРМАЦИИ

Обозначим множество всех точек в исследуемой среде как X. Далее обозначим функцию распределения количества информации в точке $x \in X$ в момент времени t как $\psi_t(x)$. Мобильный робот способен перемещаться в пространстве X и исследовать неизвестные области в этом пространстве при помощи своих сенсоров. В каждый момент времени робот способен

исследовать только то множество точек из пространства *X*, которое находится в пределах области действия его сенсоров:

$$D(x,\delta) = \{x' \quad : \quad \|x' - x\| \leq \delta\} \tag{4.3.1.1}$$

Иными словами, робот, находясь в точке x может сканировать своими сенсорами любые точки x' пространства *X*, расстояние до которых не превышает радиуса действия сенсоров δ. Робот составляет свою оценку распределения количества информации $\widehat{\psi_t}$, которую затем актуализирует истинными значениями ψ_t при помощи своих сенсоров.

Формула (4.3.1.1) не учитывает наличие препятствий в исследуемой среде, которые могут перекрывать поле видимости сенсоров. Следует отметить, что это далеко не единственная проблема, которая возникает у алгоритма E^3 при работе в среде с препятствиями. Причина этих проблем состоит в том, что авторы разрабатывали алгоритм E^3 для исследования сред без препятствий.

При исследовании неизвестной среды робот имеет возможность запоминать координаты точек, в которых он уже был. Эта возможность обозначается индикаторной функцией $I_t(x) : X \to \{0,1\}$, которая возвращает 0 если робот никогда не был в точке x до момента времени t. В противном случае она возвращает 1. Когда робот передвигается, он обновляет функцию $I_t(x)$ согласно следующему правилу:

$$I_{t+\Delta t}(x') = \begin{cases} 1 & : \quad x' \in D(x,\delta) \\ I_t(x') & : \quad x' \notin D(x,\delta) \end{cases} \tag{4.3.1.2}$$

На каждом шаге алгоритма роботу необходимо актуализировать свою оценку $\widehat{\psi_t}$, которая меняется с течением времени. Актуализация оценки производится при помощи двухшаговой функции:

$$\begin{aligned} \text{Шаг 1:} \quad & \widehat{\psi^{\dagger}}(x') = \begin{cases} \psi_t(x') & : \quad x' \in D(x,\delta) \\ \widehat{\psi_t}(x') & : \quad x' \notin D(x,\delta) \end{cases} \\ \text{Шаг 2:} \quad & \widehat{\psi_{t+\Delta t}}(x') = \begin{cases} \widehat{\psi^{\dagger}}(x') & : \quad x' \in D(x,\delta) \\ \Upsilon\left(x', \widehat{\psi^{\dagger}}, I_{t+\Delta t}\right) & : \quad x' \notin D(x,\delta) \end{cases} \end{aligned} \tag{4.3.1.3}$$

Процесс актуализации оценки количества информации для произвольной точки x' можно кратко описать следующим образом:

- Если точка x' находится в пределах области видимости сенсоров робота, то оценка не изменяется.
- В противном случае оценка вычисляется при помощи функции $\Upsilon(x, \hat{\psi}, I)$.

Начальное значение функции оценки количества информации при $t = 0$:

$$\widehat{\psi_0}(x') = \frac{1}{|X|}, \qquad \forall x' \in X \tag{4.3.1.4}$$

Фактически, распределение количества информации в точках $x \in X$ изначально оценивается как равномерное. Таким образом, все области неизвестной среды изначально оцениваются как одинаково важные для исследования.

Функция $\Upsilon(x, \hat{\psi}, I)$ вычисляется следующим образом:

$$\Upsilon(x, \hat{\psi}, I) = \frac{\Gamma(\hat{\psi}, I)\, L(x, I)}{\int_X L(x, I)\, dx} \tag{4.3.1.5}$$

При вычислении функции $\Upsilon(x, \hat{\psi}, I)$ используются функции $\Gamma(\hat{\psi}, I)$ и $L(x, I)$.

Функция $\Gamma(\hat{\psi}, I)$ вычисляет общее количество информации во всей неизвестной среде, которую роботу ещё предстоит собрать:

$$\Gamma(\hat{\psi}, I) = 1 - \int_X \hat{\psi}(x)\, I(x)\, dx \tag{4.3.1.6}$$

При определении этой функции авторы алгоритма E^3 исходили из предположения, что общее количество информации во всей неизвестной среде, перед началом её исследования, равно 1. Таким образом, вычитая из единицы сумму уже собранной на данный момент информации, можно вычислить, сколько информации ещё осталось собрать.

Авторы признают, что предположение об общем количестве информации во всей среде, равном 1, является очень серьезным допущением. Оно удобно тем, что позволяет оценивать количество информации в областях, где робот никогда не был, но способно создать серьёзные проблемы для алгоритма в тех случаях, когда оно оказывается неверным.

Например, если общее количество информации в среде недооценено, робот может застрять в одной области среды, будучи уверенным, что собрал всю информацию во всей среде. Если же общее количество информации переоценено, то робот будет тратить слишком много усилий на дальнейшее исследование уже достаточно исследованных областей.

Функция $L(x, I)$, которая также используется при вычислении функции $\Upsilon(x, \hat{\psi}, I)$, представляет собой метрику геодезического расстояния, которое вычисляется до точки x от границы уже исследованной области или от границы исследуемой среды, в зависимости от того, что окажется ближе.

Вычисление геодезического расстояния для каждой точки $x \in X$ по отдельности нецелесообразно с точки зрения расхода вычислительных ресурсов, потому что содержит большое количество повторяющихся вычислений. Авторы алгоритма E^3 предлагают представлять множество значений функции $I(x)$ для всех $x \in X$ в виде двоичной матрицы. Тогда значение функции $L(x, I)$ для всех точек исследуемой среды (Рисунок 4.4) можно быстро вычислить при помощи алгоритма Fast Marching Method [77], асимптотическая сложность которого составляет $O(n \log n)$.

Большим преимуществом алгоритма E^3 перед жадным алгоритмом и алгоритмом Frontier exploration является то, что он способен работать в *динамических* средах, то есть в средах, которые могут произвольно изменяться без влияния исследующего их агента. Любые среды, в которых присутствуют люди или другие подвижные роботы, являются динамическими, поскольку расположение объектов может изменяться произвольным образом.

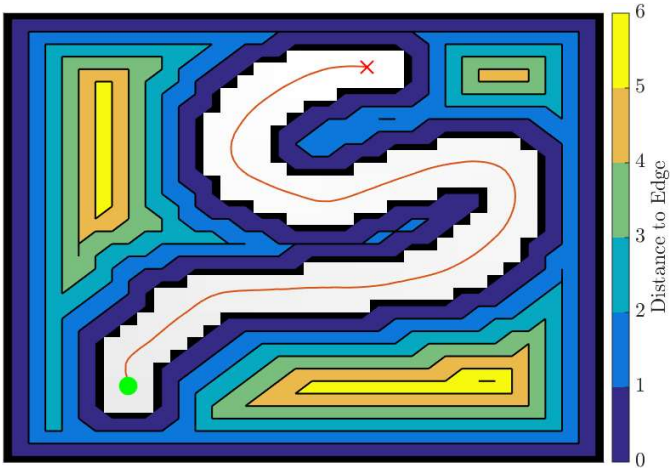

Рисунок 4.4. Пример расчета функции $L(x, I)$. Зеленая точка – робот, красная линия – его траектория движения [25].

Жадный алгоритм и алгоритм Frontier exploration не предполагают, что конфигурация препятствий в среде может измениться спустя какое-то время, поэтому они не исследуют заново территории, которые уже были ими исследованы. Заметить, что некоторые препятствия изменили свое расположение они способны только случайно – например, проезжая через уже исследованную область на пути к новому фронтиру. Такие алгоритмы исследования среды не подходят для некоторых специфических задач, например таких, как патрулирование территории.

Исследование динамической среды намного сложнее, чем исследование статической среды. При исследовании динамической среды возникает следующая проблема: необходимо решить, когда следует перейти от сбора информации к использованию этой информации для выполнения задачи. То есть на примере с патрулированием территории города эта проблема выглядит следующим образом: необходимо решить, когда нужно исследовать новые районы в городе, и когда нужно заниматься патрулированием уже исследованных районов.

Этот компромисс между исследованием и применением имеющихся знаний (*exploration vs. exploitation*) подробно описан в работах по обучению с подкреплением. Авторы алгоритма E^3 утверждают, что при его использовании

не требуется явно решать, когда от исследования переходить к применению имеющихся знаний. Алгоритм E^3 находит компромисс, исследуя и применяя новую информацию одновременно.

Существует множество методов для нахождения компромисса между исследованием и применением имеющихся знаний. Они делятся на две основные категории:

- Ненаправленное исследование
- Направленное исследование

Ненаправленное исследование не требует каких-либо специальных знаний о предмете исследования. В такой стратегии исследование нового происходит путем внесения элемента случайности в процесс принятия решений. Примером такой стратегии может служить стратегия случайного движения по местности.

Направленное исследование использует знания о предмете исследования и пытается максимизировать объём получаемых знаний. Примером может служить стратегия исследования путем подсчета состояний, которая подсчитывает как часто система переходит в какое-либо конкретное состояние в зависимости от принятых ранее решений.

Алгоритм E^3 использует стратегию исследования, основанную на подсчете с функцией распада, при которой подсчитанное число переходов в какое-либо состояние экспоненциально убывает с течением времени.

Функция распада в алгоритме представлена функцией «уверенности» робота в оценке количества информации в какой-либо точке среды. Когда робот измеряет количество информации в какой-либо точке среды при помощи своих сенсоров, его оценка полностью соответствует реальности – робот имеет высокую уверенность в своей оценке. Далее, когда данная точка выходит из области видимости сенсоров робота, уверенность начинает экспоненциально снижаться с течением времени, мотивируя робота снова посетить данную область через некоторый период времени.

Функция уверенности $G_t(x)$ для произвольной точки $x' \in X$ в период времени t:

$$G_{t+\Delta t}(x') = \begin{cases} 1 & : \quad x' \in D(x, \delta) \\ \left(1 - \frac{\Delta t}{\tau}\right) G_t(x') & : \quad x' \notin D(x, \delta) \end{cases} \tag{4.3.1.7}$$

В начальный момент времени роботу об исследуемой среде не известно ничего, поэтому метрика уверенности равна нулю:

$$G_0(x') = 0, \qquad \forall x' \in X \tag{4.3.1.8}$$

Финальная функция, которая численно выражает важность точки $x \in X$ для исследования, выглядит следующим образом:

$$H(x) = \big(1 - G(x)\big)\, \psi(x) \tag{4.3.1.9}$$

Функция $H(x)$ активно используется в следующем этапе работы алгоритма E^3 – процессе планирования маршрута.

4.3.2 ПРОЦЕСС ПЛАНИРОВАНИЯ МАРШРУТА

На каждом шаге алгоритм E^3 вычисляет значение функции $H(x)$ для каждой точки $x \in X$, предоставляя полученные значения в виде двумерной матрицы – карты среды. Далее в этой двумерной матрице требуется найти l наибольших значений. Авторы алгоритма E^3 поступают следующим образом:

1) Конвертируют эту матрицу в изображение, закодированное в цветовом режиме градаций серого.
2) При помощи алгоритма Watershed [74] находят на этом изображении l вершин.

После того, как l точек с максимальными значениями функции $H(x)$ будут найдены, необходимо построить кратчайший маршрут, проходящий через все эти точки. Началом этого маршрута должно быть текущее расположение робота. Задача построения кратчайшего маршрута, проходящего через все точки, известна как «задача коммивояжера» [75].

Задача коммивояжера относится к классу NP-трудных задач и является трудновычислимой.

Асимптотическая сложность нахождения точного решения путем полного перебора составляет $O(n!)$, где n – количество точек, которые требуется посетить. Существует также и алгоритм решения задачи коммивояжера при помощи динамического программирования, который имеет сложность $O(n^2\,2^n)$. С ростом n вычислительная сложность растет очень быстро, делая решение задачи коммивояжера практически неосуществимым уже при $n = 20$. Тем не менее, для малых n, решить задачу коммивояжера даже при помощи полного перебора можно достаточно быстро. Алгоритм E^3 должен выполняться в реальном времени, поэтому авторы используют $l = 10$, которое обеспечивает разумный компромисс между скоростью вычисления и точностью построенного маршрута для исследования.

Созданный таким образом маршрут не является оптимальным, однако он является хорошим начальным значением для дальнейшей оптимизации по эргодичности и затраченным усилиям при помощи линейно-квадратичного регулятора. После множества экспериментов авторы алгоритма E^3 пришли к выводу, что такой метод планирования маршрута дает стабильно хороший результат.

4.3.3 ОПТИМИЗИЦИЯ МАРШРУТА ПО ЭРГОДИЧНОСТИ И ЗАТРАЧЕННЫМ УСИЛИЯМ

Маршрут, построенный при помощи решения задачи коммивояжера, не является оптимальным, поэтому его необходимо оптимизировать. Авторы алгоритма E^3 предлагают два критерия для оптимизации: *эргодичность* маршрута и усилия, затраченные на его выполнение.

Метрика эргодичности траектории показывает, насколько хорошо проложенный маршрут покрывает область интереса. Предположим, что

множество значений из распределения $\psi(x)$ собрано в области X сенсорами робота, который двигался по траектории $x(t)$ в течение интервала времени $T = [t_0, t_f]$. Траектория $x(t)$ является эргодичной к распределению $\psi(x)$, если процент времени, проведенный в любом подмножестве $A \subset X$ от $t = t_0$ до $t = t_f$ равен мере данного подмножества $A \subset X$.

Для получения оптимального маршрута требуется минимизировать метрику эргодичности. Однако при минимизации метрики эргодичности маршрута увеличивается метрика затрачиваемых усилий, потому что увеличивается длина маршрута. Для того, чтобы получить маршрут, который будет оптимальным по обеим метриками, необходимо использовать линейно-квадратичный регулятор. При этом необходимо предварительно определить весовые коэффициенты значимости каждой метрики, которые будут указывать, какой метрике отдавать предпочтение при оптимизации.

Для вычисления метрики эргодичности траектории требуется вычислить коэффициенты Фурье для траектории движения робота (4.3.3.1) и для пространственного распределения, заданного функцией $H(x)$ (4.3.3.2).

$$F_k\big(x(T)\big) = \frac{1}{|T|}\int_T \overline{f_k\big(x(t)\big)}\, dt \qquad (4.3.3.1)$$

$$\Phi_k = \int_X \overline{f_k(x)}\, H(x)\, dx \qquad (4.3.3.2)$$

В приведенных здесь формулах $x(t)$ обозначает позицию робота в момент времени t, то есть $x(t) \in X$. Таким образом, $x(T)$ обозначает траекторию движения робота за весь период времени $T = [t_0, t_f]$.

Обозначение $\overline{f_k(x)}$ представляет собой функцию, сопряженную к функции $f_k(x)$. Значения функции $\overline{f_k(x)}$ являются комплексно-сопряженными к значениям функции $f_k(x)$ при равных значениях аргумента. Сама функция $f_k(x)$ является k-ой базисной функцией Фурье:

$$f_k(x) = e^{2\pi j \xi_k^{\mathrm{T}} x} \qquad (4.3.3.3)$$

В уравнении выше $\xi_k \in \mathbb{R}^2$ обозначает k-ую пространственную частоту, также известную под названием *волновое число*.

При вычислении метрики эргодичности маршрута более важными являются более низкие пространственные частоты, поэтому используется коэффициент, который зависит от k:

$$\Lambda_k = \frac{1}{(1 + k^{\mathrm{T}}k)^{\frac{(n+1)}{2}}} \tag{4.3.3.4}$$

Таким образом, метрика эргодичности маршрута вычисляется по следующей формуле:

$$J_{ergo}\big(x(T)\big) = \frac{1}{2} Q_e \sum_{k \in K} \Lambda_k \big\| F_k\big(x(T)\big) - \Phi_k \big\| \tag{4.3.3.5}$$

В формуле (4.3.3.5) $K = \{0,1,\dots,N_1 - 1\} \times \{0,1,\dots,N_2 - 1\}$ – это множество индексов коэффициентов Фурье, имеющее $N_1 \in \mathbb{N}$, $N_2 \in \mathbb{N}$ индексов в каждом измерении пространства. Матрица $Q_e \geq 0$ используется в качестве весового коэффициента данной метрики в процессе оптимизации линейно-квадратичным регулятором.

Метрика затраченных усилий (англ. effort) вычисляется следующим образом:

$$J_{effort}\big(u(t)\big) = \frac{1}{2|T|} \int_T u(t)^{\mathrm{T}} R_e \, u(t) \, dt \tag{4.3.3.6}$$

В данной формуле $u(t)$ является маршрутом, полученным путем решения задачи коммивояжера, а $R_e = R_e^{\mathrm{T}}$ – весовым коэффициентом данной метрики в процессе оптимизации.

Таким образом, функция потерь, по которой будет производиться оптимизация, выглядит следующим образом:

$$J\big(x(t), u(t)\big) = J_{ergo}\big(x(t)\big) + J_{effort}\big(u(t)\big) \tag{4.3.3.7}$$

Процедура оптимизации предполагает нахождение таких $x^*(t)$ и $u^*(t)$, при которых функция $J\big(x^*(t), u^*(t)\big)$ будет принимать минимально возможное значение:

$$\big(x^*(t), u^*(t)\big) = \arg\min_{x(t),u(t)} J\big(x(t), u(t)\big) \qquad (4.3.3.8)$$

Важно иметь в виду, что оптимизация линейно-квадратичным регулятором предполагает нахождение локального оптимального значения. При этом не гарантируется, что оно будет и глобально оптимальным. Более того, важно и то, какие используются начальные значения $x(t)$ и $u(t)$. Использование одних и тех же начальных маршрутов для оптимизации приводит получению одинаковых локально оптимальных маршрутов, что сводит на нет исследование новых территорий.

Авторы алгоритма E^3 смогли решить эту проблему использованием процедуры, описанной в главе 4.3.2. Таким способом генерируются маршруты, которые проходят по разным областям исследуемой среды, что обеспечивает оптимальное покрытие всей территории среды.

5 МОДИФИКАЦИИ И УЛУЧШЕНИЯ АЛГОРИМА E^3

Авторы алгоритма E^3 не выкладывали исходный код алгоритма в открытый доступ, поэтому было решено реализовать этот алгоритм самостоятельно, используя при этом фреймворк ROS. В процессе реализации алгоритма пришлось решить множество проблем, которые не были освещены в диссертации [25]. В данной главе содержатся описания этих проблем, а также описания наших решений, которые позволяют полностью ликвидировать или, хотя бы, снизить влияние этих проблем на результаты работы алгоритма.

При реализации алгоритма E^3 на реальном роботе его авторы не использовали фреймворк ROS, потому что у них не было необходимости в использовании стандартных алгоритмов из робототехники. Например, таких как SLAM. В процессе исследования среды локализация робота осуществлялась при помощи камеры, установленной на потолке над исследуемой территорией. Картографирование не производилось, потому что в исследуемой среде полностью отсутствовали препятствия. По этой же причине не требовались также глобальный и локальный планировщики. Траектория движения робота полностью контролировалась алгоритмом E^3.

В реальной жизни, как правило, выполнение задачи исследования неизвестной среды подразумевает картографирование данной среды, то есть результатом выполнения данной задачи является карта территорий данной среды. Задача патрулирования территории, которую также способен выполнить алгоритм E^3, в этом плане отличается, поскольку её результатом должно быть обнаружение объектов, которые до этого отсутствовали. Тем не менее, для выполнения этой задачи всё равно требуется карта территорий, потому что нужно запоминать предыдущее расположение объектов.

Таким образом, можно прийти к выводу, что для при выполнении задачи исследования неизвестной территории необходимо использовать алгоритмы

SLAM, следовательно, использование фреймворка ROS для реализации алгоритма E^3 является вполне оправданным.

С учетом того, что жадный алгоритм и алгоритм Frontier exploration не предназначены для работы в динамических средах, проводить сравнение всех трех алгоритмов в динамической среде не имеет большого смысла. В таком сравнении алгоритм E^3 имеет значительное преимущество, потому что два остальных алгоритма способны лишь случайно обнаружить, что расположение некоторых препятствий изменилось.

С другой стороны, в данной работе основной целью ставится сравнение алгоритмов по энергоэффективности, то есть требуется узнать, насколько эффективно все три алгоритма способны исследовать одинаковую территорию. Для такого сравнения использование динамической среды также неверно, поскольку в этом случае не будет четкого критерия остановки исследования – невозможно будет понять, когда среда полностью исследована, если она постоянно меняется.

При работе в полностью статической среде использование метрики уверенности $G_t(x)$ будет являться недостатком для алгоритма E^3, потому что она будет мотивировать робота снова посещать уже исследованные области, которые по определению статической среды меняться не могут. Это будет приводить лишь к увеличению пройденного пути, не давая никаких преимуществ.

Для того, чтобы сравнение трех алгоритмов было объективным, было принято решение исключить функцию $G_t(x)$ из уравнения (4.3.1.9). Таким образом, модифицированная финальная функция будет выглядеть следующим образом:

$$H(x) = \psi(x) \tag{5.1}$$

Разумеется, было бы полезно сравнить между собой алгоритмы E^3 с разными значениями τ, который определяет, насколько быстро снижается уверенность в оценке. Таким образом возможно будет вывести оптимальные

значения τ для сред с разным уровнем динамичности. Подробнее об этом написано в главе 7.

Алгоритм E^3 отличается от жадного алгоритма и Frontiers exploration тем, что требует полного контроля над траекторией движения робота, поскольку проводит оптимизацию траектории движения робота по эргодичности и затраченным усилиям. Непосредственное управление движением робота не соответствует принципам и архитектуре ROS, которая подразумевает использование абстракции в виде стека навигации ROS для управления роботом. Постановка целей через стек навигации дает еще одно важное преимущество – при использовании стека навигации не требуется заботиться об избегании столкновений робота с препятствиями. И наоборот, в случае непосредственного управления движением робота требуется самостоятельно находить препятствия и огибать их.

С точки зрения архитектуры наиболее правильным решением было бы реализовать механизм оптимизации траектории из алгоритма E^3 в виде глобального планировщика. Такое решение органично вписывается в архитектуру стека навигации ROS и сохраняет все его достоинства. Как и все глобальные планировщики ROS, этот планировщик также должен будет учитывать расположение препятствий. Однако реализация такого решения потребует большого количества времени и специфических знаний из области высшей математики, поскольку требуется переработать всю математическую теорию данного процесса для корректной работы с препятствиями.

Необходимо иметь в виду, что метрика эргодичности используется для динамических сред, поэтому в статических средах её использование будет в лучшем случае бесполезным. По вышеизложенным причинам реализацию оптимизации траектории по эргодичности и затраченным усилиям решено было отложить. Эксперименты, проведенные авторами алгоритма E^3, показали, что версия алгоритма без оптимизации траектории работает в среде

без препятствий примерно на 10% хуже полнофункциональной версии алгоритма.

В среде с препятствиями разрыв между алгоритмами, предположительно, будет ещё меньше. Такое умозаключение основано на том, что препятствия будут сильно снижать область обзора лазерного лидара, установленного на роботе. Более того, если расстояние между ближайшими препятствиями в среднем не превышает диаметр области видимости лидара, то не так важно на каком расстоянии от препятствия проедет робот. Например, если робот движется по коридору, ширина которого меньше радиуса действия лидара, то нет никакой разницы, по левой или по правой стороне коридора едет робот – области обзора лидара в любом случае хватит на всю ширину коридора.

Таким образом, можно сделать вывод, что оптимизация траектории по эргодичности не является определяющим фактором при сравнении алгоритмов, поэтому отсутствие этой оптимизации не делает некорректными сравнения, представленные в данной работе.

В главе 4.3 указано, что вычисление какой-либо функции производится для каждой точки $x \in X$ исследуемой среды. С точки зрения математической теории алгоритма E^3 такие утверждения являются абсолютно верными. Однако при реализации алгоритма становится очевидно, что вычислить значение какой-либо функции для каждой точки исследуемой среды не представляется возможным хотя бы потому, что множество X содержит бесконечное количество точек. Более того, множество X даже не является счетным, потому что координаты точек являются рациональными числами.

Для того, чтобы реализовать алгоритм E^3 было возможно, требуется дискретное представление множества X. В качестве такого представления авторы алгоритма используют OGM. Размеры ячейки OGM можно выбрать произвольные. Для сохранения удовлетворительной точности алгоритма желательно, чтобы размеры ячейки хотя бы не превышали размеров робота.

Разбиение OGM на ячейки меньшего размера увеличивает точность алгоритма, позволяя более точно аппроксимировать форму препятствий и области видимости сенсоров робота, но также увеличивает и вычислительную сложность алгоритма, поскольку увеличивается количество ячеек.

В процессе отладки нашей реализации алгоритма E^3 в профилировщике (Рисунок 5.1) было установлено, что большую часть (~ 68%) времени выполнения алгоритма E^3 составляет процесс вычисления функции $L(x, I)$, то есть геодезических расстояний при помощи алгоритма Fast Marching Method.

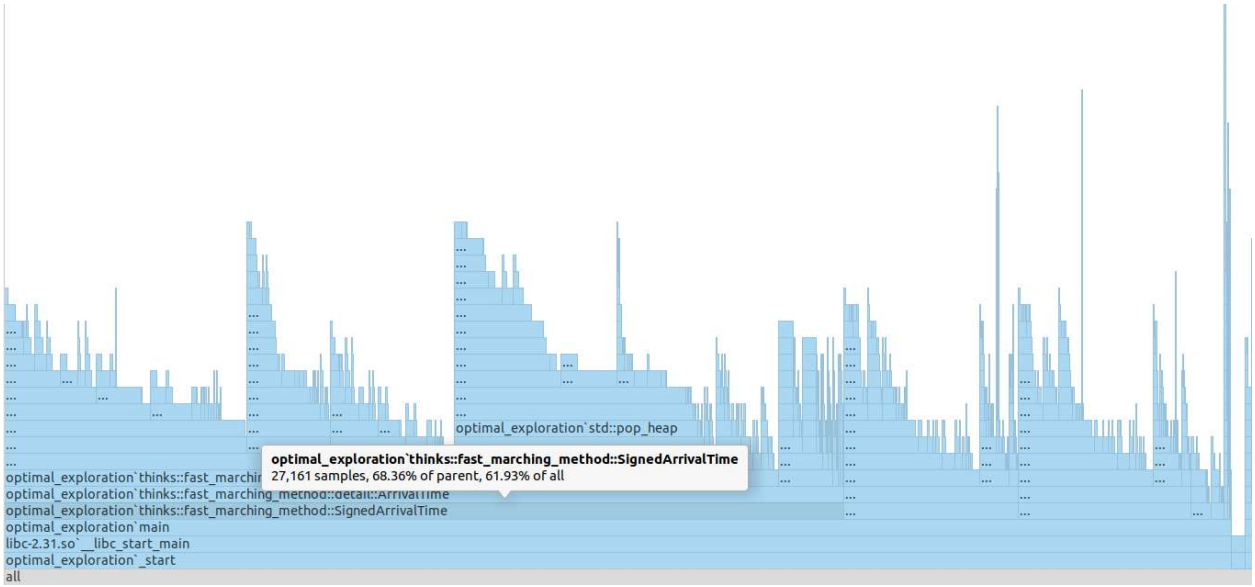

Рисунок 5.1. Отчет профилировщика о выполнении модифицированного алгоритма E^3.

Асимптотическая сложность по времени алгоритма Fast Marching Method равна $O(n \log n)$, где n – количество ячеек в OGM. Из этого можно сделать вывод, что асимптотическая сложность по времени алгоритма E^3 в целом составляет $\Omega(n \log n)$, то есть ограничена снизу функцией $f(n) = n \log n$. Таким образом, очень важно рационально выбрать размеры и количество ячеек в OGM, которую использует алгоритм E^3. Также следует отметить, что алгоритм Fast Marching Method плохо поддается распараллеливанию из-за своей последовательной природы [76], поэтому и распараллеливание алгоритма E^3 также является нетривиальной задачей. В

модифицированной версии алгоритма E^3 используется реализация алгоритма Fast Marching Method из библиотеки FMM [77].

В нашей реализации используются ячейки размера $5\ \text{см} \times 5\ \text{см}$, то есть такого же размера, как ячейки OGM используемые по умолчанию алгоритмами семейства SLAM в ROS. В таком случае каждой ячейке OGM однозначно соответствует единственная ячейка матрицы с оценками количества информации, что упрощает реализацию. При выборе размера ячейки OGM в $5\ \text{см} \times 5\ \text{см}$, общее количество ячеек OGM для тестовой среды составляет 200×200, что позволяет алгоритму E^3 работать в реальном времени.

Постановка целей для мобильного робота происходит при помощи пакета *move_base*, который реализует методы ROS Action API. ROS Action API определяет три составляющих для каждого действия (англ. action), также называемых сообщениями:

- Цель (англ. Goal)
- Обратная связь (англ. Feedback)
- Результат (англ. Result)

Разумеется, каждая задача имеет свои определения цели и результата, поэтому для каждой задачи эти три составляющих действия переопределяются. В случае задачи движения к целевой соответствующие определения сообщений цели, обратной связи и результата содержатся в пакете *move_base_msgs*.

В первую очередь рассмотрим определение сообщения цели. Сообщение цели (*MoveBaseActionGoal*) передает только целевую позу робота. Поза робота содержит не только положение робота (то есть его координаты в трехмерной системе координат), но и его ориентацию, которая представлена в виде кватерниона. В случае наземного робота будут иметь значения только координаты робота $(x; y)$ и угол поворота относительно оси Z.

Локальный планировщик ROS позволяет указать в качестве параметра допустимые отклонения от целевой позы по положению и по ориентации. Допустимое отклонение по положению будет важно в случае, когда целевая точка окажется расположенной слишком близко к препятствию, и робот не сможет её достигнуть из-за своих физических размеров. С учетом размеров робота и области действия его лазерного лидара, решено было установить допустимое отклонение по положению равным 30 см. Ориентация робота в целевой точке в данном случае не имеет значения, потому что робот оснащен лидаром с углом обзора в $360°$. В качестве допустимого отклонения по ориентации было установлено значение 2π, что делает любую ориентацию робота допустимой.

В случае *move_base* сообщение-результат является пустым, поэтому рассмотрим содержание сообщения обратной связи. Сообщение обратной связи содержит важную информацию – текущее положение робота и статус цели. Текущее положение робота важно для алгоритма E^3, потому что оно используется при решении задачи коммивояжера во время выполнения процедуры планирования маршрута. Точка среды, в которой находится робот, всегда является первой точкой маршрута.

В следующих главах приводится детальное описание трех основных концепций, которые были внедрены в алгоритм E^3 с целью того, чтобы он мог работать в неизвестных средах, которые содержат препятствия.

5.1 ВЫБОР ЦЕЛЕЙ

Алгоритм E^3 для каждой точки исследуемой среды подсчитывает значение функции $H(x)$, которая в численном виде показывает важность каждой точки для исследования. Далее из этих значений составляется двумерная матрица-карта данной среды. На этой карте при помощи алгоритма Watershed находится l точек с максимальными значениями функции $H(x)$.

При реализации этой процедуры было обнаружено, что отсутствует реальная необходимость в использовании алгоритма Watershed, который обычно используется для задач кластеризации изображений, то есть для нахождения четких границ кластеров, центрами которых становятся эти вершины. В данном случае требуется лишь найти l максимальных значений в матрице, что можно сделать, просеяв все элементы матрицы через двоичную кучу [78]. Асимптотическая сложность добавления элемента в двоичную кучу составляет $O(\log n)$, соответственно, сложность просеивания всех n элементов составит $O(n \log n)$.

В качестве реализации двоичной кучи используется очередь с приоритетом. Метрикой, по которой сравниваются элементы очереди, является значение $H(x)$. Алгоритм последовательно проходит по всем ячейкам из OGM и кладет в очередь пару, состоящую из координат ячейки и её значения $H(x)$. После просеивания в очереди будут содержаться элементы, отсортированные по убыванию $H(x)$. Затем следует извлечь из очереди требуемое количество элементов.

При извлечении элемента из очереди необходимо учитывать расстояние между ним и уже выбранными точками. Если же это расстояние не учитывать, то может оказаться так, что все l точек с максимальными значениями $H(x)$ окажутся в соседних ячейках OGM (в данном случае в 5 см друг до друга), потому что все они будут находиться в середине одного и того же кластера неизвестных точек. Такое происходит, потому что все точки в центре самого большого кластера неизвестных точек будут иметь самое большое значение $H(x)$, поскольку они имеют самое большое геодезическое расстояние до ближайших границ исследованных территорий.

В нашей реализации проверяется, что расстояние между любыми выбранными точками составляет не менее, чем ширина исследуемой среды, поделенная на l. Значение выбрано эвристически исходя из предположения,

что равномерное распределение целевых точек по исследуемой среде обеспечивает хорошее покрытие всей исследуемой среды.

В процессе реализации алгоритма E^3 эмпирическим путем было установлено, что даже $l = 10$ является чрезмерно большим. Для нормальной работы алгоритма достаточно $l = 5$. Более подробно об этом написано в главе 5.3. Даже при снижении l до 5, на завершающих этапах исследования среды возникают ситуации, когда не удается найти 5 «полезных» целей (Рисунок 5.3). В таком случае среди целей могут оказаться незначительные кластеры неизвестных ячеек OGM, которые остаются в углах уже исследованных комнат, или шум «соль и перец» [79] (Рисунок 5.2). Формально такое поведение является полностью корректным, поскольку эти кластеры также являются неизвестными территориями, пусть и малого размера. Например, в одной из таких неизвестных ячеек OGM может оказаться тонкий столбик или другое маленькое препятствие.

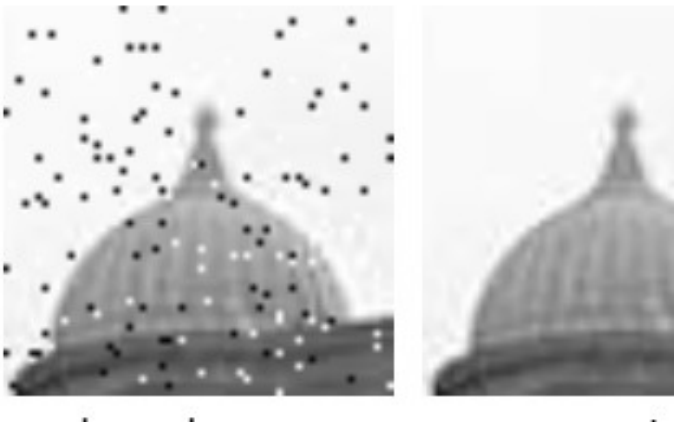

Рисунок 5.2. Слева – рисунок с шумом «соль и перец», справа – тот же рисунок без шума[5].

В реальной жизни вероятность такого события очень мала, поэтому повторное посещение роботом уже исследованных территорий ради очистки карты среды от незначительного шума является неоправданным и, чаще всего, бесполезным расходом ресурсов. Для очистки карты от шума «соль и перец»

[5] https://craftofcoding.wordpress.com/2017/05/29/experiential-programming-with-julia-ii/

гораздо больше подходят алгоритмы фильтрации изображений, которые позволяют достигнуть желаемого затрачивая гораздо меньше ресурсов.

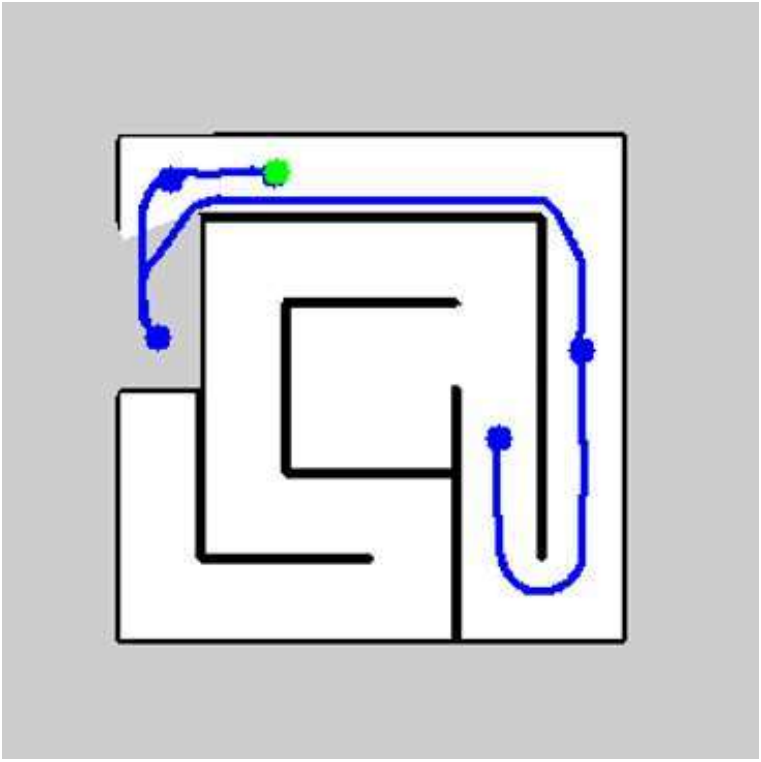

Рисунок 5.3. Избыточное количество целей (синие точки). В данном случае достаточно одной цели, которая находится на неизвестной территории (выделена серым цветом).

«Бесполезные» цели отличаются от «полезных» тем, что их значение $H(x)$ относительно невелико. Причина этого кроется в том, что геодезическое расстояние до уже исследованных областей также будет невелико, потому что такие кластеры – маленькие. Нам удалось решить проблему «бесполезных» целей введением динамического количества целей $l \leq 5$. Каждая извлеченная из очереди точка сравнивается с предыдущей извлеченной точкой. Как только значение функции $H(x)$ у извлеченной точки окажется меньше, чем 75% от значения $H(x)$ у предыдущей точки, извлечение точек из очереди прекращается. Например, для набора точек со следующими значениями $H(x)$:

$$\{150, 146, 139, 15, 13, 10, \dots, 0\},$$

в качестве целей будут выбраны только первые три точки слева.

5.2 ЗАПОЛНЕНИЕ ПРЕПЯТСТВИЙ

Робот находит препятствия в исследуемой среде при помощи лазерного лидара. Лидар испускает во всех направлениях пучки лазерных лучей. Лазерные лучи отражаются от препятствий и улавливаются лидаром при помощи фотоэлемента. Таким образом вычисляется расстояние до ближайшего препятствия по каждому из направлений.

Алгоритмы SLAM помечают на OGM как занятые только те ячейки, в которых произошло отражение лазерных лучей. При таком алгоритме работы, как занятые помечаются только те ячейки, которые находятся на внешнем контуре препятствия. Ячейки внутри препятствия остаются помеченными как неизвестные. Теоретически такое поведение является абсолютно правильным, поскольку действительно неизвестно, что находится внутри видимого сенсору участка внешней части препятствия. Выяснить это в процессе исследования при помощи только лидара невозможно.

Тем не менее, такое поведение вызывает большие проблемы в работе алгоритма E[3]. По правилам расчета оценки количества информации $\hat{\psi}(x)$, для точек, которые находятся внутри препятствия, оценка количества информации вычисляется по функции $\Upsilon(x, \hat{\psi}, I)$, поскольку они не находятся в области видимости сенсоров робота. В функции $\Upsilon(x, \hat{\psi}, I)$ происходит вычисление геодезического расстояния $L(x, I)$, которое зависит от индикаторной функции $I_t(x)$.

Так как робот никогда не сканировал своими сенсорами точки внутри препятствия, индикаторная функция $I_t(x)$ возвращает 0, что приводит к вычислению ненулевой оценки $\hat{\psi}(x)$, тем же образом, как и для любой другой неизвестной территории. Оценка количества информации $\hat{\psi}(x)$ используется при вычислении функции $H(x)$, которая численно определяет важность какой-либо территории для изучения.

Фактически, такое поведение приводит к образованию кластера из точек с ненулевым значением $H(x)$ в контурах препятствия, внутри. Далее, алгоритм E^3 устанавливает цель в центр этого кластера и планирует маршрут, проходящий через неё. Разумеется, достигнуть этой целевой точки для мобильного робота невозможно, поэтому после завершения полной петли вокруг препятствия эта цель будет отменена. Однако кластер точек с ненулевыми значениями $H(x)$ не исчезнет, поэтому на следующем шаге алгоритма E^3 в его центр снова будет установлена цель, которую невозможно достигнуть.

Таким образом, любое препятствие, которое находится в исследуемой среде, способно заблокировать работу оригинального алгоритма E^3, переведя его в бесконечный цикл. Для решения данной проблемы требуется, чтобы у всех точек среды, находящихся внутри препятствия, были нулевые значения функции $H(x)$. Физический смысл данного утверждения состоит в том, что любые замкнутые препятствия следует рассматривать как монолитные, поскольку область внутри них в любом случае будет недоступна для исследования.

В нашей модификации алгоритма E^3 обнуление значений функции $H(x)$ для всех точек внутри замкнутых контуров препятствий было реализовано путем модификации индикаторной функции $I_t(x)$:

$$I_{t+\Delta t}(x') = \begin{cases} 1 & : \quad x' \in D(x,\delta) \ \vee \ x' \in O \\ I_t(x') & : \quad x' \notin D(x,\delta) \ \wedge \ x' \notin O \end{cases} \tag{5.2.1}$$

В формуле, представленной выше, O – множество точек $x \in X$ исследуемой среды, которые находятся внутри препятствий. Таким образом, точки, находящиеся внутри препятствий, эквивалентны уже посещенным точкам, то есть в двоичной матрице значений $I_t(x)$ на их месте устанавливается 1. После этого для данных точек геодезические расстояния $L(x, I)$ считаются равными нулю, соответственно $\hat{\psi}(x)$ тоже равны нулю. Значения функции $G_t(x)$, которая представляет собой меру уверенности в

оценке количества информации, для таких точек не имеют значения, потому что при расчете $H(x)$ по формуле (4.3.1.9) другой член произведения (а именно $\hat{\psi}(x)$) будет равен нулю, а следовательно, и всё произведение тоже будет равно нулю:

$$H(x) = 0, \qquad \forall x \in O \tag{5.2.2}$$

Для того, чтобы реализовать такую процедуру, требуется найти способ, который позволит определять, что какая-либо точка исследуемой среды находится внутри препятствия. Для этого в первую очередь необходимо каким-либо образом найти контуры препятствия. Задача нахождения контуров является нетривиальной. Для её решения используются достаточно сложные алгоритмы, реализация которых собственными силами требует больших временных затрат. Необходимость в поиске контуров чаще всего возникает при обработке изображений, поэтому библиотеки для работы с изображениями (например, библиотека OpenCV [80]) имеют в своем составе функции для поиска контуров на изображении. В данной работе решено было использовать именно OpenCV.

Преобразование OGM в черно-белое изображение используется достаточно часто. Например, такой подход используется в ROS для визуализации карты исследованной территории, что значительно помогает в отладке алгоритмов. OGM в визуализаторе RViz [81] кодируются следующим образом (Рисунок 5.4):

- Свободные ячейки кодируются белым цветом.
- Занятые ячейки кодируются черным цветом.
- Ячейки, состояние которых неизвестно, кодируются серым цветом.

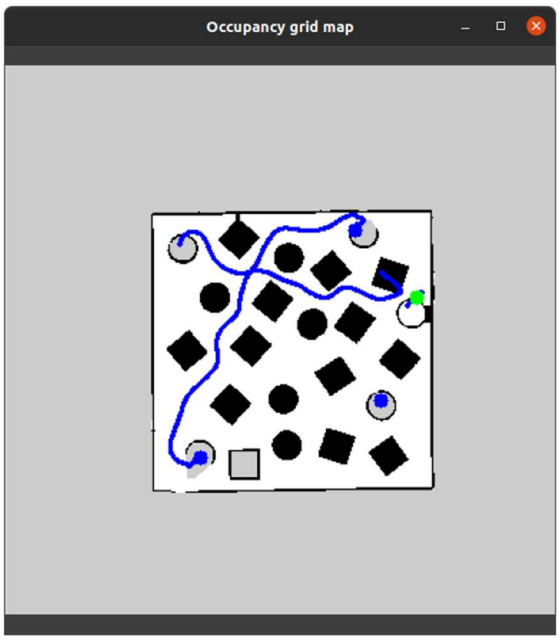

Рисунок 5.4. Представление глобальной карты среды в виде OGM.

Функция поиска контуров *findContours(...)* из библиотеки OpenCV работает только с бинарными изображениями, то есть состоящими только из черных и белых пикселей. Чтобы удовлетворить это требование пришлось частично модифицировать стандартную схему цветового кодирования OGM. Поскольку в данном случае значение имеют только занятые ячейки, которые кодируются черным цветом, следует их отфильтровать, закодировав все остальные ячейки, то есть свободные и неизвестные, белым цветом.

Прежде чем приступать к поиску контуров на изображении, следует иметь в виду, что на изображении может присутствовать шум, который будет мешать алгоритму поиска контуров. В данном случае будет присутствовать шум «соль и перец», который вызван погрешностями измерений лазерного лидара и ошибками вычислений с плавающей точкой в алгоритме SLAM. Чтобы получить хорошие результаты в поиске контуров, изображение рекомендуется очистить от шума.

Поскольку изображение бинарное, наилучшей фильтрации шума можно добиться, используя морфологические трансформации [82]. В ходе

экспериментов следующая последовательность морфологических трансформаций показала наилучший результат фильтрации шума:

1) Морфологическое открытие (opening) с ядром размером 7×7.
2) Морфологическое закрытие (closing) с ядром размером 5×5.

Данная последовательность морфологических трансформаций является морфологическим фильтром. При этом последовательность выполнения трансформаций очень важна. Морфологическое открытие удаляет шумовые пиксели белого цвета. Такие пиксели обычно появляются на внешних углах прямоугольных препятствий, из-за того, что лазерные лучи, попавшие на внешний угол препятствия, могут отразиться в непредсказуемом направлении. Важно, чтобы морфологическое открытие выполнялось первым. В противном случае контур прямоугольного препятствия в углах будет разорван.

Морфологическое закрытие удаляет шумовые пиксели черного цвета, которые появляются не так часто, как белые. Единичные пиксели черного цвета формально также являются замкнутыми контурами, однако они не требуют выполнения процедуры заполнения препятствий. Фактически, они являются ложными срабатываниями алгоритма, поэтому также подлежат удалению.

Морфологическое открытие использует ядро увеличенного размера для того, чтобы контуры, разорванные 1-2 пикселями белого цвета, после фильтрации оказались замкнуты. Разрывы контуров обычно происходят из-за погрешностей показаний лидара на внешних углах препятствий. Кроме того, в препятствиях действительно могут быть отверстия размером около 5-10 см, то есть слишком маленькие для проезда робота. В любом из этих случаев попасть внутрь препятствия для робота невозможно, поэтому их контуры также должны быть замкнутыми.

В процессе реализации были протестированы различные настройки ядер для морфологических трансформаций. В частности, были протестированы

квадратные, крестообразные и круглые ядра размерами от 3×3 до 11×11. Лучший результат показали квадратные ядра размерами 7×7 для морфологического открытия и 5×5 для морфологического закрытия. Ядра меньшего размера не обеспечивают достаточную фильтрацию шума «соль и перец», ядра большего размера приводят к объединению контуров нескольких близко расположенных препятствий.

Стоит отметить, что фильтрация морфологическими трансформациями не обеспечивает полного удаления шума «соль и перец», однако значительно снижает его количество. Применение дополнительного фильтра с другим принципом работы после морфологического фильтра потенциально может удалить ещё больше шума, но это поможет только в редких случаях с большими препятствиями, которые имеют протяженные и разветвленные контуры. С другой стороны, применение дополнительного фильтра повлияет негативным образом на производительность алгоритма E^3. Использование только морфологического фильтра обеспечивает компромисс между скоростью работы алгоритма и качеством фильтрации шума.

Функция *findContours(…)* из библиотеки OpenCV имеет следующую сигнатуру:

void findContours(InputArray image,
OutputArrayOfArrays contours,
OutputArray hierarchy,
int mode,
int method)

Данная функция имеет множество режимов работы, которые настраиваются флагами. В первую очередь нас интересует флаг *mode*, который отвечает за то, в каком режиме следует извлекать контуры. Этот флаг определяет структуру массива выходного *hierarchy*, в котором в определенном порядке содержатся все найденные контуры. Для данной задачи лучше всего подходит режим *RETR_CCOMP*, который возвращает все найденные контуры

в виде двухуровневой иерархии. Двухуровневая иерархия подразумевает, что объекты на изображении могут иметь *внутренние* и *внешние* контуры.

Внешние контуры существуют у каждого объекта и представляют собой контуры в прямом смысле этого термина. Внутренние контуры имеются только у объектов, имеющих замкнутые области. Контур, окружающий замкнутую область в объекте, является внутренним контуром. Например, фигура в форме буквы «О» имеет как внешний, так и внутренний контур. А фигура в форме буквы «С» имеет только внешний контур. При этом принимается допущение, что препятствия могут иметь только одну замкнутую область, независимо от того, сколько внутренних полостей на самом деле имеет препятствие, поскольку технически невозможно определить точное расположение и количество внутренних полостей, сканируя наружную поверхность препятствия лидаром.

Такая структура иерархии контуров удобна тем, что позволяет разделить контуры найденных препятствий на два вида:

- Препятствия с замкнутой областью (в форме буквы «О»).
- Препятствия без замкнутой области (в форме буквы «С»).

К препятствиям второго вида относятся также маленькие препятствия размером в 1-4 ячейки OGM или длинные и тонкие препятствия, толщиной в 1-2 ячейки OGM (например, стены). Для препятствий такого вида не требуется выполнять процедуру заполнения препятствия, поскольку внутри них отсутствует область из ячеек OGM, состояние которых неизвестно. Такие препятствия можно пропустить.

Далее необходимо проверить, что у каждого найденного препятствия с замкнутой областью отсутствуют другие препятствия внутри этой области. Это нужно для того, чтобы не обнулить значения $H(x)$ сразу во всей исследуемой среде. Например, предположим, что нашлось такое препятствие, которое содержит внутри себя другие препятствия. Если внутри этого

препятствия нашлись другие препятствия, значит и робот также находится внутри этого препятствия. Такое заключение можно сделать, потому что в противном случае наземный робот не сможет попасть внутрь этого препятствия и найти внутри него другие препятствия, даже если они действительно там расположены. Далее обратим внимание на внутренний контур этого препятствия. Он является замкнутым, а значит робот не имеет возможности выбраться из области, окруженной внутренним контуром. Следовательно, вся исследуемая среда ограничена этим внутренним контуром. Это значит, что данное препятствие является стенами запертой комнаты, внутри которой находится робот.

Осуществить такую проверку позволяет функция *pointPolygonTest(…)* из библиотеки OpenCV. Она принимает на вход точку и контур и возвращает информацию о том, где относительно контура находится точка: внутри, снаружи или на его границе. При этом достаточно проверять только одну точку из каждого контура, поскольку контуры не могут пересекаться.

После того, как необходимые контуры препятствий будут найдены, необходимо пройти в цикле по всем белым пикселям внутри них. Каждый пиксель изображения имеет отображение 1-в-1 в соответствующую ячейку OGM, поскольку изображение является OGM, закодированной цветами. Таким образом, необходимо пометить все ячейки OGM, которые находятся внутри препятствий так, чтобы функция $I_t(x)$ возвращала для них значение 1.

Подводя итоги, процедура заполнения препятствий выполняется следующим образом:

1) Найти контуры препятствий на OGM.
2) Отфильтровать все незамкнутые контуры.
3) Отфильтровать замкнутые контуры, которые содержат внутри себя другие контуры.
4) Для каждой точки внутри оставшихся замкнутых контуров определить, что функция $I_t(x) = 1$.

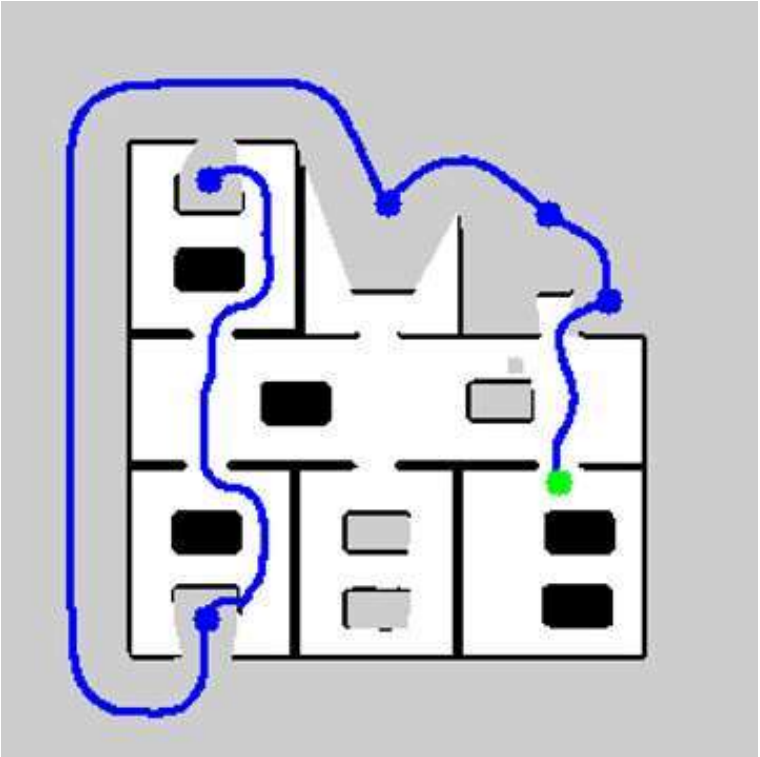

Рисунок 5.5. Обнаруженные препятствия выделены черным. В необнаруженных препятствиях могут быть установлены цели (синие точки).

На рисунке (Рисунок 5.5) можно увидеть результат работы процедуры заполнения препятствий. Препятствия, контуры которых были замкнуты, успешно найдены и заполнены пикселями черного цвета. Внутри них отсутствуют ячейки OGM, состояние которых неизвестно (изображены серым цветом), то есть заполненные препятствия воспринимаются алгоритмом E^3 как монолитные. И наоборот, препятствия такой же формы, но контур которых оказался по каким-либо причинам не замкнут, алгоритмом заполнения препятствий не обнаружились. Внутри них всё так же находятся ячейки OGM, состояние которых неизвестно, поэтому алгоритм E^3 устанавливает внутрь них цели (изображены в виде синих точек).

5.3 ПЛАНИРОВАНИЕ ПУТИ

На каждом шаге алгоритм E^3 находит l точек с максимальным значением функции $H(x)$, по которым строит маршрут путем решения задачи коммивояжера. В процессе реализации и отладки этой процедуры было

установлено, что даже $l = 10$ является чрезмерно большим. Проблема состоит в том, что маршруты для исследования среды подразумевают движение по неизвестным территориям. Так как расположение препятствий на этих территориях неизвестно, это значит, что маршруты планируются в условиях недостатка информации. В таких случаях глубина планирования, которая применяется при $l = 10$ является явно избыточной.

Длина маршрута между двумя точками, расположенными на неизвестной территории, принимается равной расстоянию между ними по прямой. С другой стороны, между этими точками могут оказаться стены, выстроенные по типу лабиринта. В таком случае длина маршрута между этими точками может оказаться гораздо больше расстояния по прямой. Любая стена, которая попадется на пути движения робота, может сделать решение задачи коммивояжера неправильным. Можно сделать вывод, что не имеет смысла тратить значительное количество усилий на решение задачи коммивояжера. С учетом того, что алгоритм для решения задачи коммивояжера имеет факториальную сложность, было принято эвристическое решение ограничить максимальное количество целей $l \leq 5$.

Авторы алгоритма E^3 не указывают явно, какую именно метрику расстояния между точками они используют. Так как они проводили эксперименты в среде без препятствий, логично предположить, что они использовали евклидово расстояние, то есть кратчайшее расстояние по прямой. Однако в среде с препятствиями кратчайшего пути по прямой между двумя точками может не быть. В таком случае, при решении задачи коммивояжера с использованием евклидова расстояния, реальная длина маршрута между точками может быть значительна недооценена.

В лабиринте может возникнуть следующая неблагоприятная ситуация: длина маршрута между парой точек, близких по евклидовому расстоянию, может оказаться значительно длиннее, чем между другой парой точек, более далеких по евклидову расстоянию. В таком случае решение задачи

коммивояжера даст неправильную последовательность точек на маршруте (Рисунок 5.6).

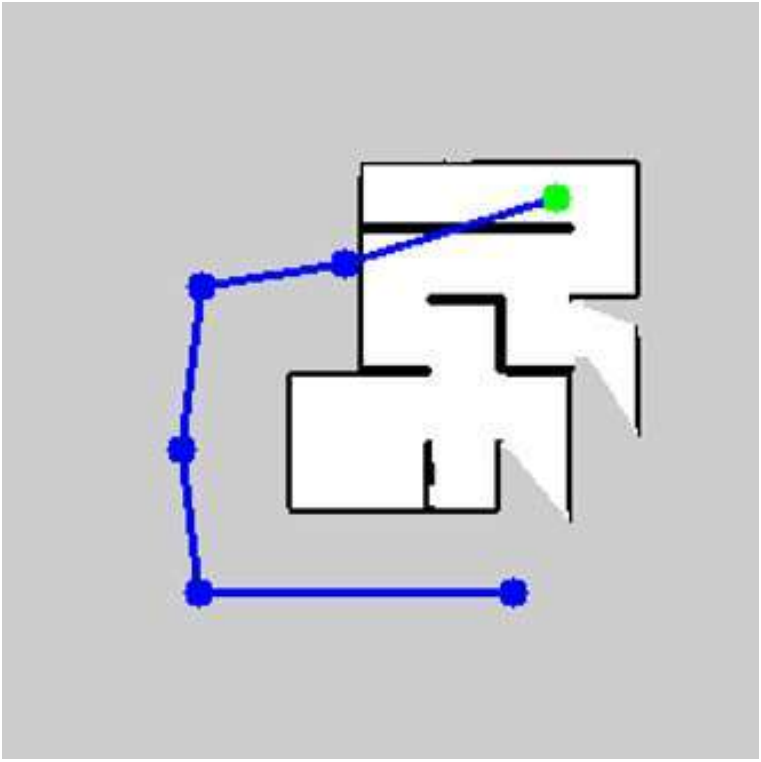

Рисунок 5.6. Использование метрики евклидова расстояния в лабиринте приводит построению неправильного пути (синяя линия).

В нашей модификации алгоритма E^3 данная проблема решается путем измерения реальной длины маршрута между точками. В пакете *move_base* имеется сервис (ROS Service API), который позволяет при помощи глобального планировщика построить маршрут между двумя произвольными точками. Вызов сервиса происходит следующим образом:

1) Составляется сообщение-запрос, содержащее координаты начальной и конечной точек маршрута.
2) Сервис планирует маршрут при помощи глобального планировщика.
3) Сервис возвращает сообщение-ответ, которое содержит маршрут в виде массива промежуточных позиций робота.

Следует отметить, что вызов сервиса является блокирующей функцией. Внутренняя реализация вызова сервиса использует для коммуникации топики

ROS, поэтому выполняется дольше, чем простой вызов функции в исполняемом коде. По этим причинам желательно минимизировать количество вызовов сервиса. Для решения задачи коммивояжера требуется знать расстояние между каждой парой точек, следовательно количество маршрутов, которые требуется спланировать, можно посчитать по следующей формуле:

$$C = \frac{l!}{(l-2)!\,2!} \tag{5.3.1}$$

Для $l = 5$ требуется спланировать 10 путей, а для $l = 10$ требуется спланировать уже 45 путей. Таким образом, с точки зрения затрат вычислительных ресурсов, рациональнее выбирать меньшее значение l.

Далее, по массиву промежуточных позиций робота, который прислал сервис планирования, требуется вычислить длину маршрута. Сделать это можно путем суммирования расстояний между парами соседних элементов массива промежуточных позиций робота длины n:

$$L = \sum_{i=0}^{n-2} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{5.3.2}$$

Найденная по формуле длина маршрута используется как метрика расстояния при решении задачи коммивояжера. В процессе реализации данной процедуры эмпирическим путем выяснился любопытный факт. Маршруты $A \to B$ и $B \to A$, спланированные глобальным планировщиком ROS имеют различную длину, где разница колеблется в пределах 1-5%. Теоретически такого быть не может, поскольку в исследуемой среде отсутствуют области с одностороннем движением. Скорее всего эта разница обусловлена накоплением ошибки из множества операций с плавающей точкой, таких как возведение в степень или извлечение квадратного корня.

В процессе реализации и отладки была выявлена ещё одна проблема алгоритма E^3: алгоритм не меняет последовательность целей на маршруте. Это не самое оптимальное поведение, поскольку в процессе движения по

маршруту робот постоянно собирает новую информацию об исследуемой среде. Например, робот может обнаружить на своем пути глухую стену, которая не позволит ему двигаться дальше по запланированному маршруту. Поэтому робот должен периодически сортировать последовательность целей на маршруте для того, чтобы быть уверенным, что он движется по кратчайшему маршруту (Рисунок 5.7).

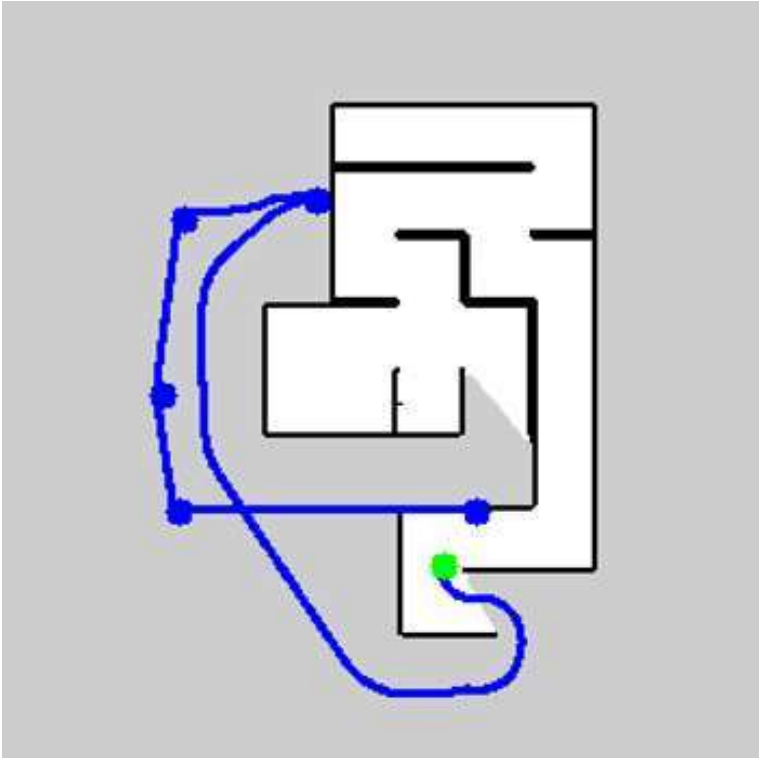

Рисунок 5.7. Последовательность целей (синие точки), не сортировалась с самого начала исследования, поэтому на уже не является оптимальной.

Процедура сортировки последовательности целей требует решения задачи коммивояжера, которая занимает много вычислительных ресурсов. Таким образом, необходимо найти компромисс между поддержанием запланированного маршрута в актуальном состоянии и расходом вычислительных ресурсов. В нашей реализации сортировка маршрута происходит после достижения каждой из целей.

В общем случае такой периодичности достаточно для того, чтобы не расходовать лишние ресурсы на движение по неоптимальным маршрутам. Однако может возникнуть ситуация, когда маршрут изначально был

запланирован так, что первая цель оказывается дальше всего, и для её достижения требуется проехать почти по всему лабиринту.

Для отсечения подобных плохих сценариев может быть полезным использование какой-либо эвристики. Например, перепланировать весь маршрут, когда после обнаружения препятствия, глобальный планировщик планирует новый маршрут, длина которого значительно больше изначально запланированного, т.е. того, на основе длины которого решалась задача коммивояжера.

6 ЭКСПЕРИМЕНТЫ

Для того, чтобы сравнить энергоэффективность модифицированного алгоритма E^3 с энергоэффективностью жадного алгоритма и Frontier exploration требуется провести серию экспериментов. В данной работе решено было провести все эксперименты в симуляторе, что позволяет ускорить процесс подготовки окружения и значительно сократить время на выполнение большого количества экспериментов.

Для получения более надежных результатов экспериментов необходимо, чтобы эксперименты проводились во множестве различных тестовых сред разных типов. В данной работе решено было использовать два типа тестовых сред:

- Лабиринты различной сложности.
- Помещения офисного типа с различной конфигурацией комнат и расположением препятствий.

Лабиринты являются классическими тестовыми средами для алгоритмов, которые подразумевают планирование маршрутов движения. Расстояния между различными ключевыми точками в лабиринтах могут быть очень большими, поэтому необходимо рационально планировать порядок посещения ключевых точек. Результат в лабиринтах наглядно показывает, насколько хорошо работает планирование маршрутов движения.

Помещения офисного типа являются наиболее приближенными к реальности областями применения алгоритмов исследования неизвестной среды. Они имеют несколько основных отличий от лабиринтов. Во-первых, в помещениях офисного типа имеется значительное количество препятствий в форме прямоугольников и квадратов. Эти препятствия симулируют стандартную офисную мебель, такую как столы, стулья и тумбочки. Во-вторых, структура стен в них не такая запутанная, как в лабиринтах. Структура стен выстроена в виде комнат и коридоров различной конфигурации. Всего

было создано 18 тестовых сред, 9 из которых были лабиринтами, а остальные 9 – помещениями офисного типа.

Все тестовые среды должны быть одинакового размера, поскольку модифицированный алгоритм E^3 наследует от оригинального алгоритма один существенный недостаток – ему требуется знать точные размеры исследуемой среды до начала исследования. Подробнее он описан в главе 7.1. В данной работе было решено сделать все тестовые среды размером $10\text{м} \times 10\text{м}$. Такой размер не слишком большой, чтобы появились проблемы с вычислительной сложностью алгоритма E^3, однако и не слишком мал, чтобы задача исследования такой среды становилась тривиальной.

Далее необходимо было определить толщину стен в тестовых средах таким образом, чтобы максимизировать количество полезного пространства при фиксированном общем размере тестовой среды. По этой причине в тестовых средах используются стены толщиной 10см. Такое значение кратно 5см, поэтому толщина стен составляет 2 ячейки OGM.

Использование тестовых сред фиксированного размера значительно упрощает определение метрики, которая будет в процентном виде выражать, насколько исследована тестовая среда. Метрика подсчитывается на основе карты в формате OGM, предоставленной алгоритмом SLAM. В OGM подсчитывается количество ячеек, состояние которых известно (не важно, свободны они или заняты), которое затем делится на общее количество ячеек в тестовой среде (40,000 ячеек для среды размером $10\text{м} \times 10\text{м}$). Данное соотношение отображается в процентном виде.

Помимо измеряемых метрик для эксперимента необходимо определить критерий остановки, то есть определить, в каком случае эксперимент считается завершенным. В данной работе мы ориентировались на критерий остановки, встроенный в реализацию алгоритма Frontier exploration из пакета *explore_lite*. Он сравнивает размеры оставшихся на карте фронтиров с некоторым пороговым значением. В случае, если на карте отсутствуют

фронтиры, размер которых больше порогового значения, алгоритм считает исследование среды законченным и завершается. Таким образом, алгоритм Frontier exploration исследует неизвестную среду только до определенного предела, после чего завершается. Остальные два алгоритма никаких критериев остановки не имеют. Однако для того, чтобы сравнение было объективным, необходимо использование одинаковых критериев остановки для всех трех алгоритмов.

В итоге эксперименты решено было проводить по следующему регламенту:

- Во всех тестовых средах первым запускается алгоритм Frontier exploration, который исследует среду до момента своего завершения.
- Затем измеряется процент исследования среды, на котором завершился алгоритм Frontier exploration.
- Далее последовательно запускаются жадный алгоритм и E^3, которые принудительно завершаются при достижении такого же процента исследования среды, как у алгоритма Frontier exploration.

Такой регламент проведения экспериментов позволяет удостовериться, что все три алгоритма провели исследование одной и той же среды в одном и том же объеме.

В процессе прохождения эксперимента вычислялись метрики процента исследованной среды и общего пройденного расстояния. Общее пройденное расстояние вычисляется на основе данных одометрии, которые генерируются в ROS на основе информации о локализации робота в данный момент из алгоритма SLAM, а также на основе данных с колесных энкодеров робота.

Данные одометрии поступают из топика */odom* в виде сообщений через определенные временные интервалы. Они представляют собой координаты (x, y) робота в глобальной системе координат. Вычислить расстояние $S_{\Delta t}$, пройденное роботом за период времени Δt можно, зная координаты робота в

предыдущий момент времени (x_t, y_t) и в текущий момент времени $(x_{t+\Delta t}, y_{t+\Delta t})$:

$$S_{\Delta t} = \sqrt{(x_{t+\Delta t} - x_t)^2 + (y_{t+\Delta t} - y_t)^2} \quad (6.1)$$

Накапливая путем суммирования расстояния $S_{\Delta t}$ в течение всего времени работы алгоритма можно подсчитать общее пройденное расстояние.

6.1 ПОДГОТОВКА ТЕСТОВЫХ СРЕД ДЛЯ ЭКСПЕРИМЕНТОВ

Для того, чтобы использовать карту в симуляторе Gazebo, она должна быть смоделирована в 3D редакторе. Ручное создание 3D моделей препятствий и расстановка их на соответствующие места занимают большое количество времени. Для решения этой проблемы была использована утилита LIRS WCT TOOL [33], разработанная в нашей лаборатории. Она позволяет создавать 3D модели для использования в симуляторе Gazebo на основе 2D изображений в формате PNG, которые представляют из себя планы помещений со стенами и препятствиями при виде сверху. Помимо самих 3D моделей, утилита генерирует и конфигурационный файл в формате **.world* для симулятора Gazebo, который позволяет сразу же запустить симуляцию на сгенерированной карте.

Утилита имеет удобный графический интерфейс, который позволяет указывать параметры генерации 3D модели при помощи стандартных элементов пользовательского интерфейса. Кроме того, существует консольная версия утилиты, в которой настройки указываются при помощи параметров командной строки. Для массовой генерации множества схожих тестовых сред лучше подходит консольная версия, поэтому именно она используется в данной работе.

Данная утилита принимает на вход бинарное черно-белое изображение, которое представляет собой карту среды при виде сверху. Высота объектов на карте кодируется градациями серого, количество которых составляет 256 на стандартном изображении с глубиной цвета 8 бит. Необходимо отметить, что

белым цветом кодируются препятствия максимальной высоты, что не соответствует интуитивным представлениям о черно-белых картинках с изображениями лабиринтов, на которых стены обычно обозначаются черным цветом. При конвертации такой картинки в 3D модель требуется указать флаг инвертирования цветов изображения.

Существует множество инструментов, которые позволяют сгенерировать лабиринт необходимого размера в виде черно-белого изображения. Как правило, все эти инструменты генерируют черно-белые изображения лабиринтов, на которых стены изображены черным цветом. Для того, чтобы использовать такой лабиринт в утилите LIRS WCT TOOL, необходимо инвертировать цвета на изображении при помощи соответствующего параметра утилиты. В данной работе использовался генератор лабиринтов с сайта Discovery Education [82], потому что он создает лабиринты в виде простых черно-белых изображений и обладает широкими возможностями по настройке параметров создаваемого лабиринта.

Для проведения экспериментов требуются лабиринты разной сложности. При этом в инструментах для генерации лабиринтов, как правило, отсутствует отдельный параметр сложности лабиринта, вместо этого имеется параметр размера лабиринта. Таким образом сложность лабиринта напрямую зависит от его размеров. Однако для проведения экспериментов требуются тестовые среды одинаковых размеров. В итоге решено было поступить следующим образом:

1) Генерировались по 3 лабиринта с размерами 6×6, 8×8 и 10×10 клеток.
2) Созданные изображения лабиринтов приводились к единому размеру в 200×200 пикселей, что будет соответствовать размерам $10\text{м} \times 10\text{м}$ при использовании OGM с ячейками стандартного размера $5\text{см} \times 5\text{см}$.

3) Изображения, растянутые до размеров 200×200 пикселей бинаризировались пороговым значением для того, чтобы избавиться от артефактов, возникших после растяжения изображения.

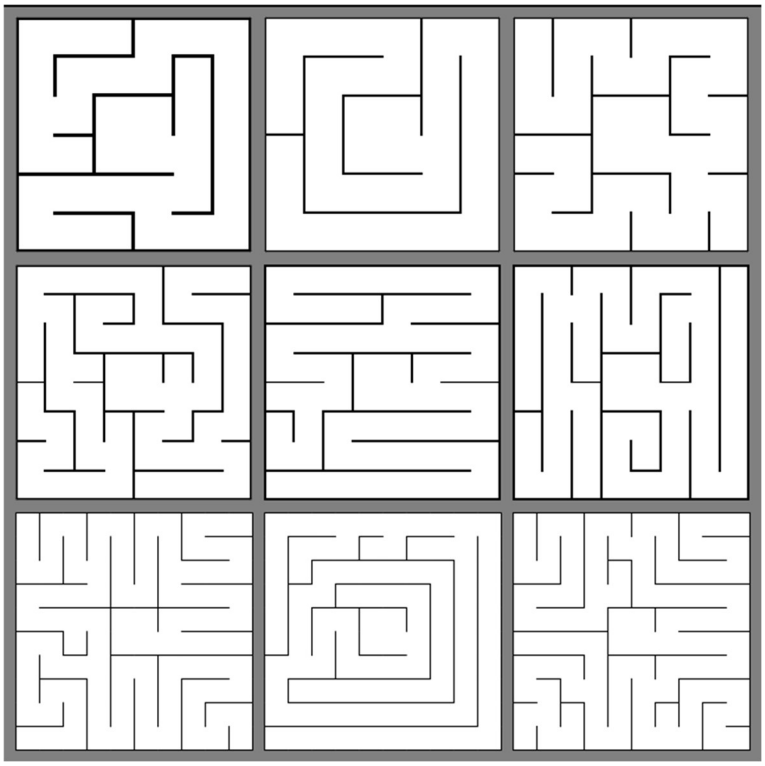

Рисунок 6.1. Автоматически сгенерированные тестовые среды в виде лабиринтов.

Сгенерированные лабиринты (Рисунок 6.1) имеют одну общую особенность. В центре каждого лабиринта имеется небольшая комната квадратной формы, из которой существует только один выход. Данная комната нужна для того, чтобы на начальном этапе исследования ограничить область видимости лидара робота, который располагается в центре тестовой среды (в центре данной комнаты). Таким образом, в любом лабиринте робот в начале исследования среды имеет одинаковое количество знаний о ней, независимо от расположения стен в лабиринте.

Для автоматической генерации помещений офисного типа с комнатами инструментов не нашлось, поэтому расположение комнат и препятствий было спроектировано в ручном режиме. Происходило это следующим образом:

1) В графическом редакторе создавался холст размером 200×200 пикселей.
2) На холсте рисовались стены толщиной в 2 пикселя для того, чтобы разделить холст на комнаты и коридоры.
3) В стенах проделывались проемы для проезда робота.
4) Создавались два примитива препятствий – квадратное и прямоугольное, у которого длина в два раза больше ширины.
5) Примитивы препятствий клонировались и расставлялись в большом количестве в различных местах изображения. Расстановка препятствий была максимально приближена к расстановке столов и стульев в офисных кабинетах.

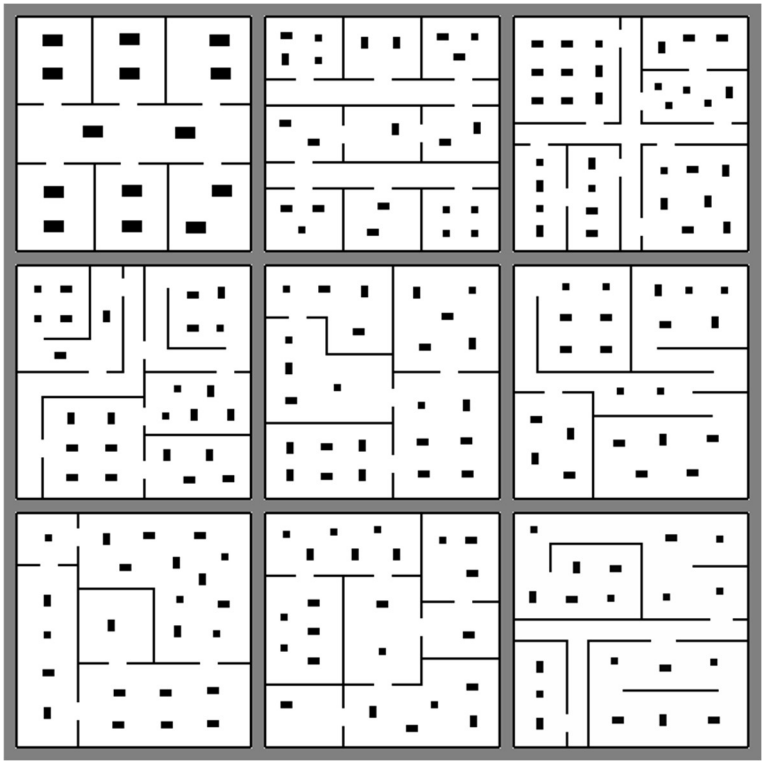

Рисунок 6.2. Созданные вручную тестовые среды офисного типа с комнатами и препятствиями.

Созданные в ручном режиме изображения тестовых сред офисного типа (Рисунок 6.2) сохранялись в формате PNG при помощи графического

редактора. Затем каждое изображение тестовой среды было преобразовано в 3D модель, пригодную для использования в симуляторе Gazebo (Рисунок 6.3).

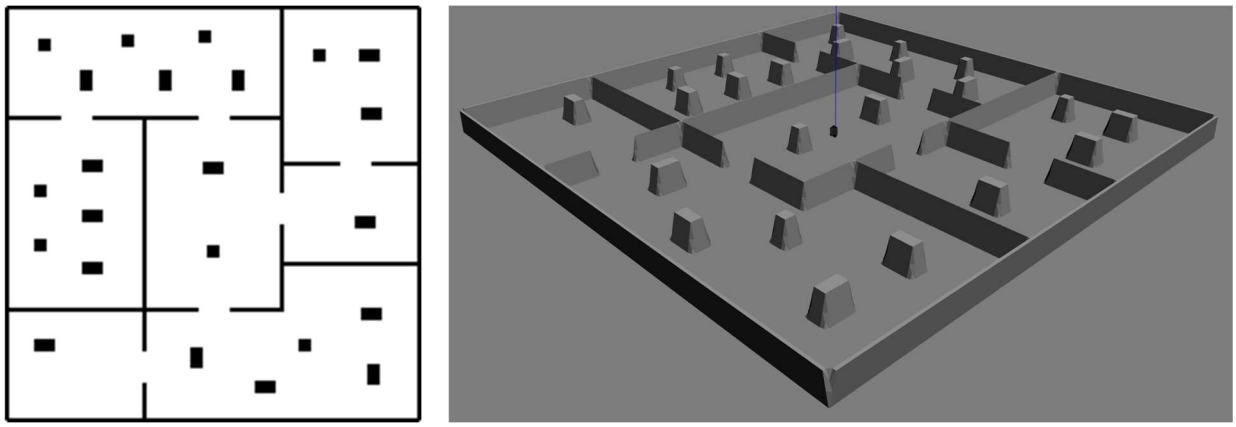

Рисунок 6.3. Слева – 2D изображение тестовой среды, справа – созданная по ней 3D модель, импортированная в симулятор Gazebo.

6.2 РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

Тестовые среды, созданные по методике, описанной в главе 6.1, были использованы для сравнения алгоритмов исследования неизвестной среды. Эксперименты проводились по регламенту, описанному в главе 6. Перед началом каждой попытки робот устанавливался в центр исследуемой тестовой среды, что обеспечивало равные расстояния от начальной позиции робота до границ исследуемой среды. Это позволяло гарантировать, что роботу придется перемещаться в различных направлениях для того, чтобы исследовать тестовую среду полностью. После завершения экспериментов были составлены таблицы с результатами работы алгоритмов в лабиринтах (Таблица 6.1) и в помещениях офисного типа (Таблица 6.2).

Таблица 6.1. Результаты тестирования алгоритмов в лабиринтах.

№ карты	**Алгоритмы (общее пройденное расстояние, метры)**		
	Модификация E^3	**Frontier exploration**	**Жадный алгоритм**
1	94.7	**52.5**	147.2
2	70.5	**66.3**	91.5
3	77.4	**72.6**	144.3

№ карты	Алгоритмы (общее пройденное расстояние, метры)		
	Модификация E³	**Frontier exploration**	**Жадный алгоритм**
4	**101.0**	129.6	167.0
5	105.4	**103.3**	171.2
6	134.3	**110.3**	230.2
7	**169.1**	192.4	204.5
8	194.8	**170.4**	214.0
9	135.1	**121.3**	306.7
Среднее	**113.79**	**113.19**	**186.29**

Таблица 6.2. Результаты тестирования алгоритмов в помещениях офисного типа.

№ карты	Алгоритмы (общее пройденное расстояние, метры)		
	Модификация E³	**Frontier exploration**	**Жадный алгоритм**
1	**80.9**	123.6	198.9
2	**78.0**	193.8	230.3
3	134.9	**117.3**	117.8
4	**66.5**	135.3	101.7
5	**56.1**	151.7	123.0
6	**63.7**	167.1	254.5
7	**26.5**	128.8	49.9
8	135.0	132.9	**107.4**
9	**75.2**	185.6	213.0
Среднее	**79.64**	104.88	155.17

Анализ результатов сравнения алгоритмов исследования неизвестной среды показал:

Во-первых, жадный алгоритм исследования неизвестной среды в среднем требует проехать наибольшее расстояние для того, чтобы полностью исследовать тестовую среду. Такой результат был ожидаемым, поскольку жадный алгоритм не учитывает расстояние до целей и, соответственно, не пытается его минимизировать. Разница с другими алгоритмами в общем пройденном расстоянии особенно заметна в лабиринтах. Это логично, поскольку лабиринты по определению имеют длинные и запутанные системы коридоров. При этом, чем сложнее лабиринт, тем хуже результат показывает жадный алгоритм по сравнению с другими алгоритмами. В тестовых средах офисного типа у жадного алгоритма отставание в результатах от других алгоритмов не такое значительное, как в лабиринтах, потому что структура комнат и коридоров в них не такая запутанная.

Во-вторых, алгоритм Frontier exploration в целом показывает хорошие результаты при исследовании неизвестной среды. В среднем в лабиринтах общее пройденное расстояние примерно такое же, как у модифицированного алгоритма E^3, однако в офисных помещениях общее пройденное расстояние больше, чем у E^3. Хорошие результаты алгоритма Frontier exploration в лабиринтах объясняются тем, что он всегда движется к ближайшему фронтиру. Его принцип работы в лабиринтах в некоторой степени похож на принцип работы алгоритма поиска в глубину (Depth First Search (DFS) [84]). Робот под управлением алгоритма Frontier exploration старается всегда исследовать каждую ветвь лабиринта до самого конца, чтобы в дальнейшем не пришлось в неё возвращаться для исследования остатков неизвестных территорий. Одним из главных преимуществ алгоритма Frontier exploration, которое объясняет его широкую распространенность, является то, что он прост в реализации и нетребователен к вычислительным ресурсам. По этим причинам алгоритм Frontier exploration является наиболее универсальным алгоритмом, подходящим для различных вариантов применения. В случае,

когда его уровня производительности достаточно для решения задачи, нет необходимости использовать более продвинутые алгоритмы.

В-третьих, модифицированный алгоритм E^3 успешно работает в неизвестной среде с препятствиями. Он показывает хорошие результаты в лабиринтах (такие же, как алгоритм Frontier exploration) и самые лучшие среди всех трех алгоритмов результаты в тестовых средах офисного типа. В лабиринтах проблемы у модифицированного алгоритма E^3 вызывает то, что он не пытается исследовать каждую ветвь лабиринта до самого конца. Робот под управлением нашей модификации алгоритма E^3 оставляет в каждой из ветвей лабиринта маленькие кластеры неизвестных территорий, предпочитая им большие кластеры неизвестных территорий в других ветвях лабиринта. В дальнейшем ему приходится возвращаться в уже посещенные ветви лабиринта для исследования маленьких кластеров неизвестных территорий, пропущенных ранее. Однако даже возвращаясь в уже исследованные ветви лабиринта, модифицированный алгоритм E^3 всё равно показывает результат не хуже, чем Frontier exploration, который старается исследовать ветви лабиринта до самого конца при первом их посещении.

В помещениях офисного типа модифицированный алгоритм E^3 показывает наилучшие результаты. Для полного исследования неизвестной среды роботу под управлением этого алгоритма в среднем требуется проехать меньшее расстояние, затрачивая меньше энергии из бортовой батареи на работу моторов. Следовательно, модифицированный алгоритм E^3 является наиболее энергоэффективным среди всех трех протестированных алгоритмов.

7 ДАЛЬНЕЙШАЯ РАБОТА

7.1 НЕДОСТАТКИ И ОГРАНИЧЕНИЯ АЛГОРИТМА

В данной главе описаны недостатки модифицированного алгоритма E^3, выявленные в процессе тестирования, и его ограничения, которые были известны ещё до начала тестирования, поскольку они обусловлены самой структурой алгоритма E^3.

Самым значительным ограничением нашей модификации алгоритма E^3 является невозможность работы со средами, размер которых не известен до начала исследования. Это ограничение является наследием оригинального алгоритма E^3, математическая теория которого создавалась для работы с неизвестными средами фиксированного размера. Для работы алгоритма E^3 перед началом исследования неизвестной среды требуется определить точные размеры этой среды. Это необходимо, потому что матрица, в которой будут храниться значения $\hat{\psi}(x)$ также будет иметь фиксированные размеры, подобно тому, как в матрице OGM хранятся сведения о препятствиях.

Если размеры исследуемой среды были недооценены, то робот будет исследовать среду только в указанных границах. И наоборот, если размеры исследуемой среды были переоценены, то алгоритм E^3 будет ставить целевые точки за пределами среды, где их невозможно будет достигнуть. В обоих случаях робот не сможет полностью исследовать эту неизвестную среду.

В процессе анализа алгоритма был предложен вариант обойти это ограничение путем постепенного наращивания матрицы значений $\hat{\psi}(x)$ подобно устройству динамических массивов в языках программирования, однако такое решение имеет одну проблему, суть которой на примере объясняется далее.

Предположим, что неизвестная среда для исследования имеет форму квадрата. Значения $\hat{\psi}(x)$ на границах исследуемой среды должны быть равны нулю, потому что значения функции $L(x, I)$ на границах среды также должны

быть равны нулю. По правилам вычисления значения функции $L(x, I)$, вычисляется геодезическое расстояние от точки x до ближайшей уже исследованной территории или до границ среды, в зависимости от того, что окажется ближе. Такие правила вычисления $\hat{\psi}(x)$ составлены авторами оригинального алгоритма E^3 для того, чтобы робот уделял больше внимания исследованию центральных областей неизвестной среды и не застревал в углах и на границах этой среды. Если же исключить из расчета условия про границы среды, то в начале исследования максимальные значения функции $\hat{\psi}(x)$ окажутся в одном или нескольких углах исследуемой среды, потому что у них будет максимальное геодезическое расстояние от начальной позиции робота. Алгоритм E^3 расставит целевые точки по углам среды, и робот начнет исследование с углов и границ среды, что не является самой оптимальной стратегией.

Ещё одним недостатком модифицированного алгоритма E^3 является то, что он не стремится исследовать каждую ветвь лабиринта до конца. Из-за этого приходится впоследствии возвращаться снова и исследовать оставшиеся небольшие кластеры неизвестных территорий. Возможно, что этот недостаток способен исправить более продвинутый алгоритм выбора целевых точек, который будет отдавать приоритет маленьким, но находящимся близко к роботу кластерам неисследованных территорий.

Существует ещё одна проблема модифицированного алгоритма E^3, которая иногда проявляется в лабиринтах. На начальном этапе исследования среды алгоритм ничего не знает о расположенных в ней стенах, поэтому первый маршрут для исследования строится так, будто стены в исследуемой среде отсутствуют. Этот маршрут будет перестроен только по достижении первой целевой точки. В наиболее неудачном случае может получиться так, что для достижения первой целевой точки роботу придется проехать через

весь лабиринт. В итоге неизвестная среда будет исследована, однако исследование будет проведено неоптимальным способом.

7.2 ВОЗМОЖНЫЕ УЛУЧШЕНИЯ

Для улучшения модифицированного алгоритма E^3 в первую очередь следует исправить его недостатки и ограничения. Исправление уже выявленных его недостатков может серьёзно улучшить производительность в лабиринтах. Например, можно спроектировать более продвинутый алгоритм выбора целевых точек, который будет мотивировать робота исследовать ветвь лабиринта или комнату до конца, чтобы не приходилось впоследствии в неё возвращаться. Это позволит значительно сократить расстояние, которое требуется проехать роботу до полного исследования неизвестной среды.

Кроме того, можно добавить эвристику, которая будет определять, что спланированный ранее маршрут уже не является самым оптимальным. В таком случае станет возможно спланировать новый маршрут при превышении этой метрикой некоторого порогового значения, а не проехав в худшем случае через весь лабиринт до первой целевой точки. Это также увеличит производительность алгоритма в лабиринтах.

Также необходимо доработать математическую теорию алгоритма E^3 с целью того, чтобы он мог исследовать среды, размер которых заранее неизвестен. Это улучшение будет полезным, потому что на данный момент практическая применимость модифицированного алгоритма E^3 достаточно сильно ограничена. Стратегия наращивания матрицы значений $\hat{\psi}(x)$ по образцу работы динамических массивов выглядит самой перспективной для решения этой задачи. Однако необходимо установить, какие именно действия необходимо выполнить до и после процедуры наращивания матрицы.

После исправления всех ограничений и недостатков алгоритма E^3 можно перейти к расширению его возможностей. Здесь перспективной выглядит задача обобщения алгоритма E^3 для исследования неизвестной среды силами

нескольких мобильных роботов. В данной задаче имеется сложная проблема решения задачи коммивояжера для нескольких агентов [85]. К этой проблеме можно привести следующую аналогию: если задача коммивояжера – это полный перебор всех возможных вариантов маршрутов, то задача коммивояжера для нескольких агентов это полный перебор всех полных переборов всех возможных вариантов маршрутов [86]. Точное решение этой задачи скорее всего будет невозможно получить в реальном времени даже для малого числа целевых точек и роботов. Возможно, что для успешной работы алгоритма будет достаточно эвристического решения.

Кроме того, алгоритм E^3 способен работать в трехмерном пространстве, поэтому можно реализовать алгоритм для квадрокоптера. Такая реализация будет полезна при исследовании системы пещер или подземелий, которые помимо расположения в горизонтальном пространстве, могут иметь перепады высот. В рамках этой задачи необходимо решить, как хранить и пересчитывать значения $\hat{\psi}(x)$ для трехмерной среды.

Помимо улучшений алгоритма, следует провести дополнительное тестирование. Это может быть тестирование на реальном роботе и/или в динамических окружениях. Например, было бы полезно протестировать функцию снижения уверенности алгоритма в оценке количества информации $G_t(x)$, которая имеет коэффициент τ. В процессе тестирования можно определить оптимальный коэффициент τ для сред с различным уровнем динамичности.

ЗАКЛЮЧЕНИЕ

Во время обзора литературы, проведенного в рамках данной работы, был изучен алгоритм E^3, который использует новый перспективный подход к исследованию неизвестных сред. Самый существенный недостаток алгоритма E^3 заключается в том, что он не способен работать в средах с препятствиями, поэтому не подходит для реального применения.

В настоящей работе алгоритм E^3 был значительно модифицирован путем разработки новой теоретической концепции, которая позволяет алгоритму успешно работать в среде, где может находиться произвольное количество препятствий произвольной формы. Также были выявлены и исправлены несколько недостатков оригинального алгоритма E^3 в процедурах планирования пути и выбора целей. Эти недостатки были вызваны тем, что авторы алгоритма не учитывали особенности сред с препятствиями. В настоящей работе процедуры планирования пути и выбора целей были доработаны с учетом работы в средах с препятствиями

Модифицированный алгоритм E^3 был реализован с использованием фреймворка ROS. Данная реализация является единственной реализацией алгоритма E^3 с использованием ROS. Эта реализация использовалась в экспериментальном сравнении энергоэффективности алгоритмов исследования неизвестной среды, проведенном в робототехническом симуляторе Gazebo.

Сравнение модифицированного алгоритма E^3 с жадным алгоритмом исследования и с алгоритмом Frontier exploration показало, что модифицированный алгоритм E^3 в среднем является самым энергоэффективным среди всех трех алгоритмов. Он показал наилучший результат в тестовых средах офисного типа, которые были спроектированы таким образом, чтобы быть приближенными к реальным средам из области возможной практической эксплуатации алгоритма. В лабиринтах

модифицированный алгоритм E^3 не смог показать результатов, которые бы значительно превосходили результаты алгоритма Frontier exploration. Результаты тестовых запусков проанализированы, в главе 7.2 описаны найденные недостатки алгоритма, возможные варианты их исправления и перспективные направления дальнейшего развития алгоритма.

Модифицированный алгоритм E^3 был представлен в научной статье [87] и в докладе на международной конференции «The 13th Asian Control Conference (ASCC 2022)». Исходный код модифицированного алгоритма E^3 опубликован в репозитории: https://gitlab.com/LIRS_Projects/modified-e3-exploration.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Hockstein N. G. et al. A history of robots: from science fiction to surgical robotics //Journal of robotic surgery. – 2007. – Vol. 1. – №. 2. – P. 113-118.
2. Global industrial robotics market revenue 2018-2028 [Электронный ресурс] / Global industrial robotics market revenue 2018-2028 - Statista — Режим доступа — URL: https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/ (дата обращения: 01.06.2022).
3. Mobile robotics market size worldwide 2019-2030 [Электронный ресурс] / Mobile robotics market size worldwide 2019-2030 - Statista — Режим доступа — URL: https://www.statista.com/statistics/1255116/global-mobile-robotics-market-size-forecast/ (дата обращения: 01.06.2022).
4. Khazetdinov A. et al. RFID-based warehouse management system prototyping using a heterogeneous team of robots //Robots in Human Life. – 2020. – P. 263.
5. Drew D. S. Multi-agent systems for search and rescue applications //Current Robotics Reports. – 2021. – Vol. 2. – №. 2. – P. 189-200.
6. Kolpashchikov D., Gerget O., Meshcheryakov R. Robotics in Healthcare //Handbook of Artificial Intelligence in Healthcare. – Springer, Cham, 2022. – P. 281-306.
7. Patil D. et al. A survey on autonomous military service robot //2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). – IEEE, 2020. – P. 1-7.
8. Ryumin D. et al. A multimodal user interface for an assistive robotic shopping cart //Electronics. – 2020. – Vol. 9. – №. 12. – P. 2093.
9. Duffner F. et al. Post-lithium-ion battery cell production and its compatibility with lithium-ion cell production infrastructure //Nature Energy. – 2021. – Vol. 6. – №. 2. – P. 123-134.

10. Lipu M. S. H. et al. Intelligent algorithms and control strategies for battery management system in electric vehicles: Progress, challenges and future outlook //Journal of Cleaner Production. – 2021. – Vol. 292. – P. 126044.
11. Emadi A. Energy-Efficient Electric Motors, Revised and Expanded. – CRC Press, 2018.
12. Mirzaeinia A., Hassanalian M., Lee K. Drones for borders surveillance: autonomous battery maintenance station and replacement for multirotor drones //AIAA Scitech 2020 Forum. – 2020. – P. 0062.
13. Niroui F. et al. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments //IEEE Robotics and Automation Letters. – 2019. – Vol. 4. – №. 2. – P. 610-617.
14. Ardiny H., Witwicki S., Mondada F. Autonomous exploration for radioactive hotspots localization taking account of sensor limitations //Sensors. – 2019. – Vol. 19. – №. 2. – P. 292.
15. O'Flaherty R., Egerstedt M. Optimal exploration in unknown environments //2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – IEEE, 2015. – P. 5796-5801.
16. Turtlebot 3 [Электронный ресурс] / Turtlebot 3 Burger — Режим доступа — URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (дата обращения: 01.06.2022)
17. Yamauchi B. A frontier-based approach for autonomous exploration //Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'. – IEEE, 1997. – P. 146-151.
18. Quin P. et al. Approaches for efficiently detecting frontier cells in robotics exploration //Frontiers in Robotics and AI. – 2021. – Vol. 8. – P. 1.
19. 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. – Vol. 3. – P. 2149-2154.

20. Sombolestan S. M., Rasooli A., Khodaygan S. Optimal path-planning for mobile robots to find a hidden target in an unknown environment based on machine learning //Journal of Ambient Intelligence and Humanized Computing. – 2019. – Vol. 10. – №. 5. – P. 1841-1850.
21. Tovar B., Murrieta-Cid R., LaValle S. M. Distance-optimal navigation in an unknown environment without sensing distances //IEEE Transactions on Robotics. – 2007. – Vol. 23. – №. 3. – P. 506-518.
22. Li H., Zhang Q., Zhao D. Deep reinforcement learning-based automatic exploration for navigation in unknown environment //IEEE transactions on neural networks and learning systems. – 2019. – Vol. 31. – №. 6. – P. 2064-2076.
23. Khaksar W. et al. A review on mobile robots motion path planning in unknown environments //2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS). – IEEE, 2015. – P. 295-300.
24. O'Flaherty R., Egerstedt M. Optimal exploration in unknown environments //2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – IEEE, 2015. – P. 5796-5801.
25. O'Flaherty R. W. A control theoretic perspective on learning in robotics : дис. – Georgia Institute of Technology, 2015.
26. Kabir R. H., Lee K. Receding-horizon ergodic exploration planning using optimal transport theory //2020 American Control Conference (ACC). – IEEE, 2020. – P. 1447-1452.
27. Kabir R. H., Lee K. Efficient, decentralized, and collaborative multi-robot exploration using optimal transport theory //2021 American Control Conference (ACC). – IEEE, 2021. – P. 4203-4208.

28. Zakiev A. et al. Partially unknown environment exploration algorithm for a mobile robot //Journal of Adv. Research in Dynamical & Control Systems. – 2019. – Vol. 11. – №. 08.
29. Quigley M. et al. ROS: an open-source Robot Operating System //ICRA workshop on open source software. – 2009. – Vol. 3. – №. 3.2. – P. 5.
30. Белов Н. В., Буянов Б. Я. РЕАЛИЗАЦИЯ АЛГОРИТМА SLAM С ИСПОЛЬЗОВАНИЕМ MICROSOFT KINECT //ТЕХНОЛОГИИ ИНФОРМАЦИОННОГО ОБЩЕСТВА. – 2019. – P. 393-395.
31. Dooraki A. R., Lee D. J. Memory-based reinforcement learning algorithm for autonomous exploration in unknown environment //International Journal of Advanced Robotic Systems. – 2018. – Vol. 15. – №. 3. – P. 1729881418775849.
32. Hou L., Zhang L., Kim J. Energy modeling and power measurement for mobile robots //Energies. – 2018. – Vol. 12. – №. 1. – P. 27.
33. 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.
34. Van Nam D., Gon-Woo K. Solid-state LiDAR based-SLAM: A concise review and application //2021 IEEE International Conference on Big Data and Smart Computing (BigComp). – IEEE, 2021. – P. 302-305.
35. Tang H. et al. A novel hybrid algorithm based on PSO and FOA for target searching in unknown environments //Applied Intelligence. – 2019. – Т. 49. – №. 7. – С. 2603-2622.
36. Tang H. et al. A multirobot target searching method based on bat algorithm in unknown environments //Expert Systems with Applications. – 2020. – Т. 141. – С. 112945.
37. Ghute M. S., Kamble K. P., Korde M. Design of military surveillance robot //2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC). – IEEE, 2018. – С. 270-272.

38. Nayyar A. et al. Smart surveillance robot for real-time monitoring and control system in environment and industrial applications //Information systems design and intelligent applications. – Springer, Singapore, 2018. – С. 229-243.
39. Azeta J. et al. An android based mobile robot for monitoring and surveillance //Procedia Manufacturing. – 2019. – Т. 35. – С. 1129-1134.
40. Перышкин А. В., Гутник Е. М. Физика. 8 класс //М.: Дрофа. – 2019. – Vol. 200. – №. 9.
41. Pradhan S. K., Chakraborty B. Battery management strategies: An essential review for battery state of health monitoring techniques //Journal of Energy Storage. – 2022. – Vol. 51. – P. 104427.
42. Rao M. V. S., Shivakumar M. Overview of Battery Monitoring and Recharging of Autonomous Mobile Robot [J] //International Journal on Recent and Innovation Trends in Computing and Communication. – 2018. – Vol. 6. – №. 5. – P. 174-179.
43. Stanford Artificial Intelligence Laboratory [Электронный ресурс] / Stanford Artificial Intelligence Laboratory — Режим доступа — URL: https://ai.stanford.edu/ (дата обращения: 01.06.2022).
44. Gmapping [Электронный ресурс] / Gmapping – ROS Wiki — Режим доступа — URL: http://wiki.ros.org/gmapping (дата обращения: 01.06.2022).
45. Navfn [Электронный ресурс] / Navfn – ROS Wiki — Режим доступа — URL: http://wiki.ros.org/navfn (дата обращения: 01.06.2022).
46. DWA Local Planner [Электронный ресурс] / DWA Local Planner - ROS Wiki — Режим доступа — URL: http://wiki.ros.org/dwa_local_planner (дата обращения: 01.06.2022).
47. Choi H. S. et al. On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward //Proceedings of the National Academy of Sciences. – 2021. – Vol. 118. – №. 1. – P. e1907856118.

48. University of Southern California [Электронный ресурс] / University of Southern California — Режим доступа — URL: https://www.usc.edu/ (дата обращения: 01.06.2022).
49. Willow Garage [Электронный ресурс] / Willow Garage — Режим доступа — URL: https://www.willowgarage.com/ (дата обращения: 01.06.2022).
50. Open Robotics [Электронный ресурс] / Open Robotics — Режим доступа — URL: https://www.openrobotics.org/ (дата обращения: 01.06.2022).
51. ODE [Электронный ресурс] / Open Dynamics Engine — Режим доступа — URL: https://www.ode.org/ (дата обращения: 01.06.2022).
52. Bullet [Электронный ресурс] / Bullet Real-Time Physics Simulation — Режим доступа — URL: https://pybullet.org/ (дата обращения: 01.06.2022).
53. Dynamic Animation and Robotics Toolkit [Электронный ресурс] / DART — Режим доступа — URL: https://dartsim.github.io/ (дата обращения: 01.06.2022).
54. OGRE [Электронный ресурс] / OGRE — Режим доступа — URL: https://www.ogre3d.org/ (дата обращения: 01.06.2022).
55. DARPA [Электронный ресурс] / Defense Advanced Research Projects Agency — Режим доступа — URL: https://www.darpa.mil/ (дата обращения: 01.06.2022).
56. NASA [Электронный ресурс] / NASA — Режим доступа — URL: https://www.nasa.gov/ (дата обращения: 01.06.2022).
57. Move_base [Электронный ресурс] / Move_base - ROS Wiki — Режим доступа — URL: http://wiki.ros.org/move_base (дата обращения: 01.06.2022).
58. ROBOTIS [Электронный ресурс] / ROBOTIS — Режим доступа — URL: https://www.robotis.us/ (дата обращения: 01.06.2022).

59. Naval Research Laboratory [Электронный ресурс] / The U.S. Naval Research Laboratory — Режим доступа — URL: https://www.nrl.navy.mil/ (дата обращения: 01.06.2022).
60. Nomad 200 [Электронный ресурс] / The Nomad 200 Robot — Режим доступа — URL: http://ksl-web.stanford.edu/projects/aibots/Nomad-200.html (дата обращения: 01.06.2022).
61. Elfes A. Using occupancy grids for mobile robot perception and navigation //Computer. – 1989. – Vol. 22. – №. 6. – P. 46-57.
62. Topiwala A., Inani P., Kathpal A. Frontier based exploration for autonomous robot //arXiv preprint arXiv:1806.03581. – 2018.
63. Rahim R. et al. Breadth first search approach for shortest path solution in Cartesian area //Journal of Physics: Conference Series. – IOP Publishing, 2018. – Т. 1019. – №. 1. – С. 012038.
64. Quin P. et al. Expanding wavefront frontier detection: An approach for efficiently detecting frontier cells //Australasian Conference on Robotics and Automation, ACRA. – 2014.
65. Gao W. et al. An improved frontier-based approach for autonomous exploration //2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV). – IEEE, 2018. – P. 292-297.
66. Hassan M., Liu D., Paul G. Collaboration of multiple autonomous industrial robots through optimal base placements //Journal of Intelligent & Robotic Systems. – 2018. – Vol. 90. – №. 1. – P. 113-132.
67. Frontier_exploration [Электронный ресурс] / Frontier_exploration - ROS Wiki — Режим доступа — URL: http://wiki.ros.org/frontier_exploration (дата обращения: 01.06.2022).
68. Explore_lite [Электронный ресурс] / Explore_lite - ROS Wiki — Режим доступа — URL: http://wiki.ros.org/explore_lite (дата обращения: 01.06.2022).

69. Koopman P. Bresenham line-drawing algorithm //Forth Dimensions. – 1987. – Vol. 8. – №. 6. – P. 12-16.
70. Miller L. M. et al. Ergodic exploration of distributed information //IEEE Transactions on Robotics. – 2015. – Vol. 32. – №. 1. – P. 36-52.
71. NxR Lab [Электронный ресурс] / NxR Lab – SIRA - Shah Institute of Robotics and Automation — Режим доступа — URL: https://sirainstitute.com/nxr-lab-2/ (дата обращения: 01.06.2022).
72. Northwestern University [Электронный ресурс] / Northwestern University — Режим доступа — URL: https://www.northwestern.edu/ (дата обращения: 01.06.2022).
73. Yang J. An easily implemented, block-based fast marching method with superior sequential and parallel performance //SIAM Journal on Scientific Computing. – 2019. – Vol. 41. – №. 5. – P. C446-C478.
74. Kornilov A. S., Safonov I. V. An overview of watershed algorithm implementations in open source libraries //Journal of Imaging. – 2018. – Vol. 4. – №. 10. – P. 123.
75. Николаева Д. С., Копылова Е. С., Бунтова Е. В. Решение задачи коммивояжёра с использованием метода ветвей и границ //Human Progress. – 2018. – Т. 4. – №. 4. – С. 4.
76. Kotas P. et al. A massive parallel fast marching method //Domain Decomposition Methods in Science and Engineering XXII. – Springer, Cham, 2016. – P. 311-318.
77. Fast Marching Method [Электронный ресурс] / The Fast Marching Method – GitHub — Режим доступа — URL: https://github.com/thinks/fast-marching-method (дата обращения: 01.06.2022).
78. Villacarlos R. L. et al. A Tale of Two Trees: New Analysis for AVL Tree and Binary Heap //arXiv preprint arXiv:2010.04752. – 2020.
79. Булыга Ф. С., Курейчик В. М. МЕТОД ПОНИЖЕНИЯ ШУМА НА ЦИФРОВЫХ ИЗОБРАЖЕНИЯХ. – 2021.

80. OpenCV [Электронный ресурс] / OpenCV: Home — Режим доступа — URL: https://opencv.org/ (дата обращения: 01.06.2022).
81. RViz [Электронный ресурс] / RViz - ROS Wiki — Режим доступа — URL: http://wiki.ros.org/rviz (дата обращения: 01.06.2022).
82. Jamil N., Sembok T. M. T., Bakar Z. A. Noise removal and enhancement of binary images using morphological operations //2008 International Symposium on Information Technology. – IEEE, 2008. – Vol. 4. – P. 1-6.
83. Puzzlemaker [Электронный ресурс] / Puzzlemaker - Discovery Education — Режим доступа — URL: https://puzzlemaker.discoveryeducation.com/ (дата обращения: 01.06.2022).
84. Pereira R. F. et al. Iterative Depth-First Search for Fully Observable Non-Deterministic Planning //arXiv preprint arXiv:2204.04322. – 2022.
85. Bektas T. The multiple traveling salesman problem: an overview of formulations and solution procedures //omega. – 2006. – Vol. 34. – №. 3. – P. 209-219.
86. Cheikhrouhou O., Khoufi I. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy //Computer Science Review. – 2021. – Vol. 40. – P. 100369.
87. Mavrin I. et al. Modified E^3 exploration algorithm for unknown environments with obstacles //2022 13th Asian Control Conference (ASCC). – IEEE, 2022. – P. 1413-1418.