Федеральное государственное автономное
образовательное учреждение высшего образования
«Санкт-Петербургский государственный электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)»

На правах рукописи

Филатов Антон Юрьевич

МАСШТАБИРУЕМЫЕ АЛГОРИТМЫ ОДНОВРЕМЕННОГО ПОСТРОЕНИЯ КАРТЫ И ЛОКАЛИЗАЦИИ СТАИ МОБИЛЬНЫХ РОБОТОВ

05.13.11. – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
к.т.н., доцент Кринкин Кирилл Владимирович

Санкт-Петербург
2021 г.

Оглавление

Введение

По данным исследовательской компании Gartner [52] в 2019 году произведено и введено в эксплуатацию более 300 тыс. беспилотных автомобилей. По прогнозам аналитиков Gartner к 2023 году это число возрастёт до 745 тыс. единиц. Такая статистика говорит о безоговорочном росте индустрии автопилотируемых транспортных средств. Выручка одного из гигантов этой области – компании TeslaMotors – в 2019 году составила $24.58 млрд [53]. Таким образом, можно уверенно говорить о коммерческом интересе к области производства беспилотных автомобилей. Следовательно, любое научное или инженерное изобретение в данной области является интересным и востребованным.

Одна из задач, которые ставятся перед беспилотным автомобилем – ориентация на местности. От того, насколько точна карта местности, построенная в памяти бортового компьютера и согласно которой движется транспортное средство, зависит, насколько аккуратно и точно будет двигаться автомобиль. Немаловажную роль играет определение его собственного положения на этой карте. Определять собственное положение необходимо с точностью до сантиметров, ни одна спутниковая система локализации на это не способна.

Для обеспечения точной локализации применяются алгоритмы, решающие задачу SLAM (англ. Simultaneous Localization And Mapping) – одновременного определения собственного положения и построения карты. Строить карту согласно работы алгоритма вместо использования заготовленной заранее имеет смысл в том случае, если известно, что местность может изменяться по сравнению с тем, что было запечатлено заранее. Например, автомобиль, который движется по дорогам общего пользования, должен избегать ям на дорогах или объезжать временно перекрытые участка дороги. Снабдить беспилотный автомобиль такой информацией заранее является сложной задачей. Поэтому использовать заготовленную карту, безусловно, полезно, но необходимо также обновлять эту карту по мере движения.

В качестве другого примера использования алгоритма, решающего задачу SLAM, приведем задачу построения плана местности на Луне или на Марсе. Марсоход не может определить собственное положение, используя спутниковую локализацию, поэтому задача локализации на неизвестной карте стоит наиболее остро. Такая же задача может возникнуть в более «приземленных» условиях. Например, автономный робот-спасатель, который должен проникнуть в разрушающееся здание в поисках нуждающихся в спасении людей, не может рассчитывать на использование только плана здания – ему нужно получать информацию о проходимости коридоров и комнат прямо по ходу его движения.

Использование нескольких агентов, работающих сообща и вместе строящих карту местности, позволяет ускорить построение карты и увеличить точность локализации каждого робота в отдельности. Ускорение работы достигается за счет того, что исследуемая область может быть поделена на участки, каждый из которых исследует только один или несколько агентов, а затем все изученные участки карты объединяются. Увеличение точности по сравнению с одноагентным алгоритмом достигается за счет того, что каждый агент не только дополняет, но и верифицирует карту других агентов. Это позволяет вовремя исправлять возможные ошибки, которые возникают по ходу вычисления собственного положения и построения карты.

Целью данной работы является разработка масштабируемых алгоритмов для многоагентного решения задачи SLAM, применимых для роботов с ограниченными вычислительными ресурсами. В контексте данной работы роботом является передвижная платформа, оснащенная вычислительным устройством, а также лидаром – датчиком, который может измерять расстояние до препятствий, окружающих робота. Ограничение ресурсов распространяется в первую очередь на вычислительное устройство, анализ применимости различных лидаров в данной работе не проводится. Робот с ограниченными вычислительными ресурсами – это робот, имеющий ограниченные вычислительную мощность и объём оперативной памяти, небольшие размеры и питание от портативной батареи. На текущий момент типичными

представителями таких устройств являются продукты Raspberry Pi или Jetson Nano.

В рамках поставленной цели, мобильный робот, решающий задачу SLAM, является автономным; единственным источником данных об окружающей среде является лидар.

Во время выполнения данной работы были **поставлены и решены следующие задачи**:

1 Классификация и сравнительный анализ существующих одноагентных и многоагентных алгоритмов, решающих задачу SLAM.

2 Разработка масштабируемого многоагентного алгоритма SLAM на базе теории Демпстера-Шафера.

3 Разработка алгоритма фильтрации двумерных лазерных сканов для освобождения вычислительных ресурсов.

4 Программная реализация масштабируемого многоагентного алгоритма решения задачи SLAM.

5 Оценка точности карты и траектории, построенных в результате работы многоагентного алгоритма, а также оценка производительности на вычислительных устройствах с ограниченными ресурсами.

Объектом исследования является взаимодействие стаи мобильных роботов при решении задачи одновременного построения карты и локализации.

Предметом исследования является архитектура, точность и быстродействие многоагентного алгоритма, решающего задачу одновременного определения положения и построения каты.

Методы исследования. Для решения поставленных задач использовались методы математической статистики, теории вероятностей, теории графов.

На защиту диссертационной работы выносятся следующие **положения**:

1 Масштабируемые метод и алгоритм решения многоагентной задачи SLAM на базе теории Демпстера-Шафера, которые позволяют снизить требования к вычислительным ресурсам.

2 Масштабируемый корреляционный фильтр двумерных лазерных сканов для многоагентного алгоритма SLAM, обеспечивающий снижение размерности входных данных.

3 Архитектура программного обеспечения, реализующего масштабируемый многоагентный алгоритм, решающий задачу SLAM.

Научная новизна работы заключается в следующем:

1 Предложен многоагентный горизонтально масштабируемый алгоритм решения задачи SLAM для стаи мобильных роботов на базе теории Демпстера-Шафера, строящий карту окружения. Применение этой теории позволяет увеличить точность построенной карты, а также уменьшить влияние шумов по сравнению с классической Байесовской теорией.

2 Разработан алгоритм фильтрации двумерных лазерных сканов на базе построения гистограмм и вычисления коэффициента корреляции Пирсона, позволяющий обрабатывать только часть входных данных без ущерба для точности.

3 Разработана масштабируемая архитектура компонентов многоагентного алгоритма SLAM, допускающая выход из строя любого числа агентов, кроме одного последнего.

Теоретическая значимость работы заключается в применении теории Демпстера-Шафера к модели карты занятости, которая строится в процессе работы алгоритма, решающего задачу SLAM. В работе получены закономерности, связывающие параметры алгоритма и характеристики мобильного робота и увеличивающие за счет этого точность построения карты и локализации.

Практическая значимость. Разработанные алгоритмы и программное обеспечение могут быть применены для решения задачи одновременной локализации и построения карты при одновременной работе нескольких подвижных роботов. В частности, они могут быть использованы для роботов-курьеров, сервисных и складских роботов. Поскольку разработанные методы и алгоритмы нацелены на сокращение требований к вычислительным ресурсам, то

их применение позволит понизить стоимость производства таких роботов и более эффективно использовать вычислительные ресурсы.

Апробация результатов. Результатом выполнения диссертационной работы является разработанный многоагентный алгоритм решения задачи SLAM и реализующее его программное обеспечение. Промежуточные результаты были представлены на конференциях FRUCT, проходивших с 2016 по 2019 год. Во время выполнения работы было получено 1 свидетельство о регистрации программного обеспечения (ПО) для ЭВМ [3], а также 9 публикаций в научных изданиях, среди которых 2 публикации в изданиях, входящих в перечень ВАК [1], [2], 7 публикаций в изданиях, индексируемых в базе данных Scopus [18], [19], [20], [31], [32], [33], [34], среди которых одна работа в журнале Q2 и одна – в журнале Q1.

Достоверность и обоснованность представленных в диссертационной работе научных выводов подтверждается согласованностью теоретических и практических результатов, корректностью доказательств, а также апробацией основных теоретических положений в научных статьях и выступлениях на научных конференциях. Достоверность результатов диссертационной работы также подтверждается проведенными экспериментами с использованием программной реализации многоагентного масштабируемого алгоритма одновременной локализации и построения карты стаи роботов.

Внедрение результатов работы. Научные результаты, полученные в результате работы над диссертацией были внедрены в учебный процесс СПБГЭТУ «ЛЭТИ» им. В. И. Ульянова (Ленина).

Личный вклад автора. Все результаты, изложенные в диссертации и сформулированные в положениях, выносимых на защиту, получены автором лично или при его непосредственном участии.

Структура и объём диссертационной работы. Диссертационная работа состоит из введения, пяти глав, заключения и списка литературы из 55 наименований. Объём работы 111 машинописных страниц, она содержит 39 рисунков, 3 таблицы.

1 Постановка задачи slam и обзор ее решений

Прежде, чем говорить о многоагентных системах необходимо классифицировать существующие одноагентные подходы и определить, возможно ли масштабирование этих подходов и их стоимость.

Задача SLAM – это задача, которая ставится перед движущимися роботами, которым необходимо исследовать неизвестное окружение и построить карту местности. Необходимо отметить, что цели построить маршрут, позволяющий исследовать окружение, не ставится. Для решения задачи достаточно, чтобы робот двигался по какому-либо маршруту и имел возможность снимать измерения окружения.

Следует отметить, что задача SLAM состоит из двух задач: построение карты с условием, что положение агента и состояние окружающей среды известно в каждый момент времени; и локализация, то есть определение местоположения, если известна карта. Из формулировки этих подзадач понятно, что для успешного решения одной задачи необходимо, чтобы сначала была решена другая задача. Именно поэтому краеугольным камнем становится их одновременное решение. Поиск робастного алгоритма ведется уже больше двадцати лет [37].

Данная задача актуальна для мобильных агентов в абсолютно разных сферах жизни: начиная с роботов-пылесосов и заканчивая автопилотируемыми автомобилями, роботами-спасателями и марсоходами.

Кроме того, необходимо учитывать, что автономные мобильные агенты не обладают пока выдающимися вычислительными мощностями. Это накладывает ограничение на уровень сложности алгоритмов решения поставленной задачи. Также существует класс сложных алгоритмов, которые должны выполняться на серверах, в то время как мобильные агенты выполняют лишь роль наблюдателей за окружением. В последнем случае необходимо следить за качеством связи между всеми узлами системы, что приводит к снижению устойчивости и автономности системы.

Дополнительную сложность в рассматриваемую задачу вносит тот факт, что часто нет возможности воспользоваться такими сильными инструментами, как GPS. Например, это невозможно в помещении, где погрешность GPS сопоставима с шириной коридора. Мобильный агент может полагаться только на датчики камеры, лазерного дальномера, гироскопа или сонара.

Данная глава посвящена обзору современных и классических решений задачи SLAM. Основные тезисы из данной главы описаны в работах [1], [2].

1.1 Классификация алгоритмов решения задачи SLAM

На данный момент, исходя из разных начальных условий, существуют различные подходы к решению задачи SLAM. Можно провести классификацию решений по различным критериям [33]. Справедливо заметить, что нет прямой связи между какими-то определенными типами из одной и из другой классификации. Однако, среди наиболее популярных алгоритмов прослеживается закономерность в реализации определенных комбинаций типов из разных классификаций.

По размерности наблюдений.

Трехмерные.

Если оснастить агента видеокамерой, то он полностью наблюдает окружающее трехмерное пространство, и построенная им карта также будет иметь размерность, равную трем [13], [20], [29], [42]. Такие алгоритмы обычно являются достаточно требовательными к вычислительным ресурсам, поскольку их операционные входные данные являются облаками точек, полученных либо при помощи камеры или комбинации камер, либо с помощью трехмерного лидара. Пример такого облака точек представлен на рисунке 1.1.

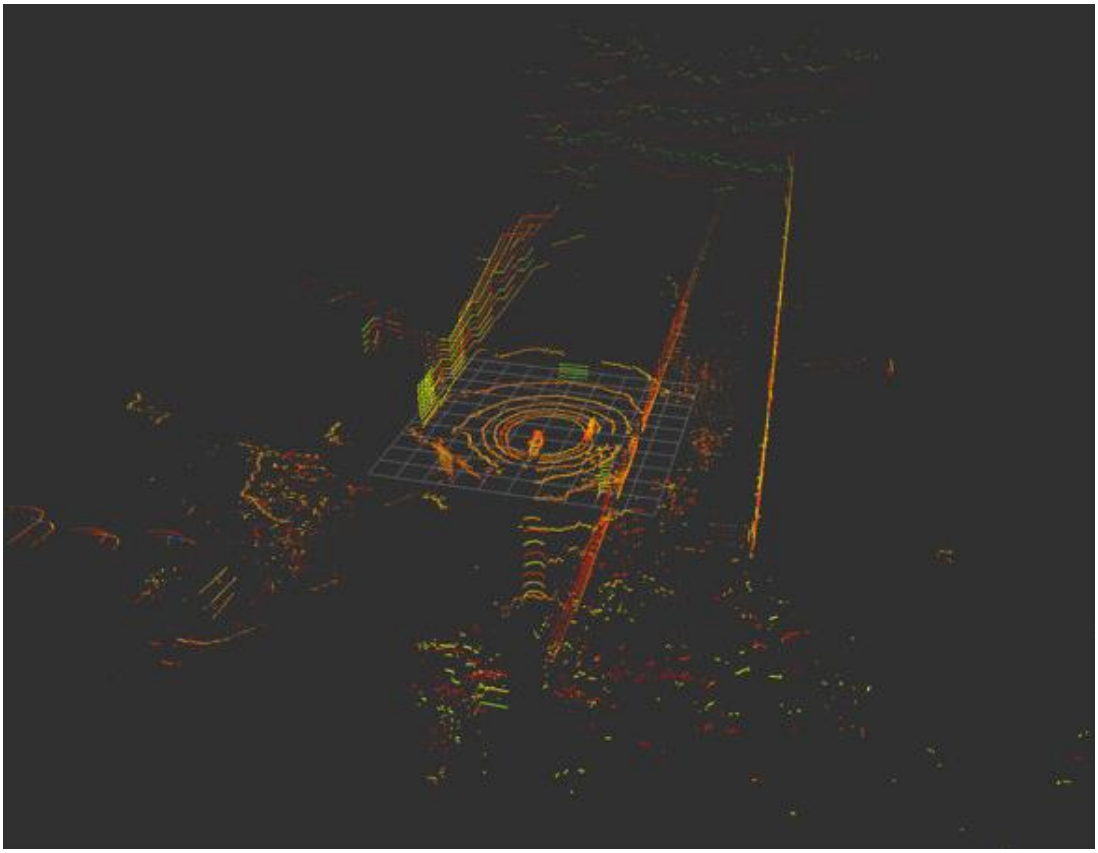


Рисунок 1.1 – Пример облака точек, полученного с трехмерного лидара

Двумерные.

Часто для упрощения обработки входных данных или по иным причинам используются лазерные дальномеры или сонары, которые проводят измерения в плоскости [15], [24], [27]. Построенная при использовании таких датчиков карта представляет собой план окружения. Двумерные алгоритмы являются более быстрыми, чем трехмерные в силу меньшего количества входных данных. Их достоинством является тот факт, что они могут обрабатывать данные лидаров, обладающих очень высоким разрешением и, следовательно, строить точный план карты, имея низкую погрешность при построении траектории движения.

По типу используемых сенсоров.

Обычно этот способ классификации напрямую связан с предыдущим, однако, есть и некоторые исключения.

Визуальный.

В качестве сенсоров используется видеокамера [20], [42]. Это может быть, как одна камера, так и набор из нескольких, а также «всенаправленная» камера, которая снимает за один раз окружение на 360°.

Лазерный.

В качестве сенсоров используется лазерный дальномер – устройство, которое измеряет расстояние во всех направлениях с шагом дискретизации 0.5-1° [15], [24].

Обычно визуальный алгоритм строит трехмерную карту, а лазерный – двумерную, но эта зависимость не является обязательной. Так, например, возможно строить трехмерную карту при помощи лазерного лидара [13] или двумерную карту, используя видео камеру. Классические алгоритмы, стоящие у истоков развития задачи SLAM работали именно так: по набору видеоизображений строилась двумерная карта препятствий.

По способу обработки входных измерений.

Основанный на выделении особых точек.

Часто нет необходимости строить полную карту местности, достаточно ограничится некоторым набором ориентиров [6], [40], [51]. Такие ориентиры должны быть выделены из данных сенсоров и использованы для построения карты. Исторически именно такой тип алгоритмов был разработан первым.

Использующий «сырые» измерения.

Гораздо более полную карту можно получить, если использовать все данные, полученные от сенсоров [13], [15], [24], [29]. Если видеокадр либо лазерный скан использовать полностью, то информации для построения карты становится больше, и карта становится намного точнее.

По способу представления карты.

Использующий сетку занятости.

Карта в алгоритме SLAM – это один из результатов работы алгоритма, поэтому вопрос о ее структуре стоит наиболее остро. Очевидным выглядит решение представить карту в виде массива ячеек (двумерного или трехмерного), где каждая ячейка содержит вероятность быть занятой [15], [27]. Тогда все

пространство можно разделить на небольшие области, каждая из которых либо занята каким-то препятствием, либо свободна.

Графовый.

В современных подходах все чаще используют другой способ представления карты. Графовый подход предполагает, что карта распределена по узлам графа [23], [24], [54]. В каждом узле находится одно или несколько измерений. Ребра графа содержат в себе информацию о перемещении агента, которое необходимо совершить, чтобы начать наблюдать измерения из соответствующего узла. Достоинством такого алгоритма является возможность определить, посещал ли робот место, с которого снимается текущее наблюдение или нет. Если посещал, то текущее наблюдение обязательно будет коррелировать с наблюдением из уже построенных вершин. Тогда в графе образуется цикл, инвариантом которого является сумма векторов-трансформаций между узлами графа, равная нулю (обычно на практике этот инвариант может не выполняться в силу ошибок и погрешностей вычислений). В этот момент включается механизм изменения весов ребер графа с целью достижения нуля в векторной сумме весов. Это приводит к увеличению точности построенной карты.

По характеру изменения окружающей среды.

Статический.

Такой тип алгоритмов [13], [15], [23], [29] предполагает, что окружение не изменяется со временем. Если агент полностью изучил какую-то комнату и покинул ее, а через некоторое время вернулся обратно, то он обнаружит, что ничего в этой комнате не изменилось. Такой тип алгоритмов может быть применен в офисных помещениях или складах.

Динамический.

В случае, если SLAM алгоритм применяется за пределами зданий (например, в городе, где присутствует интенсивное движение), появляется необходимость уметь корректировать карту с учетом перемещающихся препятствий [24], [41].

1.2 SLAM, основанные на выделении особых точек

Исторически первыми появились алгоритмы, которые выделяли особые точки на наблюдаемом окружении и строили карту, состоящую из таких ориентиров [6], [40], [51]. Идея очень близка к человеческому восприятию мира. Ориентируясь в незнакомом городе, человек отмечает для себя, например, высокие башни и определяет приблизительное расстояние между ними, а дальше использует их, как ориентир.

1.2.1 Общий подход к решению задачи SLAM на основе выделения особых точек.

Для автономных вычислительных агентов можно реализовать подобный алгоритм: необходимо выделить на снятых измерениях особые точки, из которых и будет в будущем состоять карта. Чем больше таких точек будет определено в процессе работы, тем точнее будут результирующая карта и позиция агента. Также на точность влияет количество позиций, с которых удалось различить особую точку. Поэтому для таких алгоритмов вопрос о робастном выделении особых точек стоит очень остро.

Работа алгоритма разделена на несколько этапов, в числе которых можно выделить **снятие измерений** (скан, кадр и пр.), **скан матчинг** [34] (сопоставление измерения и карты) и **обновление карты**. Схема этапов представлена на рисунке 1.2.

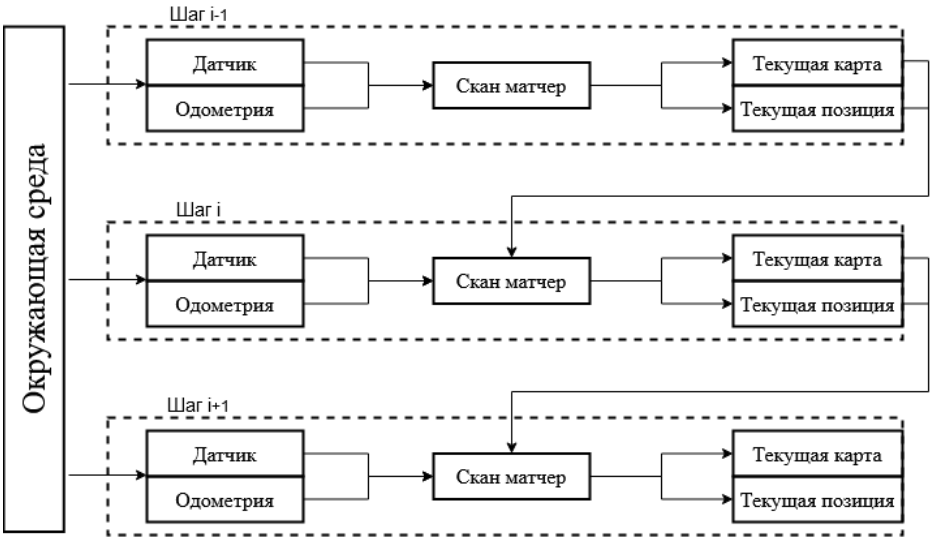


Рисунок 1.2 – Схема этапов работы алгоритма SLAM

На очередном шаге алгоритма входными данными являются: построенная к текущему моменту карта, гипотеза о положении агента, а также очередное наблюдение. Первым делом необходимо выделить особые точки или ориентиры. Затем после такой предобработки запускается компонент, который носит название **скан матчер** (англ. Scan matcher – сопоставитель сканов), задача которого сопоставить текущее наблюдение с уже построенной картой, чтобы определить текущее положение агента. После того, как позиция оценена, появляется возможность обновить карту, если это необходимо.

В рассматриваемых алгоритмах карта состоит из набора ориентиров (их координат относительно точки, где алгоритм начал свою работу). Построенные координаты ориентиров зависят от точки начала, однако расстояние и взаимное расположение между ними остается неизменным. Именно этот принцип лежит в основе такого рода алгоритмов. Если составить матрицу ковариаций ориентиров, то появится возможность применить алгоритм фильтрации Калмана для обновления карты [10].

1.2.2 EKF SLAM

Первая идея решения задачи SLAM, предложенная в 1990 году [51], опиралась на расширенный фильтр Калмана. Идея в том, что все препятствия должны коррелировать между собой, то есть (если агент работает в неподвижном окружении) взаимное расположение препятствий не должно меняться. А это значит, что, пронаблюдав пару ориентиров под разными углами, можно повысить точность оценки их расположения.

Более подробно алгоритм выглядит следующим образом. Алгоритм состоит из множества итераций, каждая из которых включает несколько шагов.

1 Агент производит очередное наблюдение, выделяет особые точки. В соответствии с текущей априорной оценкой позиции агента (исходя из расположения ориентиров относительно агента) вычисляется расположение ориентиров в глобальных координатах.

2 Координаты найденных ориентиров сопоставляются с координатами ориентиров, которые были включены в карту на предыдущих шагах. Этот шаг позволит уточнить позицию ориентиров, которые уже наблюдались ранее, либо изменить дисперсию погрешности расположения ориентиров. Если найденный ориентир отсутствует в карте, то он добавляется в карту.

3 Исходя из уточненной карты (в том числе уточненных позиций наблюдаемых ориентиров) строится апостериорная оценка позиции агента.

Необходимо заметить, что априорная оценка на шаге 1 строится, исходя из апостериорной оценки, построенной на шаге 3 на предыдущей итерации, а также исходя из данных одометрии – сведений о перемещении агента, измеренных датчиками перемещения (например, гироскопом).

Однако следует заметить, что этот алгоритм непригоден для масштабирования. Каждый новый ориентир, будучи включенным в карту, увеличивает размерность матрицы ковариаций ориентиров. Следует помнить, что карта с точки зрения рассматриваемого алгоритма – это вектор ориентиров, содержащий их координаты. Матрица ковариаций ориентиров «связывает» ориентиры между собой. Это нужно для того, чтобы в случае уточнения координат одного из препятствий можно было также уточнить координаты остальных ориентиров.

Такой подход, однако, начинает требовать больших вычислительных затрат, как только ориентиров набирается значительное количество. Помимо того, что растет вектор ориентиров, при добавлении нового элемента в вектор необходимо вычислять ковариацию нового препятствия со всеми предыдущими. Кроме того, специфика алгоритма, основанного на фильтре Калмана, требует вычислять матрицу, обратную к матрице ковариаций, что влечет за собой увеличение времени работы алгоритма с квадратичной скоростью.

Современные сенсоры позволяют снимать наблюдения окружающей среды с частотой не менее 30 Гц, следовательно, алгоритм должен успевать обрабатывать не менее 30 итераций в секунду. Именно поэтому применение EKF SLAM в современных системах практически исчезло, а ему на замену пришли

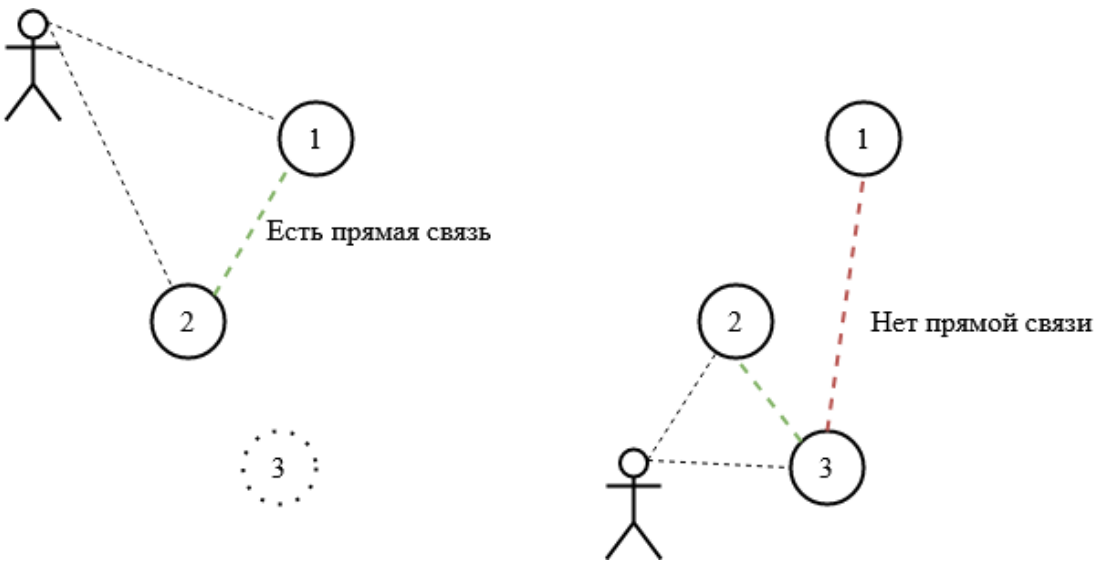
алгоритмы, основанные на более сложных идеях или допущениях, но менее требовательные к вычислительным ресурсам.

1.2.3 FastSLAM

Предложенная в [39], [40] идея продолжает концепцию EKF SLAM, но увеличивает производительность за счет отсутствия корреляции между препятствиями.

Другими словами, общий подход к алгоритму остается практически таким же, как в алгоритме EKF SLAM, но считается, что каждый ориентир не зависит от всех остальных.

Такая постановка вопроса неизбежно порождает неоднозначность карты. Разрешить ее можно единственным образом – найти точку, с которой можно будет пронаблюдать два - три объекта одновременно. Если же такая точка не найдена, остается лишь надеяться, что человек не ошибся, когда строил траекторию между точками, в которых он производил наблюдения. Графически последовательность наблюдений и возможная результирующая карта представлены на рисунке 1.3.



а) б)

Рисунок 1.3 – Процесс определения особых препятствий наблюдателем согласно алгоритму FastSLAM: а) наблюдатель видит препятствия 1 и 2, и запоминает их взаимное расположение;

б) наблюдатель видит препятствия 2 и 3, и делает то же самое, но связь между 1 и 3 напрямую не строится

Чтобы разрешить неопределенность часто прибегают к аппарату, называемому **фильтром частиц,** который будет подробно рассмотрен в следующей главе. На этапе определения положения выбирается не одна наиболее вероятная гипотеза, а несколько: например, три. В этом случае будут существовать три возможные карты, и каждое новое наблюдение будет необходимо сопоставлять с тремя картами. Процесс будет происходить до тех пор, пока какая-то из гипотез не будет отброшена, поскольку новые наблюдения перестанут каким бы то ни было образом соотноситься с картой.

Тесты, проведенные авторами в исходной статье, показывают, что точность подхода FastSLAM (несмотря на существенное пространство для неопределенности) остается высокой. И действительно, алгоритм FastSLAM – это один из базовых на сегодняшний день алгоритмов.

1.3 Байесовская теория

Уже упоминалось, что идея выделения особых точек на кадре накладывает значительные ограничения на структуру карты и на алгоритм в целом. Существует альтернативный метод представления карты, который позволяет хранить модель карты в динамически изменяющемся массиве [15], [27]. В простом случае речь идет о двумерном массиве, каждый элемент которого – это «клетка», соответствующая области пространства. Другими словами, карта представляет собой двумерный план окружающей среды, разделенный на ячейки. Каждая ячейка такой карты содержит число – вероятность быть занятой. Пример карты с ячейкой размером 10 × 10 см показан на рисунке 1.4.

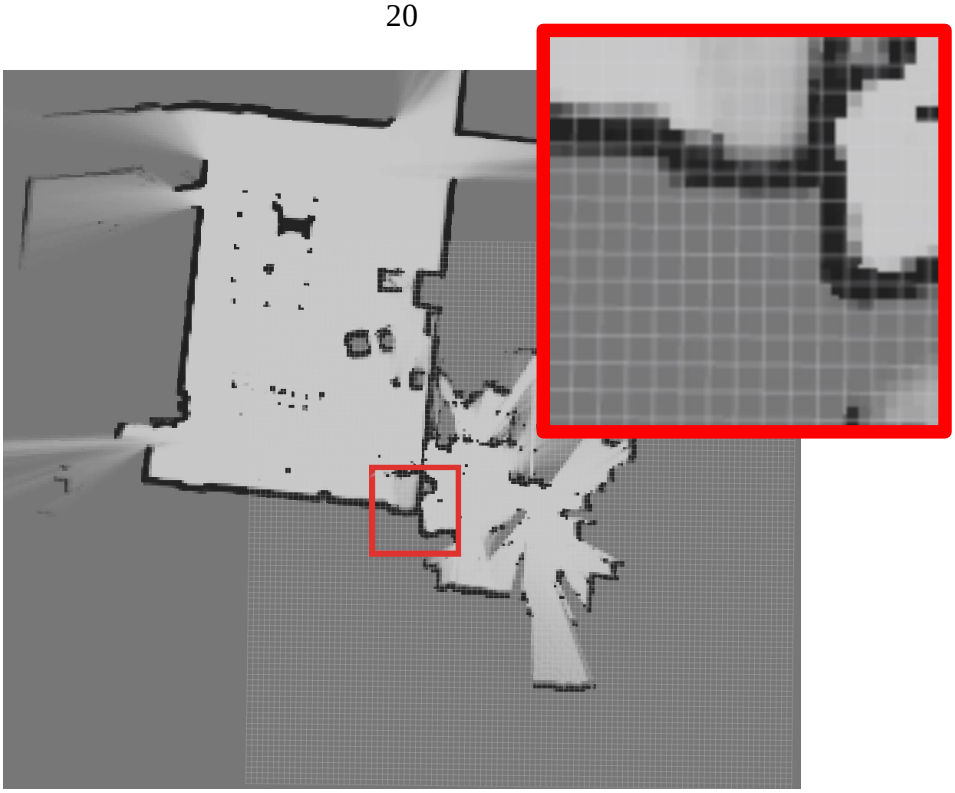


Рисунок 1.4 – Пример карты занятости

По рисунку понятно, что при достаточно маленьком размере ячеек, такая карта неотличима от обыденного плана окружения, а также содержит больше информации, чем набор ориентиров.

Лучше всего такой формат карты подходит для лазерных SLAM алгоритмов, когда в качестве источника наблюдений выступает лазерный дальномер. Если визуализировать лазерный скан, то окажется, что он имеет похожую на карту структуру, и, «накладывая» скан на карту, как показано на рисунках 1.5 и 1.6, можно определить текущую позицию агента.

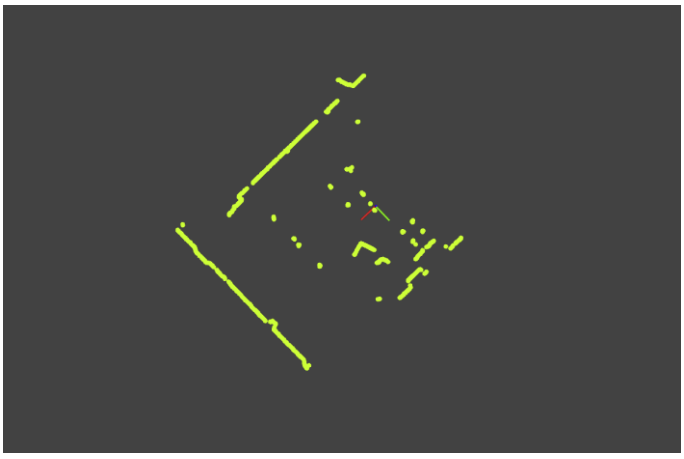


Рисунок 1.5 – Пример лазерного скана

Рисунок 1.6 – Результат работы скан матчера: сопоставленные лазерный скан и карта для вычисления текущей позиции

Для SLAM алгоритмов, которые основаны на карте, состоящей из ячеек, наибольшую популярность имеет подход к вычислению позиции агента, согласно Байесовской теории. Другими словами, основываясь на построенной карте *m*, последнем известном перемещении агента *u*, текущем наблюдении *z*, а также последней известной позиции агента *x*, можно вычислить вероятность текущего положение агента *X*, при котором возможно произвести именно такое наблюдение *p(X | m,u,z,x)* [4]. Задача алгоритма в том, чтобы вычислить *X*, вероятность которой максимальна.

Байесовская теория гласит, что поиск такой вероятности пропорционален произведению следующих вероятностей:

$$p\left(X\left|m,u,z,x\right.\right) \propto p\left(z\left|m,X\right.\right)\cdot p\left(X\left|x,u\right.\right), \tag{1.1}$$

где X – текущее положение агента,

x – предыдущая позиция агента,

m – построенная на данный момент карта,

u – перемещение агента,

z – текущее наблюдение.

Первый множитель – это вероятность наблюдать данное измерение при условии нахождения в позиции X в итеративно построенной карте m. Второй

множитель – это вероятность оказаться в позиции X, если подействовать управляющим воздействием u на предыдущую известную позицию x.

Остановимся подробнее на втором множителе. Входное воздействие – обычно это управляющий сигнал или сведения, полученные от некоторого датчика, о том, насколько агент передвинулся. Например – команда двигаться с максимальной скоростью вперед в течение одной секунды. На первый взгляд кажется, что, зная предыдущую позицию x, найти теперь новую позицию агента X не составляет труда. Однако здесь есть условности. Агент, представляет собой тележку на колесах. Находясь в идеальных условиях с максимальным коэффициентом трения, а также возможностью мгновенно набирать постоянную скорость, действительно достаточно провести элементарные математические вычисления, чтобы найти $X = x + u$. Но на практике агенту необходимо время, чтобы разогнаться, затормозить, колеса могут прокручиваться из-за плохого сцепления с дорогой. Обычно позиции $X = x + u$ выбирают как начальную оценку позиции агента и называют **одометрией**. Ясно, что настоящая позиция агента находится в некоторой окрестности одометрии.

Подробно опишем первый множитель. Он показывает вероятность того, что наблюдение z доступно с позиции X на карте m. Если позиция X является «истинной», то такая вероятность будет равна 1, а если позиция неверна – должна быть близкой к нулю. Именно для этих целей выбирается карта в виде ячеек занятости. Накладывая скан (который является наблюдением Z) на ячейки карты m, можно оценить, как много точек лазерного скана попало в «занятые» ячейки карты. Поиском позиции X, для которой вероятность $p(z|m,X)$ максимальна, занимается скан матчер.

Если карта является фрактальной, то может существовать несколько позиций, для которых вероятность $p(z|m,x)$ одинаково высока. В этом случае важную роль играет второй множитель из уравнения (1), который ограничивает область поиска позиции X.

После того, как наиболее вероятная позиция агента найдена, а скан z конфликтует с картой или может ее дополнить, карту необходимо обновить.

Случай конфликта – это наиболее неприятная ситуация. В простом случае нет инструмента, чтобы проверить: произошла ли ошибка в определении позиции X на текущем шаге или на одном из предыдущих (случай динамического окружения не рассматривается). Наиболее популярным решением является: полагаться на более новые данные и отбрасывать «старые» вероятности в конфликтующих ячейках карты.

1.4 Фильтр частиц. Gmapping

Основным недостатком подхода, описанного в 1.3, является невозможность **простой** проверки консистентности выходных данных. Если в процессе работы алгоритма что-то пошло не так, сам алгоритм не сможет распознать эту ситуацию и впоследствии получит неверные данные.

Для решения этой проблемы применяется **фильтр частиц** [22]. Назовем частицей состояние системы в определенный момент (позиция агента и построенная им карта).

Предположим, что мобильный робот добрался до одной из двух присутствующих на карте вертикальных колонн. Также допустим, что других ориентиров у агента нет. Следовательно, агенту необходимо выполнить локализацию, используя лишь данные, о нахождении перед одной из колонн и неизвестно, перед какой именно. В этот момент можно сделать две равновероятные гипотезы о текущем положении. Именно такой алгоритм использует Gmapping [15].

Во время выполнения алгоритма могут возникать ситуации, когда придется добавлять гипотезы, и тогда может оказаться более 10 различных гипотез о положении агента и, соответственно, о карте. Карты в частицах отличаются, поскольку после разделения возможных позиций все частицы «принимают» одинаковые последующие наблюдения и встраивают их в текущую карту частицы. Таким образом, уже через несколько шагов после разделения на две частицы, карты могут значительно различаться.

Если во время работы алгоритма происходит ситуация, когда измерения не могут быть сопоставлены с картой единственным образом, необходимо создать новую частицу и отслеживать сразу обе, вычисляя правдоподобность каждой из них.

Правдоподобность гипотезы основывается на консистентности карты. Так, если самые новые наблюдения не противоречат построенной карте (это можно выяснить на этапе скан матчинга при вычислении стоимости скана), то карту можно считать консистентной и гипотезу правдоподобной. Если же новые наблюдения регулярно противоречат карте, можно сделать вывод о том, что гипотеза о позиции агенты была сделана неверно, и ее правдоподобность – низкая. Пример карт, которые могут быть быстро построены на промежуточном шаге алгоритма gmapping в реальных условиях, представлен на рисунке 1.7.

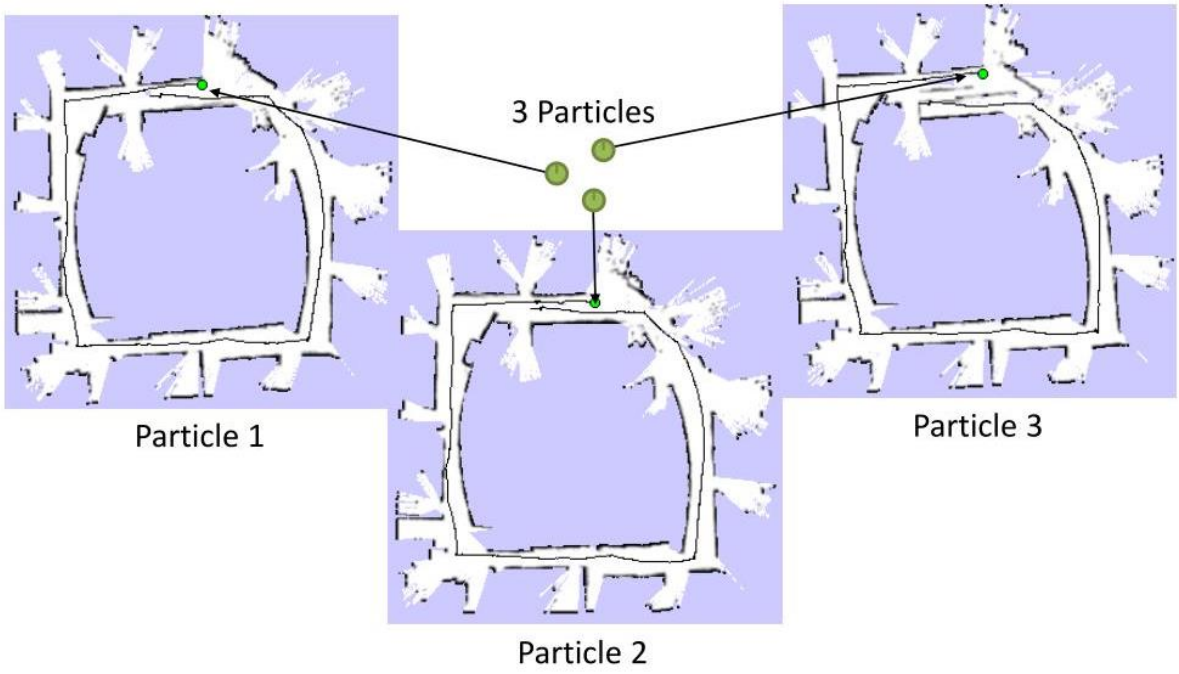


Рисунок 1.7 – Три частицы в алгоритме gmapping.

Фильтр частиц (гипотез) должен вычислять правдоподобность всех гипотез и следить за тем, чтобы в наборе частиц не было невероятных. Одновременно с этим он должен дополнять набор частиц, если на этапе скан матчинга были найдены несколько высоковероятных гипотез. В обоих случаях фильтр обладает инструментарием, чтобы добавлять или удалять частицы. Этот процесс называется **ресамплинг** [25].

Примером алгоритма, использующего такой подход является gmapping, который в данный момент считается стандартным алгоритмом решения задачи SLAM.

Алгоритм фильтрации частиц, применяемый в gmapping представляет последовательность следующих шагов:

1) наложить текущее наблюдение на карту в каждой частице;
2) вычислить вес каждой частицы;
3) если есть частицы с предельно низким весом – удалить их;
4) если на шаге 1 есть несколько способов наложить наблюдение на карту – добавить частицы;
5) перейти к шагу 1.

Недостатком такого алгоритма является его вычислительная сложность, поскольку каждая новая частица добавляет дополнительную задачу SLAM. Кроме того, решение об удалении частицы из списка рассматриваемых принимается на основе различных эвристик.

1.5 Графовый SLAM. Cartographer

Наиболее популярным в данный момент является графовый подход к построению карты [23], [24], [54]. При таком подходе начальные измерения объединяются в узлы графа, а ребрами служит трансформация между позициями, с которых можно эти измерения снять. Более подробно простой графовый алгоритм можно описать следующим образом. Когда агент получает новое измерение, он добавляет в граф новую вершину, в которую записывает это измерение. Новая вершина соединяется одним ребром с предыдущей добавленной вершиной. Вес этого ребра – это совокупность координат: перемещение и поворот между позициями агента на текущем и предыдущем шаге. Этот вес можно получить, используя одометрию или скан матчер, возможна и комбинация этих методов. Спустя некоторое время после начала работы алгоритма будет построен граф из некоторого количества вершин, каждая из которых соединена только с соседними вершинами.

Такой граф представляет собой карту, и часто вместо «графовый алгоритм» говорят «алгоритм с графовым представлением карты». Для того чтобы построить карту в обычном ее понимании, необходимо пройти по всем узлам графа и, используя позицию агента и позиции точек в измерениях, нанести их на карту. В результате получится облако точек, описывающих карту. Также вместо такого облака можно использовать сетку занятости.

Когда агент посещает место, в котором он уже побывал, то есть снимает измерение, совпадающее с одной из существующих вершин графа, то подключается алгоритм замыкания циклов и обновления ребер в цикле. Может оказаться, что сумма перемещений в ребрах, образующих цикл, не равна нулю (то есть накопившаяся ошибка заставляет данные перемещения агента противоречить данным сенсора). В этом случае необходимо обновить веса ребер так, чтобы сумма перемещений агента внутри цикла действительно равнялась нулю.

Процесс перебалансировки ребер невозможно стандартизовать, поскольку непонятно, какое из ребер внесло наибольшую ошибку. Обычно для решения такой задачи используется фильтрация, основанная на старении данных: более старые измерения изменяют значительнее, чем более новые. Существует подход, где каждое ребро графа изменяется одинаково, то есть накопленная неточность измерений делится поровну на все ребра.

Для того чтобы минимизировать описанные выше недостатки, существуют различные модификации этого подхода. Так, например, вместо создания новой вершины графа, содержащей каждое новое измерение, несколько наблюдений группируют в одной вершине. Тогда граф становится меньше и процесс поиска повторяющихся вершин значительно ускоряется. Однако в этом случае возникает вопрос о достаточном количестве наблюдений в вершине. В некоторых случаях используют разделение по расстоянию: как только агент удалился от первоначальной точки на некоторое пороговое расстояние, наблюдения станут записываться в другую вершину.

Краеугольным камнем графового алгоритма является вычислительная сложность. Ведь для замыкания циклов необходимо сопоставить новое измерение

с каждой вершиной графа. Существует подход, который позволяет ускорить этот компонент алгоритма. Согласно этому подходу проверка уникальности нового измерения происходит не сразу в момент появления этого измерения. Процесс поиска циклов в графе запускается отдельно от процесса снятия измерений. Например, измерения могут поступать с частотой 30 наблюдений в секунду, и запускать с такой частотой перебор всех вершин в графе – очень затратная задача. В этом случае поиск циклов происходит гораздо реже: например, раз в 2 секунды.

Если объединить две рассмотренные модификации для графового алгоритма (объединение нескольких измерений в одну вершину графа и вынесение замыкания циклов в отдельный поток), то полученный алгоритм будет близок к алгоритму, представленному компанией Google и носящему название cartographer [24].

На высокопроизводительных настольных компьютерах Cartographer работает практически в режиме реального времени: алгоритм успевает обработать очередное наблюдение **до** момента, когда будет снято следующее. Если запустить такой алгоритм на низко производительных компьютерах, таких как Raspberry Pi [47], то можно увидеть, что замыкание циклов – самая дорогостоящая операция – не успевает завершиться до появления очередного наблюдения. Для решения этой проблемы в алгоритме Cartographer замыкание циклов состоит из двух этапов: пред- и постобработки. На этапе предобработки обновляются веса ребер графа, а постобработка запускается один раз в конце работы алгоритма, чтобы продемонстрировать пользователю конечную карту.

Подобный подход, когда новое измерение необходимо встроить в уже построенную карту, отлично подходит для масштабирования алгоритма. В многоагентном алгоритме такой инструмент очевидно полезен, поскольку агент вычисляет положение другого агента, используя его наблюдения. Однако недостатком многоагентного графового подхода является его вычислительная сложность.

1.6 Выводы по первой главе

Среди решений одноагентной задачи SLAM на данный момент самыми точными являются графовые алгоритмы. Такое название класса алгоритмов обеспечивается графовой структурой карты. В вершинах графа располагаются измерения, снятые роботом с разных позиций. Ребра графа являются взвешенными и описывают перемещение, которое должен совершить робот из предыдущей вершины, чтобы получить возможность снять наблюдение из следующей вершины. Однако высокая точность обусловлена повышенными требованиями к вычислительным ресурсам. Для уменьшения этих требований можно воспользоваться фильтром частиц. Использование такого фильтра позволяет сохранять и осуществлять рейтинг нескольких гипотез о состоянии карты и позиции на случай, если самая вероятная гипотеза содержит ошибку. Но и применение фильтра частиц также требует повышенного количества вычислительных ресурсов.

Для того, чтобы решить задачу на низкопроизводительных устройствах, необходимо базироваться на двумерных неграфовых, одногипотезных и лазерных алгоритмах. Именно такой класс алгоритмов является наименее требовательным к ресурсам, но проигрывает в точности другим классам. Проблеме увеличения точности посвящены следующие главы.

2 Многоагентные алгоритмы решения задачи slam, их структура и архитеткура

Данная глава посвящена анализу многоагентных алгоритмов решения задачи SLAM, особенностям их архитектуры. В ней описаны характеристики, которыми должен обладать многоагентный алгоритм, подходящий для маломощных устройств. Классификация многоагентных алгоритмов во многом совпадает с классификацией одноагентных, поэтому данная глава посвящена только новым проблемам, с которыми сталкиваются группы агентов, решающие задачу SLAM. Основные положения из этой главы представлены в [19].

В отличие от одноагентного SLAM алгоритма в системе, содержащей несколько агентов, возникают дополнительные задачи.

1 *Определить роли мобильных агентов*. Агенты могут быть равноправны и выполнять одинаковые задачи, а могут реализовывать, например, клиент-серверную архитектуру.

2 *Определить способ вычисления взаимного расположения агентов*. Это необходимо, чтобы объединять построенные карты. Существуют подходы, когда взаимное расположение агентов известно в начальный момент времени. Также существуют алгоритмы, которые предполагают вычисление взаимного расположения в момент, когда агенты находятся в непосредственной близости относительно друг друга.

3 *Определить способ объединения карт*. Возможен подход, когда новые измерения агентов встраиваются во все карты сразу. Существуют алгоритмы, в которых, наоборот, карты объединяются малое количество раз за всю работу алгоритма. В любом случае необходимо определить методику разрешения конфликта новых данных с накопленными.

4 *Определить, что является выходными данными*. Если в системе присутствует несколько агентов, каждый из которых хранит собственный отличный от остальных экземпляр карты, то возникает вопрос: данные какого из экземпляров карты считать выходными?

2.1 Классические методы решения задачи многоагентного SLAM

Прежде, чем анализировать рассмотренные задачи, следует обратиться к алгоритмам, которые являются пионерами в этой области.

Один из первых алгоритмов, решающих задачу многоагентной одновременной локализации и построения карты при условии заранее неизвестного взаимного расположения агентов предложен в [43]. В этой статье авторы описывают алгоритм, основанный на информационных фильтрах. Такой подход позволяет избавиться от свойственного SLAM алгоритму недостатка, основанного на расширенном фильтре Калмана (EKF SLAM): вычисление матрицы ковариаций найденных на карте препятствий. Вместо этого предлагается вычислять информационную матрицу, при построении которой допускается, что препятствия могут быть независимы. Основным недостатком матрицы ковариаций в EKF SLAM алгоритме является вычислительная сложность работы, а также сложность добавления в нее новых препятствий. Кроме того, при большом количестве препятствий, работа с матрицей большой размерности неизбежно влечет за собой погрешности вычислений. Применение же информационной матрицы, напротив, позволяет учитывать только те препятствия, которые мобильный агент наблюдает в конкретный момент времени. Такой подход одновременно увеличивает скорость работы алгоритма, основанного на выделении препятствий, и позволяет создать децентрализованную многоагентную систему.

В предложенном методе процесс объединения карт является **прямолинейным**. Если есть несколько карт, которые следует объединить, то сначала объединяется пара карт, потом полученная карта объединяется с третьей и так далее. Для того чтобы выполнить слияние карт, взаимное расположение агентов предлагается вычислять при помощи **наложения** карт, построенных разными агентами. Препятствия на одной карте разбиваются на тройки, и для каждой тройки (независимо от других) находятся соответствия на другой карте (информационной матрице). Сложность такого алгоритма составляет $O(N\log(N))$. После нахождения наиболее вероятной трансформации между картами, позиция

каждого препятствия в обеих картах пересчитывается по Байесовским правилам объединения случайных величин.

Качество работы такого подхода напрямую зависит от способа вычисления дескрипторов препятствий. В данной статье авторы предполагают, что, распознав препятствие, будет возможно определить, является ли оно новым, или такое препятствие уже присутствует в карте. Более того, если препятствие существует, должен быть способ определения, какое именно это препятствие. Требование к наличию такого механизма – основной недостаток этой работы, поскольку он напрямую зависит от окружения. К недостаткам можно отнести и тот факт, что авторы не предлагают метода определения ошибок при добавлении препятствий в карту.

Другой выделяющийся многоагентный SLAM алгоритм был предложен в [46]. Авторы этой работы принимали участие в конкурсе RoboCup Rescue Virtual Robots в 2006 и выиграли награду за лучшую построенную карту. Они утверждают, что предложенный ими графовый по структуре алгоритм позволяет уменьшить количество используемой памяти для хранения карты.

Авторы предлагают запустить графовый алгоритм на каждом агенте независимо, а затем объединять построенные графы карт, используя методы замыкания и оптимизации циклов, которые используются при добавлении новых узлов графа. В качестве сенсора авторы предлагают использовать лазерный дальномер. Сравнивая два последовательных скана при помощи скан матчера, они предлагают вычислять перемещение мобильного робота между точками, с которых роботы наблюдают такие лазерные сканы. Когда такая трансформация вычислена, в карту-граф добавляется новый узел, содержащий новый лазерный скан, а ребром графа является именно эта трансформация. Карта занятости, позволяющая визуализировать результат работы алгоритма, строится в конце, для чего используются все узлы графа.

Алгоритм замыкания циклов основывается на выделении особых точек на лазерном скане, поскольку он разрабатывался в условиях конкурса, где окружение

было построено и маркировано таким образом, что позволяло выделять особые точки быстро и робастно.

К недостаткам работы можно отнести тот факт, что авторы не рассматривают ситуации, когда скан матчер допустил ошибку при вычислении взаимного расположения лазерных сканов. Также не учитываются возможные ошибки при замыкании циклов, хотя подобные ошибки могут быть решены, если мобильный робот несколько раз посетит одно и то же место, и алгоритм замыкания циклов выполнится несколько раз.

В [5] рассмотрена задача робастного наложения карт в многоагентном алгоритме SLAM. Предлагаемый метод состоит из двух шагов: определение взаимного расположения агентов и наложения карт препятствий. Для выполнения первого шага агенты должны располагаться в непосредственной близости относительно друг друга, начинать движение в одном направлении, чтобы в течение некоторого времени иметь возможность наблюдать одинаковое окружение. Во время выполнения второго шага каждый агент получает карты от другого агента, объединяет построенные карты и удаляет из них дубликаты. Этот алгоритм основывается на расширенном фильтре Калмана, следовательно, карта каждого агента сопровождается матрицей ковариаций. Таким образом, задача поиска дубликатов в карте является NP-трудной, поскольку она требует сравнения каждой особой точки в карте с каждой другой.

Обзор «классических» решений задачи многоагентного SLAM иллюстрирует, что при создании всех подобных алгоритмов авторы исходили из возможности выделения особых точек на карте. Новые решения отступают от данной концепции. Авторы «классических» решений старались предлагать децентрализованную систему, в которой если и существует центральный узел, то он выполняет лишь роль маршрутизатора для распределения потоков информации. Дальнейший анализ продемонстрирует, что такая иерархия не является единственно возможной.

2.2 Роли в многоагентном SLAM алгоритме

Архитектура многоагентного SLAM алгоритма влияет на алгоритм, который выполняет каждый агент. Поэтому в первую очередь необходимо разобраться, какие роли могут выполнять агенты, находящиеся в системе.

Самой очевидной является архитектура с одним сервером и множеством эквивалентных агентов. Например, такая архитектура была представлена в [20]. Следуя предложенному алгоритму, карта в единственном экземпляре строится на сервере, который принимает начальные наблюдения от летающих дронов. Дроны не выполняют никаких специальных действий, кроме снятия измерений и отправки их серверу. Перед началом работы алгоритма указывается начальная позиция всех агентов, соответственно сервер имеет возможность строить одну карту, в которую последовательно встраиваются наблюдения каждого агента. Одновременно с этим сервер решает задачу локализации для каждого дрона.

Аналогичный подход предлагается в [11] и [28]. Отличие заключается в том, что позиция агентов изначально может быть не задана с абсолютной точностью. Для того чтобы в этом случае строить одну карту, предлагается использовать фильтр частиц, подобный алгоритму gmapping, описанному в главе 1.

К достоинствам клиент-серверной архитектуры можно отнести следующие факты:

– система устойчива к потере одного или нескольких мобильных агентов до тех пор, пока в системе остается хотя бы один мобильный агент;

– стоимость роботов, выступающих в роли агентов, может быть очень низкой, поскольку они могут выполнять роль только подвижных сенсоров без каких-либо вычислительных операций.

К недостаткам системы можно отнести зависимость от центрального сервера. Как и в любой другой клиент-серверной архитектуре мощности сервера должно быть достаточно, чтобы обработать все данные, полученные от клиентов. Это потенциально усложняет масштабирование системы. Кроме того, сбой в работе сервера влечет за собой безвозвратный выход из строя всей системы. Также в случае проблемы с сетевым соединением у сервера, в первую очередь

будет страдать точность построенной карты. Описанные недостатки привели к меньшей популярности алгоритмов, использующих клиент-серверную архитектуру, нежели децентрализованных алгоритмов.

2.3 Способы вычисления взаимного расположения агентов и слияние карт

Если изначально знаний о расположении роботов в пространстве нет, то необходимо вычислять их взаимную ориентацию в процессе работы программы. Если в системе отсутствует сервер, то эту задачу берут на себя роботы-агенты. Например, алгоритм, в котором агенты напрямую вычисляют взаимное расположение, когда наблюдают друг друга, представлен в [58]. В этой работе авторы предлагают обозначить роботов яркими метками, различимыми на камере, чтобы быстро отличать робота от любого объекта окружения. А самих роботов снабдить всенаправленной камерой, чтобы не упустить из виду других агентов во время выполнения алгоритма SLAM.

Другим подходом [12] является наличие у всех агентов априорного знания о том, какой агент несет ответственность за определенную область пространства. Например, пространство можно заранее разметить. Тогда, оказавшись в месте, откуда можно наблюдать метку, агент сразу определит, кто ответственен за эту область пространства, и кто уже наблюдал эту метку. Такой подход позволяет вычислять расположение между агентами, исходя из уже построенных карт.

Другой важной задачей для многоагентных алгоритмов является так называемое **слияние карт** – это подход для объединения карт, созданных всеми агентами в системе [8], [9], [35].

Как правило, при объединении карт необходимо решить следующие вопросы:

- какая информация должна быть предоставлена агентам; должна ли это быть полная структура карты (какой бы она ни была) или только ее часть;
- что должен делать агент, если карта другого агента отличается от его собственной;

– насколько агент может доверять тем частям на чужой карте, которые неизвестны на собственной.

Самым простым решением является архитектура сервер-клиент, где на сервере существует только одна копия карты, и клиенты обновляют именно эту карту. Если же агенты независимы и общаются друг с другом напрямую, то на картах, созданных независимыми агентами, может возникнуть конфликт. Тогда вопрос об урегулировании этих конфликтов остается открытым. Конфликт может возникнуть в случае сбоя одноагентного алгоритма SLAM, если этот сбой не был обнаружен и обработан должным образом. Обычно решение этой проблемы зависит от типа алгоритма SLAM. Вероятностные методы являются наиболее распространенными. Наивным является метод использования средней суммы карт, это приводит к ситуации, когда ошибочная карта одного агента вносит ошибку в карту всех других агентов. Высокоуровневая схема слияния карт показана на рисунке 2.1.

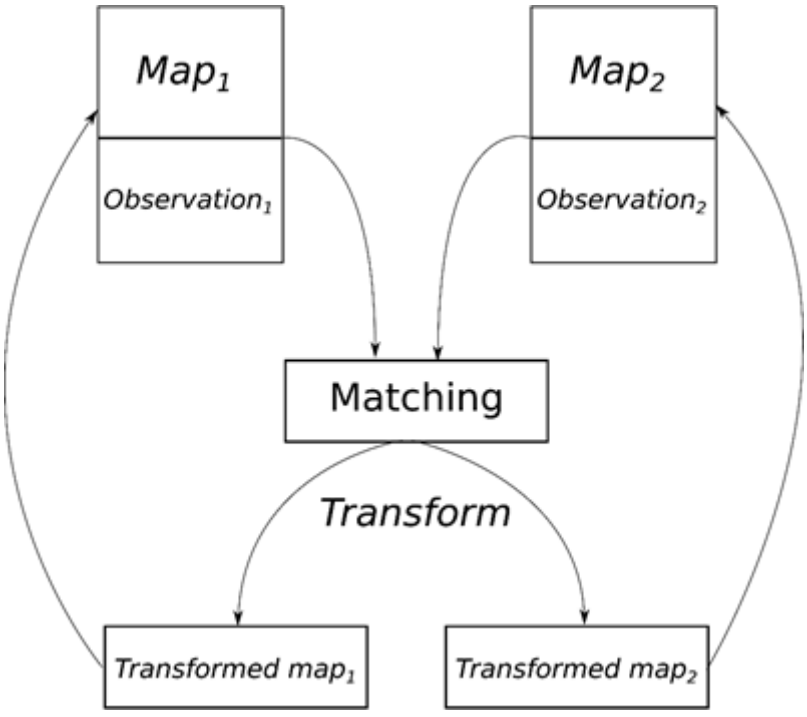


Рисунок 2.1 – Схема слияния карт

Некоторые части этой схемы могут отличаться в зависимости от конкретной архитектуры алгоритма. Например, в [20], [45], [58] этап сопоставления преобразованной и исходной карт тривиален – предыдущая карта заменяется на

обновленную. Но в графовых подходах архитектура позволяет обновлять карту, обнаруживать конфликты и удалять повторы на карте. Эти преимущества снижают быстродействие, и поэтому в [36] вместо полного слияния карт вычисляется только преобразование между ними. Слияние выполняется один раз в конце работы алгоритма.

Другой задачей слияния карт является нахождение консенсуса, когда две или более карты (или их части) отличаются друг от друга на разных агентах. Очевидно, что может возникнуть ситуация, когда относительное положение агентов определено правильно, но части построенных карт не совпадают. В ранее рассмотренном алгоритме SLAM на основе графов существуют предустановленные модули, которые могут исправить это несоответствие, но для других подходов этот вопрос является более сложным. Для таких случаев необходимо решить, является ли равенство transformed map 1 и transformed map 2 (рисунок 2.1) необходимым. Если является, то обычно применяется некоторая эвристика, или окончательная карта состоит только из тех частей, которые совпадают на обеих картах. Если конечная карта на агентах различается, тогда алгоритм слияния может учитывать приоритет карт: собственную карту можно считать достоверной, а все конфликтующие части на других картах могут быть отброшены или объединены с низкой вероятностью. Также можно рассчитать корреляцию карт и обнаружить как минимум конфликтующие части.

Последний важный вопрос – это подходящее время для коммуникации агентов и передачи построенных карт. Наиболее распространенным подходом является запуск этого процесса, когда два агента расположены в одном месте или близко друг к другу. Это условие позволяет предположить, что оба агента фиксируют одно и то же наблюдение, и карта вокруг места встречи построена одинаково. Однако в [12] предложена идея разделения пространства наблюдения на непересекающиеся области и передачи каждой области одному агенту. Если агент достигает области, принадлежащей другому агенту, он может связаться именно с этим агентом и получить все знания об области, которые тот наблюдал до сих пор.

2.4 Модель многоагентных алгоритмов решения задачи SLAM

В первую очередь необходимо ввести понятие «стая», которая является исполнителем многоагентного алгоритма SLAM. Стая роботов – набор автономных роботов, имеющих сопоставимое техническое оснащение для наблюдения окружающей среды в ограниченной окрестности, не имеющих иерархии и обменивающихся информацией друг с другом для совместной разметки окружающей среды. Из этого следует, что каждый робот в стае выполняет одни и те же функции. Другими словами, в стае отсутствует вертикальная иерархия, в том числе отсутствует центральный узел – сервер, отсутствие которого может повлечь за собой остановку работы всей системы.

Каждый агент в стае самостоятельно определяет, в какой момент необходимо начать обмен информацией с другими агентами. Естественно, основания для принятия такого решения у всех членов в стае одинаковые. Модель рассматриваемого поведения в стае представлена на рисунке 2.2.

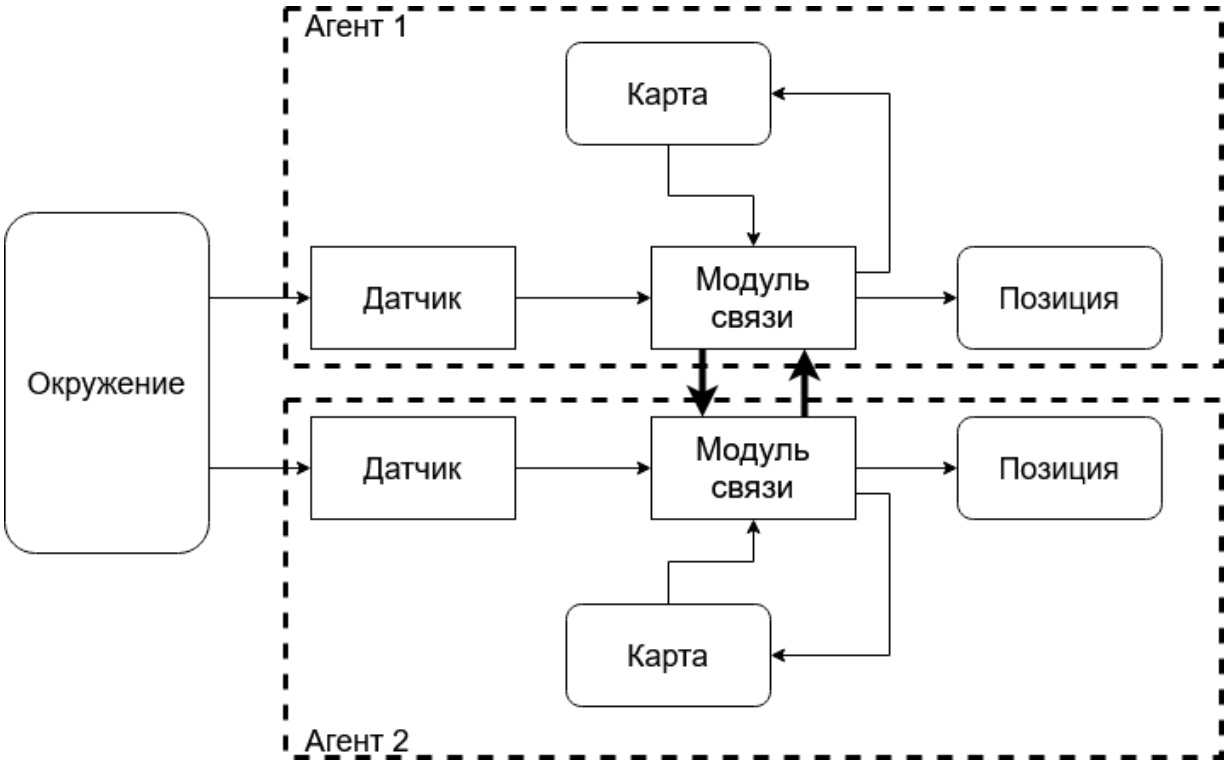


Рисунок 2.2 – Модель, описывающая рассматриваемое семейство многоагентных алгоритмов SLAM

Центральным узлом здесь является модуль связи, который может выполнять комплексирование собственных данных с данными агентов. В случае если связь с

другими роботами в стае не требуется, модуль связи обрабатывает только собственные данные

2.5 Графовый многоагентный SLAM

Среди одноагентных алгоритмов, решающих задачу SLAM, наиболее популярными на данный момент являются графовые [12], [30], [36]. Подробно структура графового алгоритма рассматривалась в главе 1. Популярность этих алгоритмов обусловлена масштабируемостью и возможностью исправлять ошибки в уже построенной карте. Последняя особенность таких алгоритмов является очень востребованной в многоагентных подходах.

В рамках графового подхода для каждого нового измерения подыскивается узел в графе. Для выполнения этого действия существует алгоритм замыкания циклов, который может запускаться в несколько раз реже, чем снятие измерений. Такая идея может быть эффективно использована в графовом алгоритме для вычисления взаимного расположения агентов. Структуру графового алгоритма можно без изменений применить для этой задачи. Если первый и второй агенты посещали одно и то же место, то после получения измерения с этого места от второго агента будет возможно выполнить замыкание циклов и определить местоположение, с которого были сняты эти измерения.

Таким образом, для графового подхода можно выделить два основополагающих достоинства: простота в масштабировании и возможность глобального изменения карты в процессе работы алгоритма. Графовый подход имеет тот же недостаток, которым обладали одноагентные графовые SLAM алгоритмы: понижение скорости работы алгоритма со временем, когда граф накапливает большое количество вершин.

Один из примеров реализации графового алгоритма приведен в [36]. В качестве датчика измерений здесь выступает лазерный дальномер. Архитектура этого решения – распределенная сеть, в которой агенты могут связаться друг с другом только находясь на небольшом расстоянии друг от друга. Во время обмена информацией каждый агент передает другому свои текущие измерения. Затем

происходит обновление карт всех агентов и обмен обновленными графами для выполнения слияния карт.

На этапе обработки текущего наблюдения вторым агентом происходит выбор узла собственного графа, с которым можно сопоставить текущее наблюдение и вычислить взаимное расположение первого и второго агентов (даже если они находятся не в одной точке пространства, а на некотором расстоянии друг от друга).

Для замыкания циклов данный алгоритм использует популярную библиотеку g2o, являющуюся стандартом среди разработчиков графовых алгоритмов SLAM.

Другой пример реализации графового алгоритма описан в [12]. Отличительной особенностью этого алгоритма является разбиение всего окружения на участки, за каждый из которых отвечает определенный агент. Как только один из агентов оказывается в области наблюдения другого, они могут установить связь и начать обмен данными. Такой подход гарантирует, что второй агент в текущий момент снимает наблюдение, которое уже обрабатывал первый. А это означает, что возможно вычислить взаимное расположение агентов и произвести объединение карт.

Отличительным достоинством рассматриваемого авторами алгоритма является малое количество передаваемого по сети трафика. По их утверждению качество построенного решения не уступает аналогичным алгоритмам.

Недостатком подхода является разделение пространства на районы: такой подход возможен, только если окружение известно заранее (офисы, склады, небольшие районы города).

2.6 Неграфовый многоагентный SLAM

Главным недостатком графовых алгоритмов является потребность в больших вычислительных ресурсах. В одной из ключевых статей о глобальной локализации [21] автор ссылается на парадигму: «точность вычислений должна зависеть от времени, которое дано на обработку». Если алгоритм разработан согласно этой парадигме, то его точность напрямую зависит от вычислительных ресурсов мобильного агента (чем производительнее агент, тем больше операций за одно и то же время он сможет выполнить, и тем точнее будет ответ). Графовый подход не укладывается в данную парадигму, ему требуется фиксированное количество времени для получения точного ответа. Поэтому в данном разделе показаны альтернативные подходы, их преимущества и недостатки.

Уже упоминалось, что одной из первых модификаций базового EKF SLAM алгоритма являлся FastSLAM. Он работал быстрее, погрешность его работы была мала по сравнению с выигранными ресурсами. Таким образом, среди многоагентных SLAM алгоритмов закономерно появился многоагентный FastSLAM алгоритм [45].

Авторы этого подхода предлагают запускать одноагентный FastSLAM на каждом отдельном агенте. В данной работе предложена архитектура сети с маршрутизатором, однако алгоритм сможет также хорошо работать и в архитектуре с распределенной сетью. Аналогично одноагентному подходу здесь используется фильтр частиц, то есть одновременно учитываются и вычисляются вероятности сразу нескольких возможных состояний карты и позиции роботов.

Как и в любом алгоритме с фильтром частиц возникает вопрос, как сопоставлять два набора частиц, построенных разными агентами. Прямолинейный вариант – сравнивать каждую частицу с каждой – занимает слишком много вычислительного времени; поэтому авторы в рассматриваемой статье идут по пути известного ухудшения точности в обмен на скорость работы алгоритма. В статье предлагается объединить все частицы с учетом их веса и построить усредненную карту на каждом агенте, а затем объединить усредненные карты. Безусловно, такой подход вносит дополнительный шум в построенную карту,

негативно влияет на все построенные во время работы частицы, но он позволяет выполнить слияние карт максимально быстро.

Другой пример неграфового подхода описан в [49]. Его отличительная особенность также состоит в том, что он представляет собой успешную реализацию архитектуры с сервером. Несколько дронов с помощью видеокамер исследуют окружение и передают данные серверу, где и происходит слияние измерений и построение карты.

Общей чертой рассмотренных выше алгоритмов является сложность обновления уже построенных участков карты. Графовый подход позволяет значительно обновлять карту, не изменяя ее консистентность, если новые измерения противоречат старым. В неграфовых подходах нет инструмента изменения части карты так, чтобы на ней не осталось артефактов. Частично это возможно при использовании фильтра частиц, однако его использование порождает неразрешимые вопросы для многоагентной архитектуры.

Алгоритм [49] реализует фильтр частиц, и сервер (в зависимости от данных, полученных от дронов) может вычислить, какая из гипотез о положении дронов имеет наибольшую вероятность.

2.7 Выводы по второй главе

Среди многоагентных алгоритмов по-прежнему большой популярностью пользуются алгоритмы, роли агентов в которых распределяются неравномерно. Ключевой недостаток таких алгоритмов заключается в том, что выход из строя разных агентов по-разному отражается на работе системы. В том числе, если в системе присутствует сервер, то неполадки любой природы в его работе могут привести к остановке работы всей системы.

Кроме того, многоагентные алгоритмы разделяются по количеству информации о первоначальном положении роботов в системе. На практике часто невозможно определить начальную позицию робота с погрешностью менее нескольких сантиметров, поэтому алгоритмы, вычисляющие взаимное

расположение агентов без использования априорных знаний, имеют более широкую область применения.

В силу популярности графовых подходов в одноагентных алгоритмах решения задачи SLAM, существует большое количество графовых многоагентных алгоритмов. Дополнительным преимуществом таких алгоритмов над неграфовыми является то, что они по определению содержат компонент, реализующий замыкание циклов, который можно использовать для определения взаимного расположения агентов.

Вместо этого предложено понятие стаи роботов, решающих задачу одновременной локализации и построения карты. К требованиям (глава 1) о принадлежности разрабатываемого алгоритма к классу двумерных неграфовых одноагентных алгоритмов добавляются требования о структуре агентов. Для обеспечения отказоустойчивости системы необходимо, чтобы все агенты выполняли одинаковые роли: и снимали наблюдения, и обрабатывали их, и выбирали время для синхронизации. Помимо этого, для увеличения области применимости разработанного алгоритма необходимо вычислять взаимное расположение агентов во время их синхронизации вместо использования априорного знания о начальном расположении агентов.

3 Многоагентный масштабируемый алгоритм решения задачи slam для стаи мобильных роботов

Данная глава посвящена описанию горизонтально масштабируемого алгоритма одновременной локализации и построения карты стаей роботов, ограничений его применимости, и ответам на вопросы к многоагентному SLAM, поставленные в главе 2.

Описанный в этой главе алгоритм был впервые представлен в [18]. Рассмотрим его более детально.

3.1 Физические и технические ограничения применения алгоритма

В выводах к главе 2 перечислены требования, которые необходимо применить к алгоритму для достижения условий высокой производительности. Помимо этого, необходимо определить условия использования алгоритма, требования к оборудованию мобильных роботов и технологиям, используемым агентами.

3.1.1 Ограничения, накладываемые на окружающую среду и агентов

Разработанный алгоритм будет использоваться агентами, находящимися в помещении. Это означает, что размер исследуемого пространства не превосходит нескольких сотен квадратных метров. В рассматриваемом случае невозможно использовать GPS, поскольку погрешность определения местоположения с помощью GPS составляет несколько метров, что недопустимо. В качестве датчика можно использовать лазерный дальномер, эффективный радиус которого измеряется десятками метров. Выбор лазерного дальномера в качестве сенсора позволяет строить карту в виде сетки занятости. Таким образом, карта, построенная в результате работы алгоритма, будет представлять собой план здания, по которому перемещаются агенты.

Важным является тот факт, что алгоритм не решает задачу управления мобильной платформой для исследования окружения. В задачу разработанного

алгоритма входят только обработка измерений, полученных от лазерного дальномера и датчика одометрии, и построение карты.

Помещение, в котором перемещаются агенты не должно содержать динамически изменяющихся областей. Появление редких и небольших шумов возможно (например, перемещение людей или других мобильных платформ). Однако не допускается перемещение мебели или колонн в помещении, особенно в моменты, когда агент не наблюдает за соответствующей частью окружения.

Следующим требованием является отсутствие перепада высоты: необходимо, чтобы все наблюдения производились в одной плоскости. Кроме того, для достижения этого требования нужно плотно зафиксировать лазерный дальномер на агенте.

Обязательным является условие, что все агенты, выполняющие алгоритм SLAM, должны быть одинаково оснащены: не только иметь одинаковые вычислительные возможности, но и одинаковую мобильную платформу и одинаковые лидары. Это требование обусловлено тем, что каждый агент выполняет одинаковый алгоритм, поэтому структура выходных данных у агентов должна совпадать. Если, например, вычислительные мощности одного из агентов окажутся больше, чем у другого, то качество построенной карты у агента с меньшей мощностью окажется хуже вследствие нехватки времени на обработку данных. В этом случае карта, полученная в результате слияния будет не только содержать области с разной степенью четкости, но и в случае пересечения некоторых областей менее четкая карта может внести значительную погрешность в результат.

Область применения предлагаемого алгоритма – это малобюджетные и, следовательно, низко производительные, агенты. Поэтому алгоритм не должен содержать шагов, на выполнение которых существует ограничение по времени (например, обработка новых измерений). Чем больше времени алгоритм получает на работу, тем более точным будет результат – это условие должно соблюдаться при запуске на любых устройствах. По такому принципу работает, например, стохастический скан матчер Монте-Карло. Идея такова, что он ищет наиболее

точную позицию агента, выбирая ее случайным образом из области поиска. Понятно, что чем больше времени получит такой скан матчер на работу, тем больше вероятность выбора позиции, наиболее близкой к оптимальной.

3.1.2 Используемые технологии

Поскольку разработанный алгоритм будет применяться для мобильных роботов, необходимо реализовать его таким образом, чтобы он мог быть запущен на максимальном количестве различных аппаратных устройств. В настоящее время существует огромное количество разнообразных платформ, которые могут выполнять алгоритм SLAM: начиная с Turtlebot [55], Duckiebot [16] – тележек, оснащенных простым микропроцессором, и заканчивая pr2-bot [56] – сложной системой взаимодействующих модулей.

Чтобы иметь возможность запустить высокоуровневый алгоритм на разных конфигурациях роботов существует большое количество различных **фреймворков** – программ, обеспечивающих и облегчающих разработку и объединение разных компонентов большого программного проекта. Такие фреймворки представляют собой прослойку между аппаратной и программной составляющими любого робота. Степень интеграции с аппаратной частью также может варьироваться. Так, например, есть целый класс различных фреймворков, называемых DDS (Data distribution service) [32], задача которых предоставить обмен сообщениями между различными узлами программы. Шаблон «читатель-писатель», реализованный в таких фреймворках, широко используется при программировании роботов, поскольку позволяет всем узлам агента выполнять свои функции децентрализованно, обмениваясь данными с соседними узлами.

3.2 Компоненты многоагентного алгоритма SLAM

Прежде, чем описывать структуру многоагентного алгоритма, рассмотрим последовательность шагов, которую каждый агент проделывает независимо от остальных. Эту последовательность шагов назовем **ядром многоагентного алгоритма**.

Описываемое ядро не является новым, но его подробное рассмотрение необходимо, поскольку многоагентный алгоритм во многом основывается на модулях и методах, использованных в одноагентном алгоритме. Необходимо подчеркнуть, что предлагаемый многоагентный алгоритм может работать с любым другим ядром.

В главе 1 на рисунке 2 была представлена структура одноагентного SLAM алгоритма, которая является основой для большинства одногипотезных неграфовых алгоритмов. В ядре разработанного многоагентного алгоритма может располагаться любой одноагентный SLAM алгоритм, для которого справедлива такая схема работы. Тогда общий алгоритм будет основываться на работе ядра и использовать его выходные данные, траектории и карты.

Упрощенная схема многоагентного алгоритма показана на рисунке 3.1.

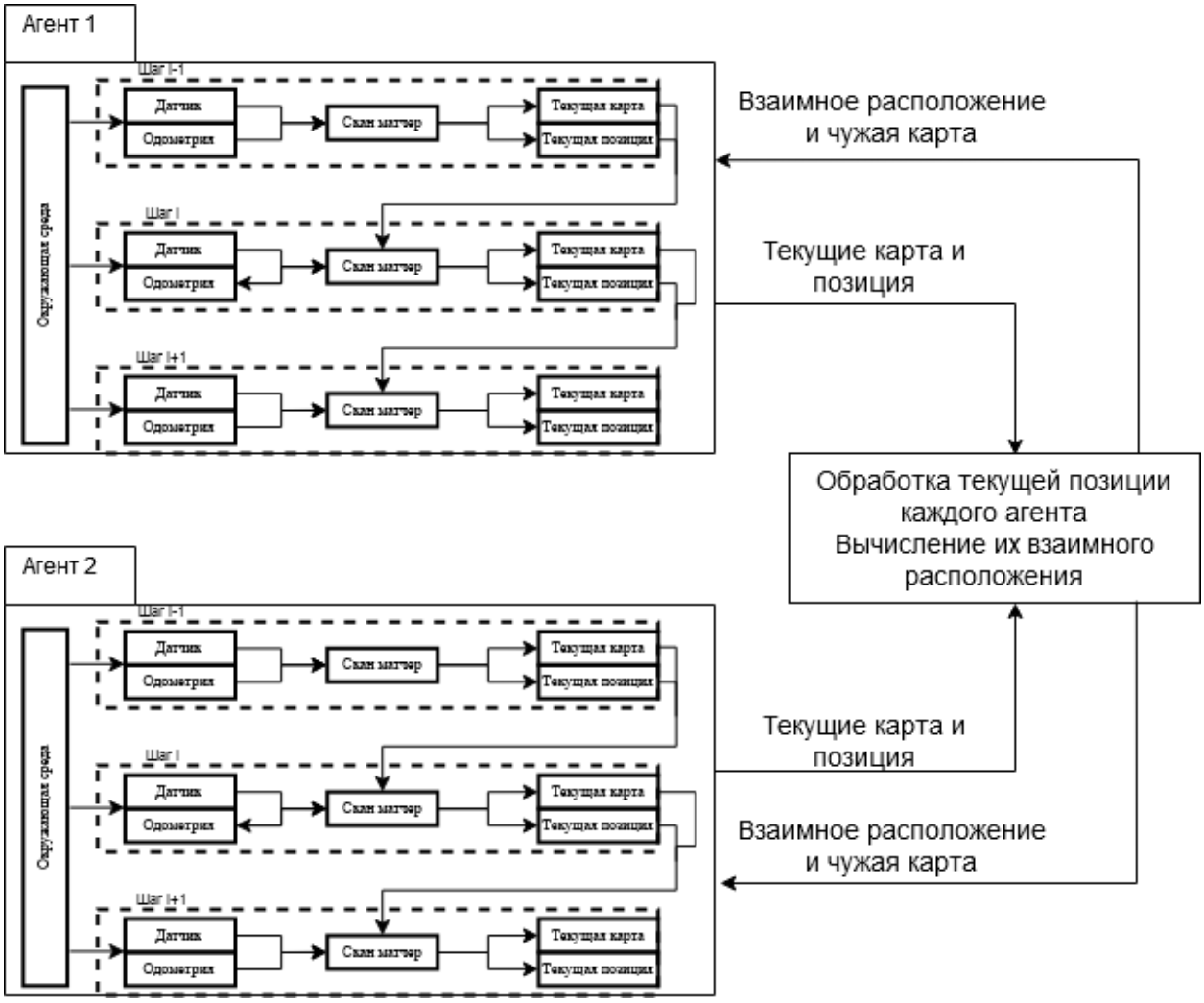


Рисунок 3.1 – Схема связи ядер, запущенных на разных агентах, друг с другом

Поскольку ядро алгоритма выполняет каждый конкретный агент, особенность многоагентного алгоритма состоит именно в методе взаимодействия между ядрами. Подробному описанию такого взаимодействия посвящена данная глава.

3.2.1 Описание особенностей ядра многоагентного алгоритма

Для того чтобы алгоритм работал устойчиво, необходимо в качестве входных данных получать данные лазерного скана и данные одометрии. Уже говорилось, что одометрия не является обязательным параметром, однако, она служит для увеличения точности. В ядре нет строгого требования на наличие одометрии, однако ее присутствие позволяет каждому агенту в отдельности совершать вычисления с большей точностью.

Одометрия (как априорная оценка позиции) вместе с новым лазерным сканом и построенной на данный момент картой передается на вход скан матчеру. Задача этого модуля в том, чтобы вычислить разницу между априорной оценкой позиции и истинной. **Истинной позицией** называется та, из которой можно произвести данный лазерный скан. Лазерный скан может противоречить карте в силу ошибок скан матчера на предыдущих шагах или в силу неполноты карты; задача – вычислить наиболее вероятную позицию скана.

Для вычисления наиболее вероятной позиции можно воспользоваться Байесовским подходом к вычислению вероятности. Тогда вероятность позиции вычисляется как средняя сумма вероятностей всех точек скана, наложенного на карту из заданной позиции. Каждая точка скана означает препятствие, и в идеальных условиях каждая точка должна попасть в занятую клетку карты. Однако недостаточно ввести бинарное разделение на занятые и свободные клетки. В реальности клетка карты может быть слишком большой, и тогда точки скана могут располагаться в ней, как показано на рисунке 3.2.

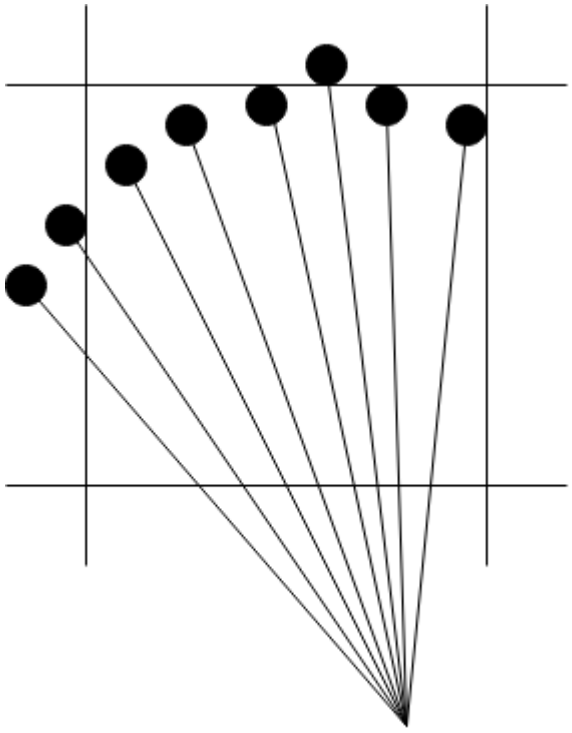


Рисунок 3.2 – Пример части лазерного скана. Много точек скана попадает на клетку, однако такую клетку нельзя считать занятой.

Логично считать, что клетка имеет некоторую вероятность быть занятой. Эта вероятность основана не только на факте, что на клетку попадает хоть одна точка лазерного скана, но и на том, как много лучей из лазерного дальномера может пройти сквозь клетку до встречи с препятствием. Такая структура клетки открывает возможность вычислять **вероятность позиции**. Вероятность позиции может быть вычислена как средняя сумма вероятностей клеток, в которые попали точки лазерного скана:

$$p_{pose}(scan) = \sum_{p \in points} o_p \cdot \omega_p \,, \tag{3.1}$$

где p – точка скана,

o_p – занятость точки p в карте,

ω_p – вес точки скана.

Вес точки скана вводится для того, чтобы сделать работу алгоритма более устойчивой в условиях коридоров. В этом случае разные позиции, расположенные вдоль направления коридора, имеют очень близкие значения вероятностей. Различие в величины вероятностей вносят точки скана, отличные от стен коридора. Обычно такие точки располагаются по ходу движения мобильной

платформы. Следовательно, точки лазерного скана, расположенные непосредственно напротив лазерного дальномера, имеют больший вес.

Задача скан матчера: вычислить позицию с наибольшей вероятностью (когда точки лазерного скана, выпущенные из этой позиции, попали на клетки карты, имеющие наибольшую вероятность быть занятыми).

Подход к вычислению такой позиции может быть произвольным. В данной работе за основу взят скан матчер Монте-Карло [21], идея которого – стохастический перебор позиций. Как только найдена позиция с большей вероятностью, чем все предыдущие, поиск начинает вестись в некотором радиусе вокруг этой позиции. Этот процесс показан на рисунке 3.3.

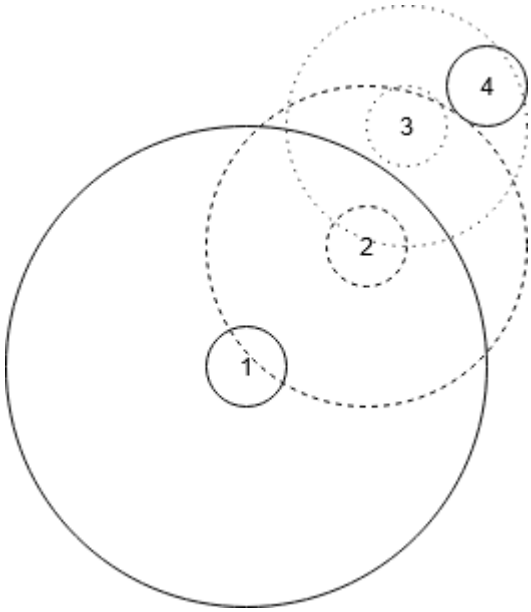


Рисунок 3.3 – Визуализация итерационного алгоритма стохастического поиска позиции с наибольшей вероятностью.

На рисунке показана последовательность шагов. Позиция 1 является априорной оценкой позиции. С некоторым радиусом вокруг нее стохастически перебираются позиции и вычисляются их вероятности. Перебор ведется до тех пор, пока не будет найдена позиция 2 с большей вероятностью, чем вероятность позиции 1. Вокруг позиции два строится радиус меньший, чем радиус вокруг позиции 1. Затем операция повторяется несколько раз. Практическое применение такого алгоритма показывает, что достаточно установить порог не для количества таких позиций, а для количества вычислений вероятностей позиций в целом.

Гарантия нахождения решения таким алгоритмом существует только в том случае, если в области стохастического поиска нет выраженных локальных максимумов вероятностей позиций. В противном случае необходимо увеличивать радиус перебора, а также количество позиций для перебора.

3.2.2 Применение теории Демпстера-Шафера для увеличения точности работы алгоритма

Согласно Байесовскому подходу в каждой ячейке карты содержится число от 0 до 1, определяющее вероятность клетки быть занятой. Чтобы вычислить эту вероятность, необходимо оценивать положение конкретных точек скана в клетке относительно точки, с которой наблюдался лазерный скан. Каждая клетка карты имеет вероятность p – быть занятой, и вероятность q – быть свободной. Эти вероятности связаны соотношением $p + q = 1$. Однако, в реальных условиях следует также ввести переменную, описывающую состояние «неизвестно». Такое состояние у клетки может быть, когда она находится в еще не исследованной области. Также клетка может быть в состоянии «неизвестно», когда два последовательных скана противоречат друг другу.

Таким образом, клетка описывается тремя вероятностями: вероятность быть занятой p, быть свободной q и быть неизвестной u. Эти три вероятности связаны между собой тождеством $p + q + u = 1$. В отличие от предыдущего подхода, где одна из вероятностей выражается через другую, этот подход позволяет выражать каждую вероятность через пару других. Встает вопрос, как объединять клетки, следуя этим правилам.

Для ответа на этот вопрос применяется теория Демпстера-Шафера [14], [57]. Чтобы строго следовать этой теории необходимо также ввести состояние конфликта в клетке карты. С естественной точки зрения конфликт – это состояние клетки не быть ни в одной из возможных состояний. Но в реальности никакая клетка карты не может быть в состоянии конфликта, а значит, эту вероятность необходимо распределять между остальными состояниями. Необходимо заметить:

если клетка находится в неизвестном состоянии, то она находится одновременно в свободном и занятом состояниях.

В контексте рассматриваемой задачи теория Демпстера-Шафера применяется во время работы скан матчера, когда необходимо выяснить вероятность клетки карты быть занятой. Сразу после работы скан матчера наступает второй этап применения этой теории: в пустые ячейки карты, которые попали в область наблюдения лидара, необходимо поместить вероятности быть свободными и занятыми. Для этого применяется правило объединения вероятностей, описываемое формулой

$$m_{1,2}(A) = \frac{1}{1-K} \sum_{B \cap C = A \neq 0} m_1(B) m_2(C), \tag{3.2}$$

где m – это вероятность состояния,

A,B,C– это различные состояния: является ли ячейка свободной, занятой или неизвестно (свободной или занятой).

K – вычисляется по формуле

$$K = \sum_{B \cap C = 0} m_1(B) m_2(C), \tag{3.3}$$

где m – это вероятность состояния,

B, C – это различные состояния.

Работа этой формулы приведена в таблице 3.1, показано слияние двух ячеек с установленными вероятностями.

Таблица 3.1. Объединение вероятностей клеток быть занятыми согласно теории Демпстера-Шафера

Вероятности	Клетка 1 (m_1)	Клетка 2 (m_2)	Объединенная клетка (m_{12})
Занята (p)	0.35	0.11	0.33
Свободна (q)	0.25	0.31	0.40
Неизвестно (u)	0.4	0.58	0.27

Более детально объяснить величину $m_{12}(p)$ можно, расписав подробнее формулу (3.2):

$$m_{12}(p) = \frac{m_1(p) \cdot m_2(p) + m_1(p) \cdot m_2(u) + m_1(u) \cdot m_2(p)}{1 - \left(m_1(p) \cdot m_2(q) + m_1(q) \cdot m_2(p)\right)}, \quad (3.4)$$

где m_1, m_2, m_{12} – это вероятности состояния клетки,

p – занятое состояние,

q – свободное состояние

u – неизвестное состояние, когда клетка одновременно и занята, и свободна.

Уже упоминалось, что такой подход объединения клеток применяется на этапе, когда необходимо сопоставить клетку карты и точку лазерного скана. Для этого сначала точку скана необходимо преобразовать в клетку с определенным распределением вероятности. Вид распределения – это отдельная исследовательская задача, выходящая за рамки данной работы. В данной работе вероятность свободного состояния ячейки прямо пропорциональна длине отрезка, который преодолевает луч лазерного дальномера, проходя сквозь клетку, до момента встречи с препятствием.

Таким образом, можно ввести понятие **занятости точки скана**. На рисунке 3.4 показана одна точка скана, наложенная на ячейку карты. Занятость точки равна отношению длины отрезка b к длине $a+b$.

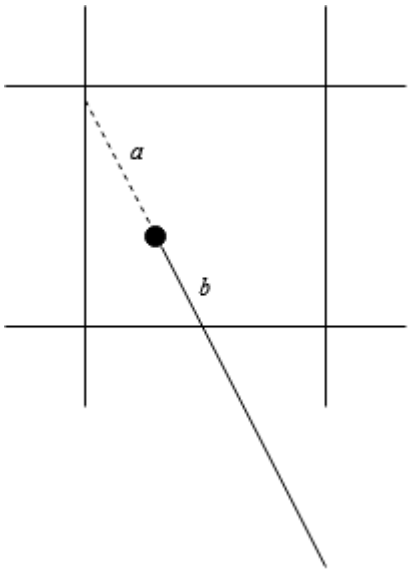


Рисунок 3.4 – Вычисление занятости точки лазерного скана, попавшей в клетку карты, через отношение $b/(a+b)$

3.2.3 Обоснование увеличения точности работы алгоритма SLAM при использовании теории Демпстера-Шафера

Этот раздел посвящен анализу значений ячеек карты в случае, когда они вычисляются согласно Байесовской теории и теории Демпстера-Шафера.

Сначала следует определить ситуацию, когда характеристики ячейки карты могут изменяться. В случае Байесовской теории ячейка содержит одно число – вероятность быть занятой. После завершения работы скан матчера, когда вычислена истинная позиция агента, наблюдающего конкретный лазерный скан, необходимо обновить это число для каждой ячейки. Тогда можно в одной системе координат, центр которой находится в текущей позиции агента, разместить и текущую карту, и текущее наблюдение. В этот момент скан будет наложен на карту так же, как и на этапе работы скан матчера. Однако, теперь нет необходимости вычислять стоимость такого наложения. Нужно просто обновить карту в соответствии с таким наложением скана и проверить, в каких клетках карты находятся точки лазерного скана. Далее необходимо изменить значение занятости в каждой клетке согласно тому, как именно точка попадает в клетку. Наиболее очевидным является способ обновления ячейки вычислением среднего значения между характеристиками занятости клетки и занятости точки. Очевидным недостатком такого подхода будет являться высокая чувствительность ячейки карты к погрешностям лазерного дальномера. На рисунке 3.5 приведен график изменения характеристики занятости клетки в зависимости от сильно колеблющихся характеристик занятости точек скана. На графике видно, что при сильном колебании изменения занятости точки, занятость клетки сильно колеблется. Такое колебание занятости точки может быть связано с шумом лазерного дальномера, когда точка оказывается то на ближней, то на дальней границе клетки.

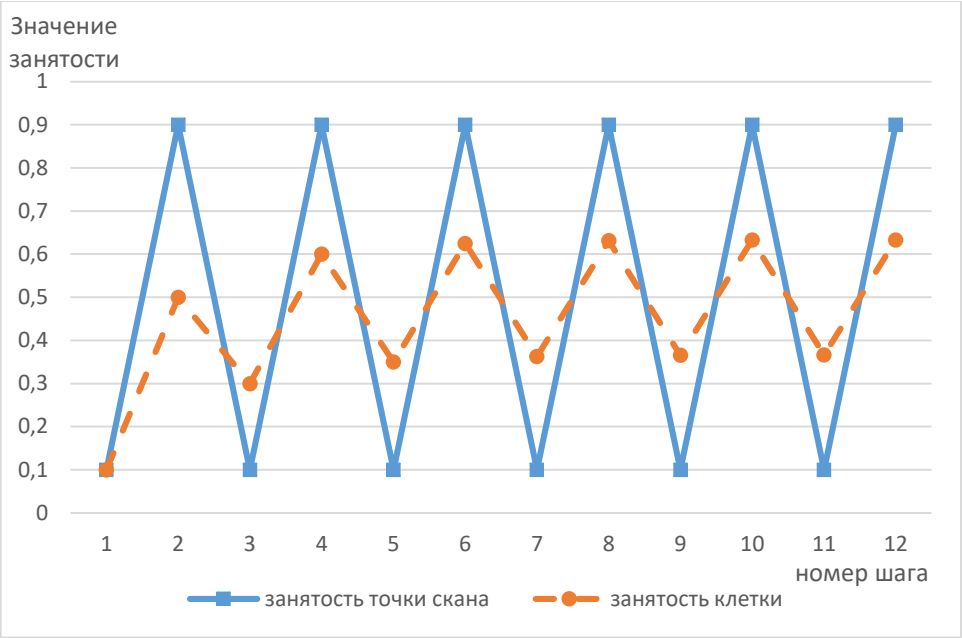


Рисунок 3.5 – Колебание занятости клетки в зависимости от занятости точки при обновлении занятости согласно среднему значению

Недостаток такого подхода также проявляется в ситуации, когда только одно из наблюдений является промахом (рисунок 3.6).

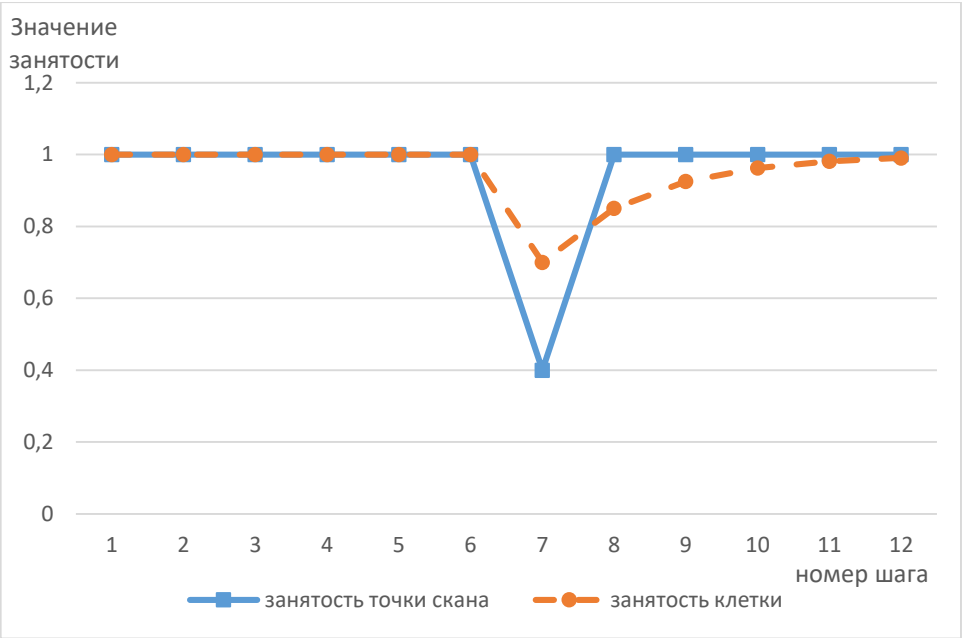


Рисунок 3.6 – Демонстрация чувствительности занятости клетки к промахам наблюдений

Для того чтобы избавиться от такой высокой локальной чувствительности, но сохранить адекватность модели, применяется оценка занятости ячейки согласно среднему арифметическому всех значений, накопленных на предыдущих шагах. Исходя из последовательности значений занятости точки скана, занятость клетки за n шагов вычисляется по формуле:

$$p_c = \frac{1}{n}\sum_{i=1}^{n} p_i , \tag{3.5}$$

где p_c – занятость клетки карты,

p – занятости точки скана,

n – количество наблюдений.

Такой подход уменьшает чувствительность к промахам дальномера, и график зависимости занятости клетки карты от занятости точки при наличии промаха сглаживается сильнее (рисунок 3.7).

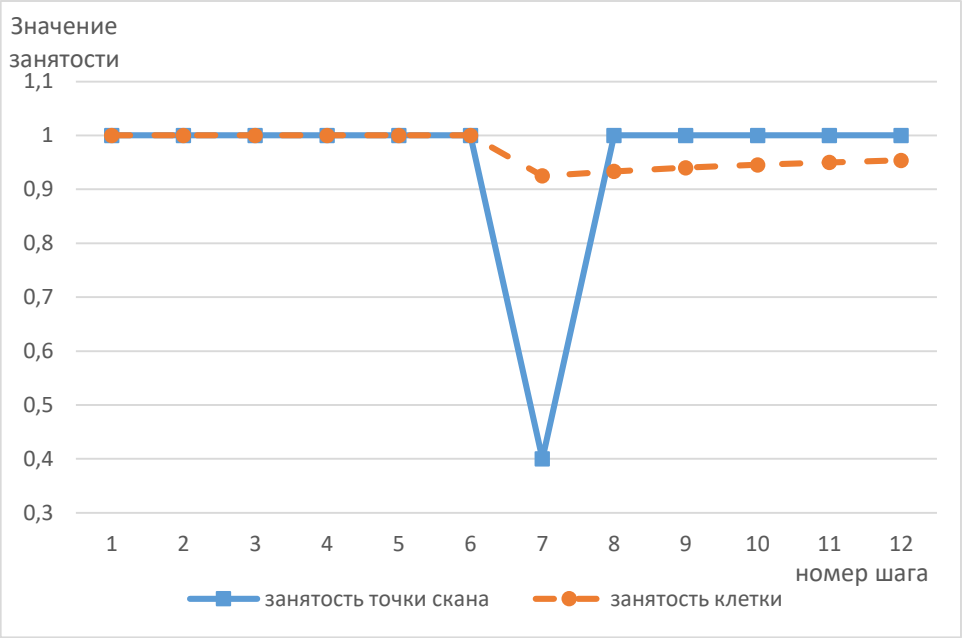


Рисунок 3.7 – Занятость клетки при наличии промаха при использовании среднего значения занятости всех поступивших ранее точек

Несмотря на то, что чувствительность к промаху стала ниже, видно, что к значению 1 вероятность клетки быть занятой устремляется не сразу и не очень быстро. Решение такой проблемы предлагает теория Демпстера-Шафера. При ее использовании необходимо заново определить понятие занятости точки. Теперь каждая клетка содержит вероятности быть свободной, занятой, а также быть одновременно и свободной, и занятой (то есть быть в неопределенном состоянии). Занятость клетки в таком случае – это совокупность трех компонент, а классическая вероятность клетки быть занятой вычисляется по формуле

$$p = \frac{p_{occ}}{p_{occ} + p_{free}} \cdot p_{unknown} + p_{occ} , \tag{3.6}$$

где p – вероятность клетки карты быть занятой в классическом смысле,

p_{occ} – вероятность клетки быть занятой, согласно теории Демпстера-Шафера,

p_{free} – вероятность клетки быть свободной, согласно теории Демпстера-Шафера,

$p_{unknown}$ – вероятность клетки быть неизвестной, согласно теории Демпстера-Шафера.

Понятие занятости точки скана тоже необходимо дополнить. При этом для точки скана необходимо использовать три состояния. Обновление занятости клетки карты – это операция объединения двух множеств согласно правилу, описанному формулой 3.2. Правило вычисления занятости точки выведено, исходя из эвристики. В разработанном алгоритме вероятность точки быть неизвестной всегда полагается 0,05, а вероятность быть занятой и свободной вычисляется по правилу, описанному формулой 3.6.

Согласно описанному подходу выброс в наблюдениях будет обработан в соответствии с рисунком 3.8.

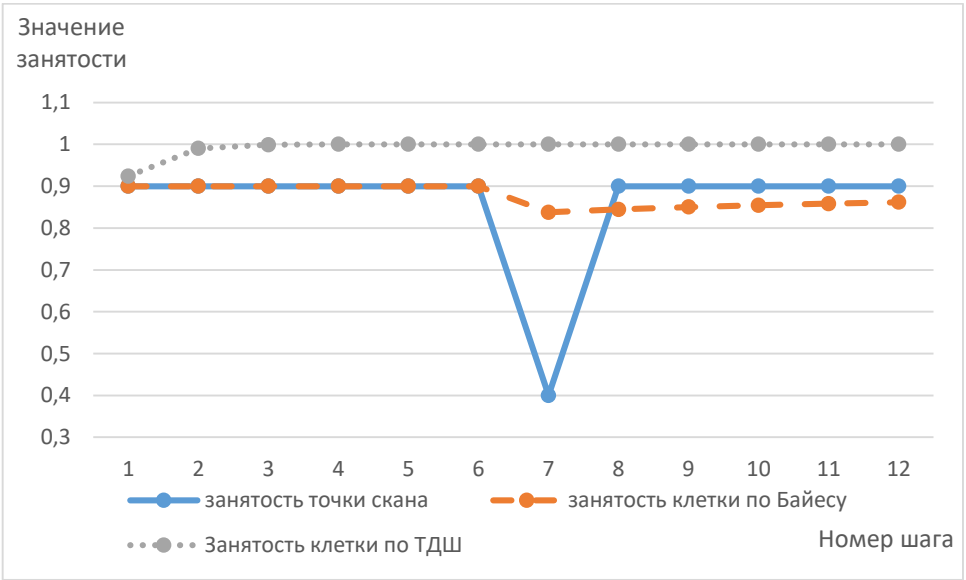


Рисунок 3.8 – Сравнение занятости клетки при использовании подхода Байеса и Демпстера-Шафера

Как видно на графике само по себе значение занятости при Байесовском подходе не превышает 0,9, поскольку занятость точки 0,9. Однако подход Демпстера-Шафера позволяет повысить это значение до 1, если объединяются

события, имеющие близкие друг к другу вероятности множества состояний. Наглядный пример приведен в таблице 3.2.

Таблица 3.2 Объединение сильно коррелирующих занятостей клетки и точки скана по Демпстеру-Шаферу

Вероятности	Клетка карты	Точка скана	Результат объединения
p_{occ}	0.06	0.04	0.008
p_{free}	0.9	0.9	0.99
$p_{unknown}$	0.04	0.06	0.002

По данным таблицы можно сделать вывод, что объединение множеств по Демпстеру-Шаферу можно понимать, как определение степени согласованности этих множеств. На графике (рисунок 3.8) можно заметить, что такое правило объединения устойчиво к промахам. Если резкое изменение значения не будет единичным, то модель адекватно отреагирует на подобное изменение. На рисунке 3.9 представлен график со значениями занятости точки скана и реакции на них Байесовской модели и модели Демпстера-Шафера.

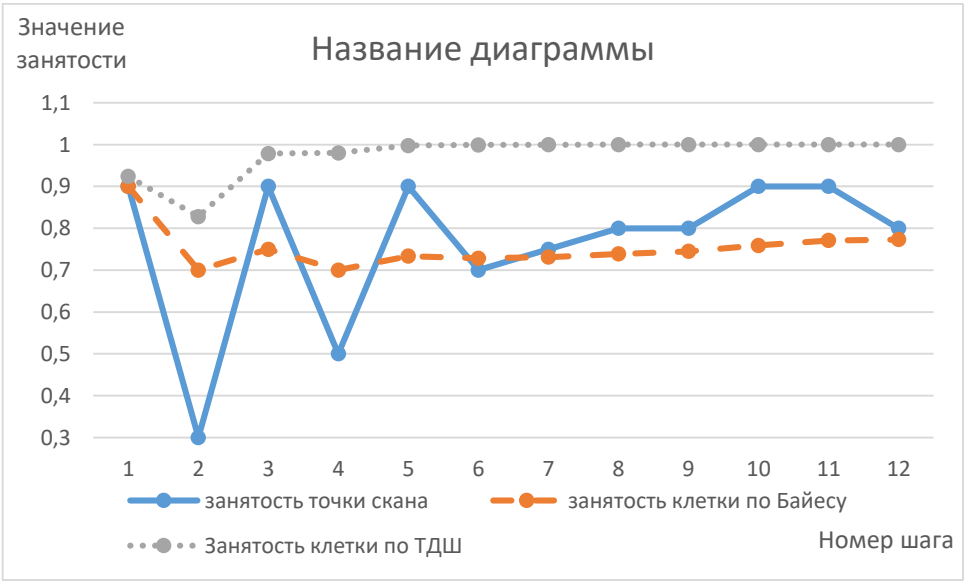


Рисунок 3.9 – Сравнение занятости клетки при использовании подхода Байеса и Демпстера-Шафера в случае не полностью занятой клетки

Если значения занятости точки скана довольно высокое (больше, чем 0,5) в течение некоторого количества шагов, то занятость клетки тоже постоянно

увеличивается. Очевидно, что если несколько раз подряд измерение будет указывать на то, что клетка карты свободна, то и занятость клетки по Демпстеру-Шаферу будет стремиться к нулю.

3.3 Описание многоагентного алгоритма SLAM

Отличительной особенностью предлагаемого алгоритма является отсутствие знания у каждого агента о других агентах в произвольный момент времени. Это позволяет системе сохранять работоспособность, когда один или несколько агентов выходят из строя, а также, когда в системе остается лишь один агент. Обмен информацией между агентами происходит только в том случае, когда два агента находятся в непосредственной близости относительно друг друга.

На основании описанных методов и гипотез можно построить алгоритм, который решает поставленную задачу для стаи роботов. Алгоритм является расширением одноагентного лазерного одногипотезного алгоритма, использующего в качестве карты сетку занятости. Каждая ячейка помимо вероятности быть свободной также содержит вероятность быть занятой и быть в состоянии неизвестности. При этом эти вероятности подчиняются теории Демпстера-Шафера. Схема такого многоагентного алгоритма представлена на рисунке 3.10.

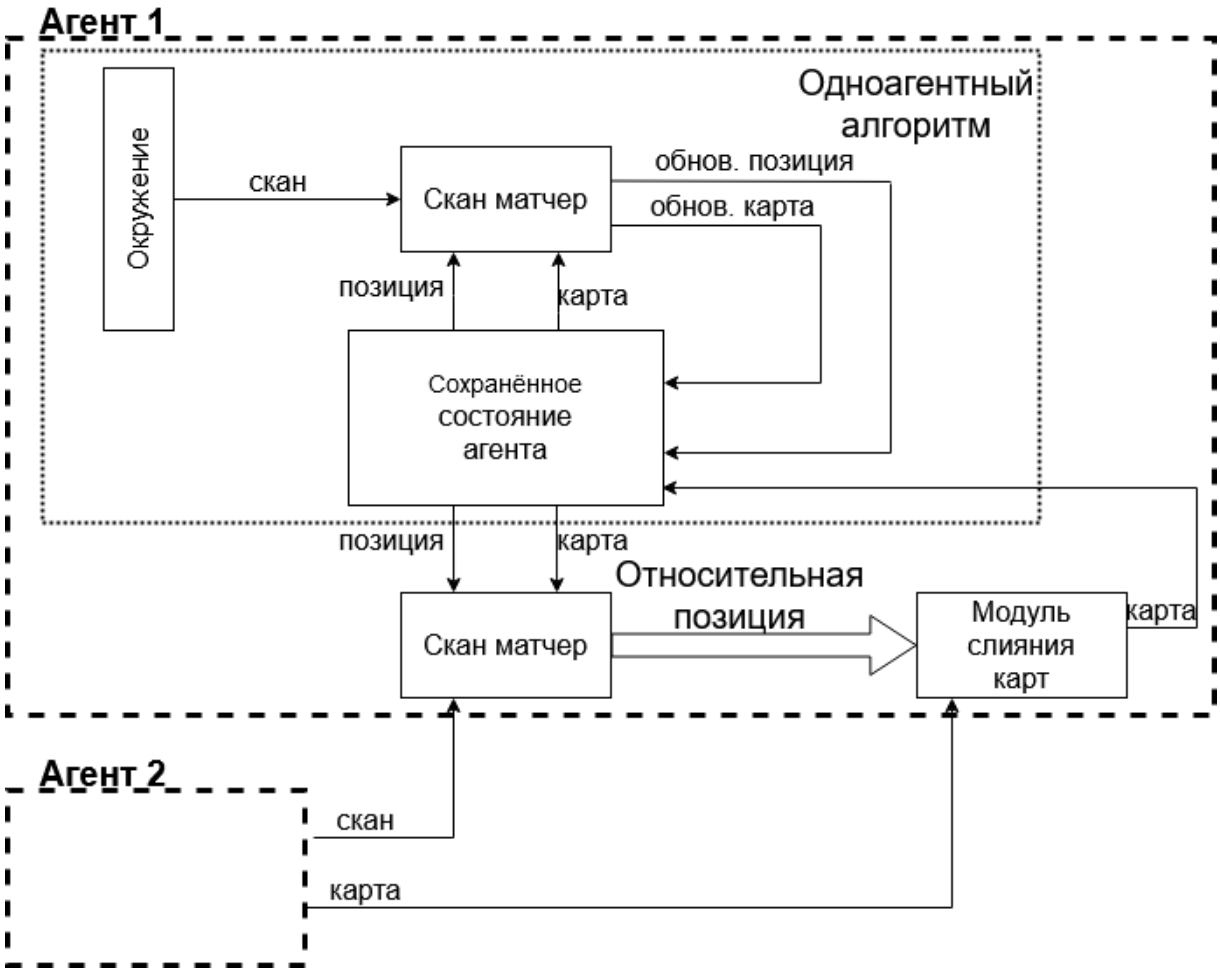


Рисунок 3.10 – Схема многоагентного алгоритма решения задачи SLAM

Рассмотрим алгоритм с точки зрения одного конкретного агента, который решает задачу SLAM, как будто действует в одноагентной системе. Агент владеет только собственными вычислительными ресурсами, которые направлены на построение карты и определение на ней собственного положения. В случае, когда в поле зрения агента попадает другой агент, они начинают процесс обмена всеми накопленными за прошедшее время знаниями. Нужно определить, что другой агент попал в поле зрения, не используя данные сенсоров; иначе потребуется маркировать робота, чтобы он отличался от окружающей среды. А поскольку алгоритм должен работать в ситуации отсутствия предварительных знаний об окружающей среде, то подход, требующий маркировки робота, не применим.

Следовательно, определить, что один агент находится в прямой зоне видимости другого, необходимо посредством некоторого отдельного сенсора. Выбор такого сенсора находится вне рамок данной работы, но как пример можно

использовать технологию передачи данных bluetooth, дальность действия сигнала которой как раз не более нескольких метров. Также можно использовать сигнал wifi и на основании силы сигнала определять расстояние между агентами. Можно снабдить агентов радиодатчиками и вычислять расстояние между ними с помощью радиосигнала.

Когда агенты находятся в непосредственной близости, они могут начинать процесс обмена построенными картами. Эти карты обязательно будут иметь общую часть, поскольку передача данных происходит, когда агенты практически находятся на одном месте и наблюдают одну и ту же часть окружения. Кроме карт агенты передают друг другу текущий лазерный скан, что позволяет применить алгоритм скан матчера, аналогичный работающему в ядре каждого агента при решении обычной задачи SLAM.

Таким образом, агент получает лазерный скан от другого агента. При помощи скан матчера он накладывает этот скан на собственную карту и определяет, на каком расстоянии агенты находятся относительно друг друга. Рассмотрим более подробно применимость алгоритма скан матчера для определения такого относительного положения. В общем случае скан матчер принимает на вход априорную позицию агента, текущий лазерный скан и построенную карту. В качестве выходных параметров для вычисления взаимного расположения выступает вектор, на который необходимо изменить априорную оценку позиции, чтобы вычислить ее апостериорную оценку. Другими словами, скан матчер вычисляет разницу между позицией, полученной на входе, и позицией, с которой в действительности можно увидеть полученный лазерный скан, при условии полученной карты.

Такое описание работы скан матчера предполагает, что при передаче ему в качестве входных данных лазерного скана, полученного от другого агента, в качестве выходных данных будет получено расстояние между агентами. При этом должно выполняться условие, что часть окружения, запечатленная на скане, присутствует в карте. Выполнение этого условия – гарантировано. Более подробно алгоритм скан матчера будет рассмотрен в процессе дальнейшего анализа.

Определив взаимное расположение агентов, можно приступать к объединению карт. Для этого второй агент передает первому построенную карту. При условии, что

обе полученные карты не содержат ошибок, слияние карт – это прямолинейная операция, где достаточно перебрать каждую ячейку обеих карт и выбрать или наибольшее, или наименьшее, или среднее значение занятости соответствующих клеток. Природа алгоритма, заложенного в ядро каждого агента, допускает в процессе построения карты возникновение ошибок, которые никогда не будут исправлены в будущем. В частности, одноагентный SLAM алгоритм может не быть многогипотезным или графовым. Тогда карта, построенная тем или другим агентом, может содержать ошибки. Не известно также, содержится ли одна и та же ошибка только в одной карте или в обеих. Далее рассмотрим проблему объединения карт, содержащих ошибки.

3.3.1 Алгоритм определения взаимного расположения агентов

В ядре одноагентного SLAM алгоритма, как уже отмечалось, находится скан матчер Монте-Карло, используемый для вычисления позиции, с которой можно наблюдать определенный лазерный скан. Достоинство скан матчера Монте-Карло в том, что он не перебирает все возможные позиции, а исходит из предположения, что все позиции, обладающие наибольшей вероятностью, сосредоточены в одном месте, Проще говоря, а распределение вероятностей позиции близко к нормальному закону (рисунок 3.11).

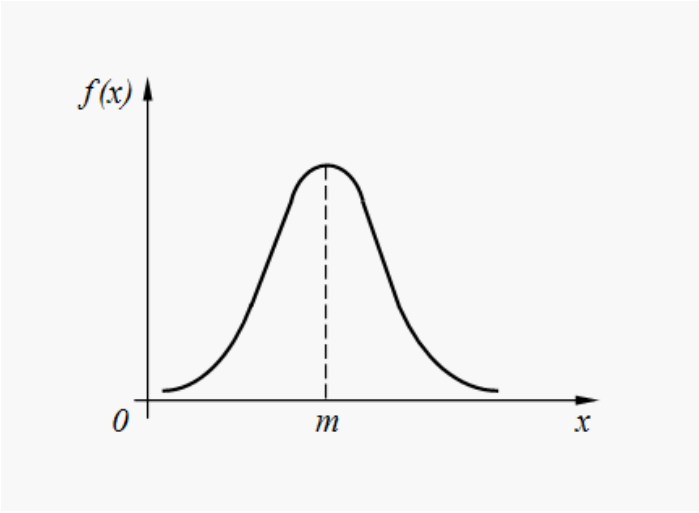


Рисунок 3.11 – Распределение вероятностей позиций, из которых можно наблюдать лазерный скан вокруг истинной позиции m.

Можно сделать вывод, что если при стохастическом поиске была найдена позиция, вероятность которой выше, чем у априорной оценки позиции, то в дальнейшем поиск нужно будет вести в области новой позиции.

Очевидный недостаток такого подхода: в реальности такое распределение вероятностей может не соблюдаться. Однако, если истинная позиция агента достаточно близка к априорной оценке, то использование такого стохастического алгоритма возможно. В случае если истинная позиция находится далеко от априорной оценки, использование такого алгоритма может привести к неверному результату, поскольку он может сойтись к локальному экстремуму, а не к глобальному.

В рассматриваемом многоагентном алгоритме скан матчер будет применяться тогда, когда позиции агентов будут различаться на несколько метров, а значит использование стохастического скан матчера невозможно. Простейшим выходом из данной ситуации является скан матчер, который вычисляет вероятности всех позиций в некоторой области поиска в пространстве (x, y, θ). Это очень долгий перебор, поэтому использование такого скан матчера в условиях ограниченных ресурсов нецелесообразно.

В предлагаемом алгоритме используется скан матчер ветвей и границ, идея которого описана в [20]. Для вычисления вероятности той или иной позиции необходимо наложить лазерный скан, «исходящий» из этой позиции, на карту. При этом необходимо вычислить корреляцию скана и карты. Другими словами, необходимо проверить, насколько точки скана соответствуют занятым ячейкам карты. Если при вычислении корреляции выяснится, что количество несовпадающих точек превысило значение, полученное предыдущей позицией, то дальнейшее вычисление корреляции бессмысленно.

Поэтому вместо вероятности позиции имеет смысл рассматривать вероятность ошибки позиции. Предлагается вычислять ее следующим образом:

$$pose\_error(scan) = \sum_{p \in scan} e^{-o_p} \cdot \omega_p , \qquad (3.7)$$

где p – точка скана,

o_p – занятость точки p в карте,

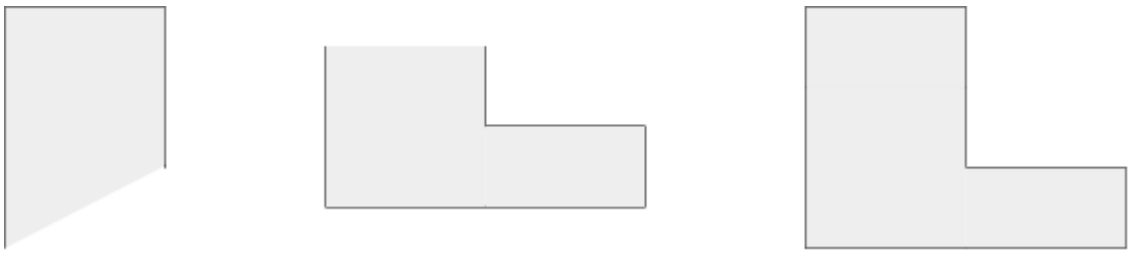
ω_p – вес точки скана.

Если становится очевидно, что вероятность ошибки такой позиции превышает вероятность ошибки уже вычисленной ранее позиции, то при оценивании вероятности очередной позиции можно остановить вычисления.

Важным фактором является порядок просмотра позиций, вероятности которых будут вычислены. Скан матчер ветвей и границ перебирает позиции по спирали, начиная от априорной позиции. Используя метод ветвей и границ на дуге спирали можно найти области с наименьшей вероятностью ошибки. Поиск в этом случае будет проводится в направлении позиций с наименьшей ошибкой. А поиск среди остальных позиций будет производиться в последнюю очередь. Когда позиция с очень низкой вероятностью ошибки найдена, скан матчер перебирает позиции в небольшой окрестности с уменьшенным шагом, чтобы нивелировать ошибку дискретизации. Эксперименты показали, что такой скан матчер работает на 24% быстрее, чем обычный.

3.3.2 Алгоритм объединения карт

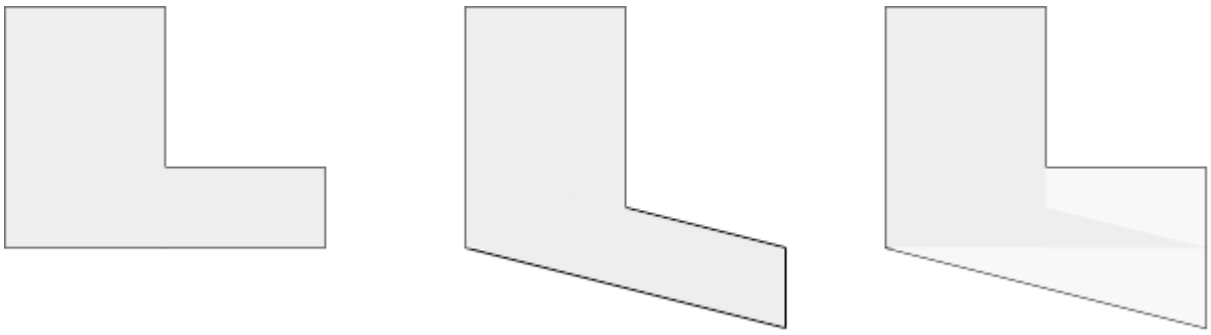
После того, как относительная позиция агентов вычислена, возникает вопрос о слиянии карт, построенных разными агентами. В случае если обе карты построены без ошибок, слияние выполняется прямолинейно с использованием полусуммы вероятностей занятости каждой клетки карты или формулы объединения вероятностей по Демпстеру-Шаферу. Пример такого слияния показан на рисунке 3.12.



а) б) с)

Рисунок 3.12 – а) карта, построенная первым агентом; б) карта, построенная вторым агентом; с) результат слияния этих карт, не содержащий конфликт

Если при построении одной или обеих карт была допущена ошибка, то результаты могут быть неверными. Например, на некоторой итерации скан матчер не нашел позицию, обладающую максимальной вероятностью и вернул другую позицию. В этом случае неверно вычисленная позиция повлечет за собой неверное встраивание скана в карту. Может возникнуть ситуация, показанная на рисунке 3.13.



а) б) с)

Рисунок 3.13 – а) карта, построенная первым агентом; б) карта, построенная вторым агентом; с) результат слияния этих карт, содержащий конфликт

Чтобы решить проблему конфликта в картах было принято решение воспользоваться методами анализов изображений. Карта занятости очевидно трактуется как изображение. Тогда на ней можно искать особые точки и их дескрипторы согласно алгоритмам ORB [48], SURF [7], SIFT [50] и прочим. Если на двух картах (рисунок 3.13) выделить особые точки, то будет видно, что некоторая часть точек, дескрипторы которых совпадают, совпадает и по расположению, а другая находится на некотором ненулевом смещении. Однако это смещение одинаково для всех таких точек.

Графически связь точек, дескрипторы которых совпадают для данной пары карт, видна на рисунке 3.14.

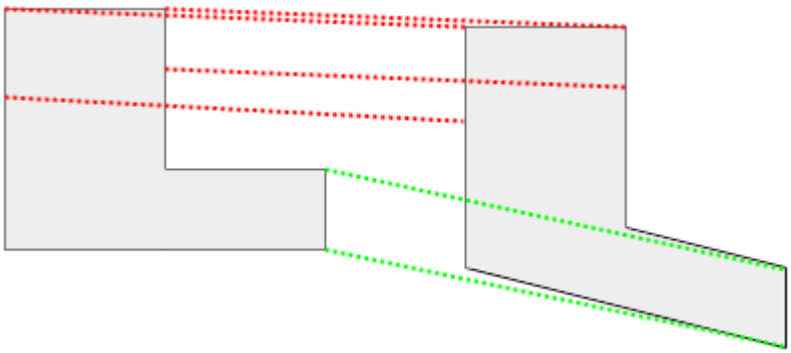


Рисунок 3.14 – Пример связи дескрипторов особых точек на картах

На рисунке 3.14 зеленым цветом обозначен кластер (англ. cluster – скопление – объединение нескольких однородных элементов, которое может рассматриваться как самостоятельная единица, обладающая определенными свойствами) точек, дескрипторы которых совпали. Видно, что особые точки разделены на два кластера с разным относительным смещением. Наблюдателю ясно: чтобы получить консистентную карту – красные точки необходимо оставить на месте, а зелёные – сместить.

3.4 Выводы по третьей главе

В главе 3 представлен алгоритм многоагентного решения задачи SLAM, удовлетворяющий требованиям, сформулированным в конце главы 2. Для увеличения точности работы алгоритма в качестве альтернативы Байесовской теории объединения вероятностей используется теория Демпстера-Шафера. Слияние карт двух агентов происходит, когда они находятся вблизи друг от друга. Это позволяет повторно использовать некоторые компоненты одноагентного алгоритма для определения взаимного расположения. Для объединения карт также можно воспользоваться формулами объединения вероятностей по Демпстеру-Шаферу. Если карты содержат конфликт, то выявить его можно при помощи алгоритмов для задач анализа изображений, а также определить согласующиеся участки карты и использовать для объединения карт только их. Такой подход позволяет удалить из карт область, содержащую конфликт.

4 Масштабируемый алгоритм фильтрации двумерных лазерных сканов

Данная глава посвящена описанию фильтра, выполняющего предобработку входных данных для описанного в главе 3 многоагентного масштабируемого алгоритма решения задачи SLAM. Алгоритм впервые представлен в [31], рассмотрим его более подробно.

У лазерных дальномеров, являющихся поставщиками входных данных, есть общий недостаток: они собирают одновременно слишком мало и слишком много информации. С одной стороны, данных мало, потому что невозможно сгладить или аппроксимировать их без значительной потери точности; с другой стороны, много, так как требуется большое количество памяти для хранения и обработки данных современных лазерных сканов, которые снимаются более 30 раз в секунду. Возникает необходимость разработки такого фильтра. Обычно не нужно снимать лазерные сканы так часто, только если сканер не установлен на автомобиле, движущемся со скоростью 60 км / ч. В этом случае окружающая среда может сильно изменяться за 0,03 секунды. С другой стороны, если робот движется в помещении и имеет среднюю скорость около 0,5 - 1 м / с, такое количество плотных облаков точек от лазерного дальномера является избыточным. Идея фильтра основана на сохранении нескольких последовательных сканов в скользящем окне и сравнении каждого нового скана со сканами из этого окна. Если каждый новый скан сильно коррелирует с каждым сканом из окна – его следует отбросить.

Важной особенностью этого алгоритма является то, что процесс фильтрации требует меньшего вычислительного времени, чем процесс скан матчинга в SLAM алгоритме. В противном случае использование этой фильтрации было бы неуместным. В среднем скан матчер, обрабатывающий сканы в реальном времени, должен работать примерно со скоростью O(100 n), где n – число точек в скане. Предлагаемый метод фильтрации работает значительно

быстрее, что в последующем будет показано как математически, так и экспериментально.

Горизонтальное масштабирование алгоритма осуществляется за счет распределения вычислений самого трудоемкого этапа фильтрации – предобработки входных данных. Входными данными является лазерный скан, который может содержать от десятков тысяч до десятков миллионов точек, которые возможно обрабатывать параллельно.

4.1 Методы уменьшения размерности лазерного скана

Основная идея разработанного алгоритма заключается в сравнении текущего лазерного скана с предыдущим. Если новый скан подобен предыдущему, то он обрабатываться не должен; а во избежание шума в наблюдениях, следует сравнивать текущий скан с несколькими предыдущими вместо одного. Таким образом, появляется **скользящее окно сканов**, которое способно оценивать каждое новое входящее сканирование.

Обычно лазерный скан состоит из нескольких тысяч точек. Для вычисления корреляции сканов методом перебора требуется $O(n^2)$ операций, что может превышать миллион итераций. Для уменьшения этого количества можно выделить особые точки на скане, на что уйдет много времени. Вместо использования необработанных данных лазерного скана для расчета корреляции предлагается построить гистограмму для каждого скана. Есть несколько подходов к созданию такой гистограммы. Один из них основан на делении по диапазонам расстояний, а другой – по диапазонам углов.

Опишем метод **построения гистограммы деления по диапазонам расстояний**. Для каждого скана известны максимальное и минимальное значения расстояния до препятствий. Следовательно, можно разделить этот разброс диапазонов на несколько интервалов, а затем вычислить количество точек, соответствующих каждому интервалу. Два последовательных сканирования обычно не должны существенно различаться, поэтому гистограммы расстояний должны быть близки друг к другу. На практике, если робот не вращается, разница

между двумя сканами незначительна, и значения каждого столбца гистограммы мало изменяются. Если робот вращается, разница более значительна. Однако можно обновить подход: вместо расчета количества точек в каждом столбце можно рассчитать медианное значение расстояний для каждого интервала. Таким образом, две последовательные гистограммы становятся более разнообразными.

Подход к построению **гистограмм деления по диапазонам углов** является противоположным делению по расстояниям. Каждый лазерный скан снимается в полярных координатах, где угол является второй степенью свободы в дополнение к расстоянию. Следовательно, можно разделить каждый скан на несколько интервалов по углам и вычислить средние величины в этих интервалах. Подсчитывать количество точек в этом случае бессмысленно, потому что оно одинаково для каждого интервала. Также возможно вычислить дисперсию вместо среднего значения.

Оба подхода к созданию гистограммы уменьшают размерность входных данных. Вместо обработки тысяч точек лазерного скана для дальнейшего расчета корреляции можно обработать только несколько десятков столбцов в гистограммах. Важно отметить, что лазерный скан не заменяется навсегда гистограммой для этого алгоритма SLAM. Гистограмма строится только для расчета корреляции сканов.

Эксперименты показывают, что корреляция гистограмм в каждом окне вычисляется быстрее, чем скан матчер находит наилучшее положение для скана. Следовательно, можно отфильтровать лишнее сканирование и сэкономить дополнительные временные ресурсы для других процедур. Конкретные числа представлены в разделе 4.5.

4.2 Критерии корреляции

Следующим шагом после создания гистограмм каждого скана является вычисление их корреляции. Здесь можно воспользоваться методами математической статистики и рассматривать гистограмму как случайную величину с неизвестным распределением. Поскольку все гистограммы из окна в

целом похожи друг на друга, можно предположить, что распределение одинаковое.

Существует несколько известных способов для вычисления корреляции случайных величин: корреляция Пирсона, Спирмана и Кендалла. Самым простым является коэффициент корреляции Пирсона. Он вычисляется по формуле

$$P_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}, \tag{4.1}$$

где X, Y – некоторые случайные величины,

cov – ковариация случайных величин,

σ – дисперсия случайной величины.

Если случайная величина состоит из n наблюдений, а x_i – наблюдаемое значение этой величины на i-м шаге, то коэффициент Пирсона можно вычислить по следующей формуле:

$$P_{X,Y} = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{\sqrt{\left[n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2\right]\left[n\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2\right]}} \quad . \tag{4.2}$$

Значение этого коэффициента находится в диапазоне от минус 1 до 1. Значение 1 – полная положительная линейная корреляция, 0 – отсутствие линейной корреляции, а минус 1 – полная отрицательная линейная корреляция. Эта корреляция называется линейной из-за следующей геометрической интерпретации. По абсциссе графика следует поместить значение первой переменной, по ординате – значение второй переменной. Если точки с результирующими координатами принадлежат одной прямой с положительной производной, то их корреляция равна 1. Если производная отрицательна, то коэффициент Пирсона равен минус 1. Если нет возможности провести какую-либо линию, тогда коэффициент равен 0.

Коэффициенты Кендалла и Спирмена используются для измерения упорядоченной связи между двумя измеренными величинами. Это мера ранговой корреляции: схожесть порядка данных при упорядочивании по каждой из величин. Коэффициент Спирмена определяется как коэффициент Пирсона между упорядоченными переменными. Другими словами, если X и Y являются случайными величинами с порядкми rgX, rgY, то коэффициент корреляции рассчитывается как

$$P_{rgX,rgY} = \frac{\operatorname{cov}(rgX, rgY)}{\sigma_{rgX}\sigma_{rgY}} \quad . \tag{4.3}$$

Расчет коэффициента Кендалла связан с количеством совпадающих пар случайных величин. Пусть (x_i, y_i) – набор наблюдений случайных величин X и Y. Пары (x_i, y_i) и (x_j, y_j), где $i < j$, называются согласованными, если выполняются оба условия $x_i > x_j$ и $y_i > y_j$ или наоборот $x_i < x_j$ и $y_i < y_j$. В противном случае их называют несогласованными.

Коэффициент корреляции Кендалла вычисляется следующим образом:

$$P_{X,Y} = \frac{\textit{число согласованных пар} - \textit{число несогласованных пар}}{C_n^2}. \tag{4.5}$$

Подводя итог, можно выделить три хорошо известных подхода к вычислению корреляции. Главный недостаток коэффициента Кендалла – алгоритмическая сложность. Для этого требуется вычислить ранг случайной величины, а затем вычислить количество согласованных пар. В худшем случае это может потребовать Nlog (N) операций. Коэффициент Спирмена менее сложен, но он также требует введения отношения порядка. Поскольку корреляция вычисляется для гистограмм последовательных сканов, правильное ранжирование значений в гистограммах является сложной задачей. Для похожих в целом гистограмм необходимо фиксировать каждое небольшое колебание значений. Следовательно, функция порядка должна быть чувствительной к этим колебаниям и одновременно показывать истинную корреляцию. Поэтому коэффициент корреляции Пирсона является наиболее подходящим для рассматриваемого

алгоритма. Его сложность составляет O(n), он не требует функции порядка и достаточно чувствителен к колебаниям значений в гистограммах.

4.3 Параметры и константы в алгоритме фильтрации сканов

В предложенном алгоритме есть четыре параметра, тонкой настройки которых необходимо уделить внимание:

- количество колонок в гистограмме и, следовательно, количество точек лазерного скана в каждой колонке;
- размер скользящего окна;
- порог внутриоконной корреляции P_{pair} (насколько новый скан должен коррелировать с каждым сканом в окне);
- порог отбрасывания – значение общей корреляции скана со сканами в окне P_{common}.

Эти четыре параметра влияют на количество и характер отфильтрованных сканов. Первый из них – это количество столбцов в гистограмме. У всех рассматриваемых гистограмм есть общая особенность: чем больше столбцов содержится в гистограмме, тем больше деталей обрабатывается для каждого скана. Рассмотрим лидар, который используется в наборе данных Массачусетского технологического института, который захватывает лазерные сканы, состоящие примерно из 1000 точек. Разделение этих точек на 50 столбцов означает, что каждая группа точек содержит в среднем 20 точек. Разделение на 10 столбцов приводит к созданию групп по 100 точек в каждой.

Чтобы определить влияние количества столбцов на количество информации, которое можно получить из гистограммы, можно рассмотреть две граничные ситуации. Первая ситуация подразумевает, что каждая точка скана содержится в отдельном столбце. Следовательно, гистограмма состоит из 1000 столбцов. Корреляция между двумя такими гистограммами очень чувствительна к каждой точке и даже к шуму, но она может показать истинное сходство лазерного скана. С другой стороны, все точки можно сгруппировать в один столбец. В этом

случае все сканы станут похожими, и не будет возможности отличить один от другого.

Этот пример показывает, что большее количество столбцов приводит к большей чувствительности корреляции сканов к незначительной разнице в сканах. Несмотря на интуитивное мнение о том, что чем выше чувствительность, тем выше точность, в реальных данных высокая чувствительность может быть наоборот ошибочной. Например, если появляется движущийся объект в области обзора лазерного скана, это неизбежно приводит к разнице в двух последовательных сканах. Более того, каждый датчик вносит шум в наблюдения, и иногда (на небольших расстояниях) этот шум может ошибочно интерпретироваться как разница между сканами. Результаты экспериментов, представленные в разделе 4.5, показывают, что для 1000 точек лазерного сканирования, угол обзора которых составляет $3\pi/4$, следует сгруппировать в 15 или 30 столбцов.

Еще одна важная константа – это размер окна, в котором находятся предыдущие лазерные сканы. Оценка корреляции текущего скана равна произведению значения корреляции каждого скана из этого окна и текущего скана. За величину корреляции принят коэффициент корреляции Пирсона. Очевидно, что если робот с лазерным сканером движется быстро, то большое количество разных сканов в окне уменьшает итоговую оценку корреляции нового скана. Это означает, что чем выше скорость робота, тем меньше сканов следует хранить в окне.

Экспериментально получена формула, связывающая размер окна со средней скоростью робота. Под средней скоростью здесь предполагается среднее расстояние в сантиметрах, которое робот проходит между двумя снимками лазерных сканов. Эта формула является эвристической и позволяет связать свойство захвата сканирования (скорость) и свойство фильтра (количество информации, которое должно находиться в окне).

$$Размер\ окна = \frac{27}{v^2}, \tag{4.6}$$

где v – средняя скорость в сантиметрах.

Чтобы определить влияние размера окна, необходимо учесть два параметра, тесно связанных друг с другом и с размером окна. Первый параметр – это порог для коэффициента корреляции Пирсона для каждой пары сканирований. Второй – это общий коэффициент корреляции, который равен произведению коэффициентов. Поскольку коэффициент корреляции Пирсона рассчитывается для двух последовательных сканирований, полученных с небольшой разницей во времени, очевидно, что в среднем они сильно корреллированы. Поэтому порог для пары сканирований должен быть не ниже 0,95, а лучше 0,98. После расчета коэффициента корреляции нового сканирования с каждым сканированием в окне необходимо объединить все коэффициенты. Хорошо известный способ сделать это – их умножить:

$$P_{common} = \prod_{i=1}^{размер\ окна} P_{pair} = P_{pair}^{размер\ окна}, \tag{4.7}$$

где P_{pair} – коэффициент корреляции каждой пары сканов.

Эта формула позволяет оценить общий порог в соответствии с выбранным размером окна и коэффициентом парной корреляции. Например, порог для окна, содержащего 5 сканирований и парную корреляцию 0,98, равен $0{,}98^5 = 0{,}904$.

4.4 Детектор коридоров

Общая идея алгоритма фильтрации основана на том, что похожие сканы следует отбросить. Для каждого скана строится гистограмма, а затем вычисляется корреляция этих гистограмм. Если робот стоит на одном месте, он снимает одни и те же сканы, которые (кроме первого) могут быть отброшены. Если же робот двигается, маловероятна съемка одинаковых сканов. Однако, такая ситуация все же возможна в окружении, не содержащем отличительных особенностей.

Если говорить о помещениях, то таким окружением является коридор. Когда робот движется по коридору вдоль стен, есть только несколько точек,

которые отличают последовательные сканы. Пример такого скана коридора показан на рисунке 4.1. Коридорные сканы являются наиболее сложными для скан матчера, и поэтому их не следует отбрасывать, чтобы не потерять полезную информацию.

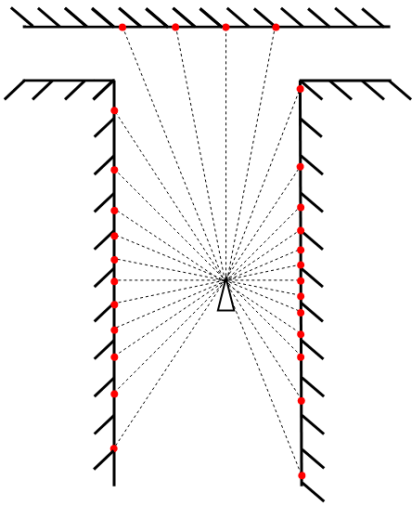


Рисунок 4.1 – Пример коридорного лазерного скана.

Следует ожидать, что корреляция таких сканов очень близка к 1, и они (согласно описанному подходу) неизбежно будут отброшены. Поэтому необходимо обнаруживать коридорные сканы перед фильтрацией и не отбрасывать их. Разработанный детектор основан на предположении, что стены коридоров прямые. Следовательно, значения расстояний точек лазерного скана меняются монотонно. Самый простой способ проверить это – просмотреть каждую точку лазерного сканирования и вычислить знак разницы расстояния текущей точки и следующей. Понятно, что такие действия необходимо проделать в каждой четверти отдельно, как показано на рисунке 4.2.

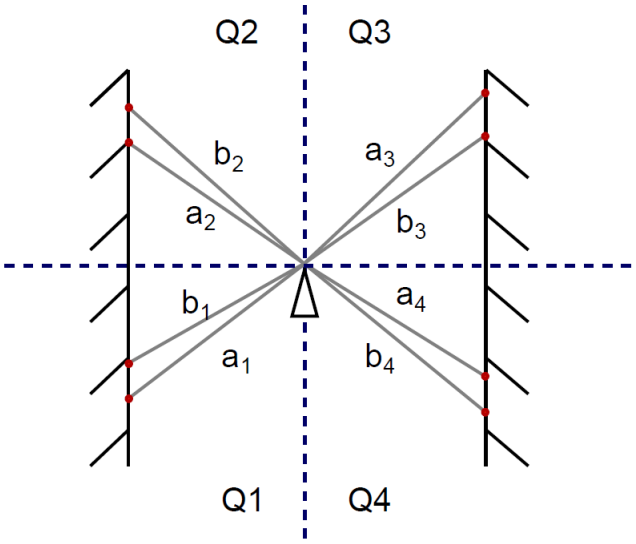


Рисунок 4.2 – Коридорный скан разделяется на четверти, чтобы отдельно рассмотреть знаки разности расстояний для последовательных точек.

Если направление движения робота совпадает с направлением коридора, то разница между a_1 и b_1 должна быть положительной. При этом разница между a_2 и b_2 должна быть отрицательной, несмотря на то, что все эти четыре точки находятся на одной стене. Также разница между a_3 и b_3 должна быть положительной, а разница между a_4 и b_4 – отрицательной. Если робот расположен перпендикулярно коридору, эти знаки меняются на противоположные, но их соотношение остается прежним.

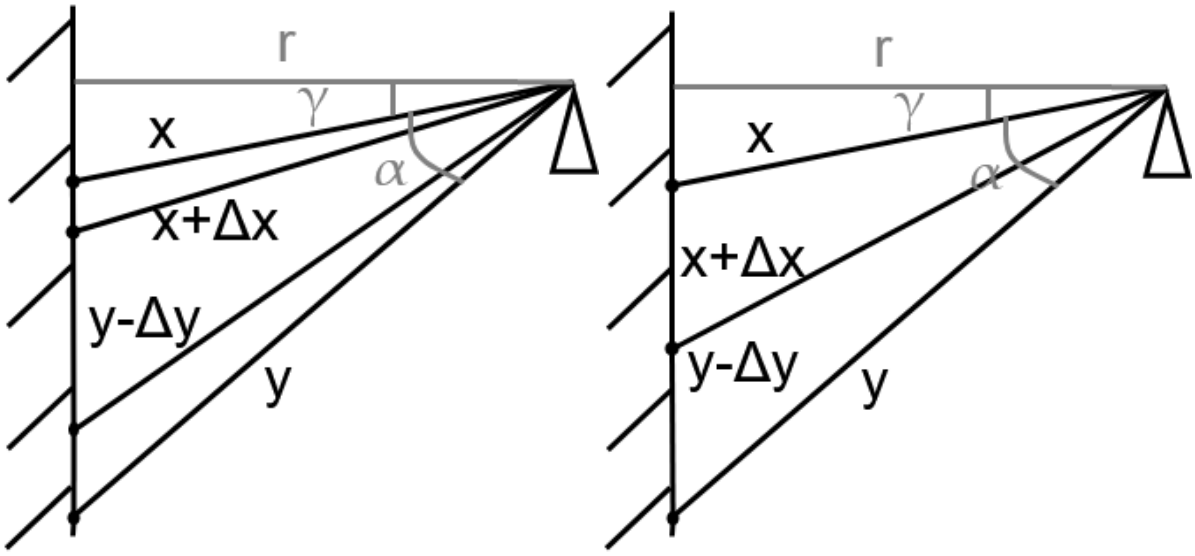
Общий алгоритм детектора коридора следующий:

```
score = 0;
foreach point in laser_scan:
    if point in quarter1 or quarter3:
        score += sign(point - point_next)
    else
        score -= sign(point - point_next)
score = score / sizeof(laser_scan)
```

Если score в итоге превышает заданное заранее пороговое значение, коридор найден. Порог зависит от ограничения понятия «коридор». Если под этим понятием понимать совершенно гладкую пару бесконечных плоских стен, то этот порог может быть близок к 1. Если стена коридора может содержать неровности

или что-то может появиться в середине коридора, то этот порог следует уменьшить. Эксперименты на реальных данных показывают, что порог, равный 0,5, позволяет обнаруживать коридоры без ошибок второго рода. Могут появиться ошибки первого рода, но в наборе данных MIT таких обнаружений в каждой последовательности было менее 0,1%.

Последнее, на что необходимо обратить внимание – это точка point_next в приведенном алгоритме. В самом простом смысле – это точка, следующая за текущей. Но такой выбор может привести к ошибке в вычислениях, если расстояния последовательных точек лазерного скана близки друг к другу и отличаются не более, чем на погрешность лазерного сканера. Возникает задача определения минимального угла между лазерными лучами, падающими на одну стену. Лазерные лучи, которые образуют этот угол, должны иметь длины, не чувствительные к погрешности измерения лазерного дальномера. На рисунке 4.3 представлена геометрическая иллюстрация этой задачи.



а) б)

Рисунок 4.3 – Иллюстрация двух последовательных лучей x и y в одном лазерном скане, где а) угол между лучами x и y достаточно большой, чтобы длины x + Δ x и y – Δ y попадали внутрь него; б) граничная ситуация: угол как раз такой, чтобы максимальные погрешности последовательных лучей совпадали.

Показаны два луча лазера с дальностями x и y. Угол α неизвестен, но его величины должно быть достаточно, чтобы отделить x и y, включая их

погрешность. Погрешность обозначена Δx и Δy. Если α достаточно велико, то угол между $x+\Delta x$ и $y-\Delta y$ положительный, как показано на рисунке 4.3 а). Уменьшение α приводит к уменьшению угла между лучами с погрешностями до граничного случая, который представлен на рисунке 4.3 б), на котором показано, что выполняется равенство $x+\Delta x=y-\Delta y$. На этих рисунках r – расстояние от сканера до коридора; γ - угол между текущим лучом и краем четверти.

Глядя на рисунок 4.3 б) можно получить зависимость α от r, Δ и γ:

$$\cos(\gamma+\alpha)=\frac{r\cos\gamma}{r+2\Delta\cos\gamma}, \tag{4.8}$$

где α – угол между лучами x и y,

r – расстояние от сканера до стены коридора,

γ – угол между текущим лучом x и краем четверти,

Δ – максимальная погрешность лазерного сканера.

Для того чтобы вычислить максимальное значение α достаточно подставить крайнее значение угла γ, равное 0. Тогда можно получить

$$\cos\alpha=r(r+2\Delta)^{-1}. \tag{4.9}$$

4.5 Оценка качества и точности работы фильтра лазерных сканов

Для тестирования фильтр сканов был включен в работу двух алгоритмов SLAM: vinySLAM [27] и Google Cartographer [24]. Фильтр определяет, следует ли обрабатывать скан или отбрасывать его, прежде чем он будет передан скан матчеру каждого из перечисленных алгоритмов. Следовательно, если скан должен быть обработан, то время, требуемое для фильтрации, добавляется к общему времени обработки скана. Поэтому необходимо оценить алгоритмическую сложность процесса фильтрации и затем представить результаты применения такого фильтра к реальным наборам данных MIT [17]. Также нужно оценить долю сканов, которые могут быть отброшены без потери общей точности.

4.5.1 Оценка алгоритмической сложности фильтра

Сначала нужно создать гистограмму, перебирая каждую точку лазерного скана. Это требует O(n) операций, где n – количество точек в скане. Построение угловой гистограммы, описанной в разделе 4.1, требует n умножений, n сложений и 1 деление. Следовательно, процесс создания гистограммы занимает примерно 2n операций, не считая затрат на память. Второй шаг – вычисление корреляции Пирсона – требует 5m сложений и 3m умножений, где m – это количество столбцов в гистограмме, которое обычно более чем в десять раз меньше, чем n. Поскольку корреляция Пирсона должна вычисляться для каждого сканирования в окне, то требуется 8mk операций, где k – размер окна.

Грубая оценка, которая учитывает только самые важные части алгоритма, говорит о том, что для фильтрации одного лазерного скана требуется около 2n + 8mk операций. В реальном эксперименте n составляет около 1000, m – около 30 - 50 и k – 5 - 10. Используя эту оценку, можно сказать, что процесс фильтрации занимает приблизительно столько же времени, что и десяток проходов через все точки лазерного скана. В то же время процесс скан матчинга в vinySLAM, занимает около 100 проходов по всем точкам лазерного скана. Таким образом, приблизительная оценка работы скан матчера составляет 100n операций. Эти оценки являются достаточно приблизительными, но они демонстрируют, что время фильтрации значительно меньше времени сопоставления сканирования – 10n против 100n.

Также необходимо измерить реальную разницу между временем скан матчинга и временем фильтрации. Эксперимент проводился с использованием последовательности mit-2011-01-25-06-29-26 на Ubuntu 18.04, ROS melodic, Intel Core I5-8500 @ 3GHz, 16 Gb RAM. Среднее время обработки сканирования в алгоритме vinySLAM составляет $12.9 \cdot 10^{-3}$ секунд, а среднее время фильтрации для 30 столбцов гистограммы и 5 элементов в окне составляет $5.9 \cdot 10^{-5}$ секунд.

4.5.2 Количественная оценка точности работы алгоритма SLAM с фильтром

Та же аппаратная среда использовалась для расчета точности алгоритмов SLAM с фильтром по наборам данных MIT. Эти наборы данных были выбраны потому, что они содержат достоверную информацию об истинном движении робота во время снятия измерений, позволяющую количественно оценить точность. Эксперимент состоит из нескольких шагов.

1 Запустить обычный алгоритм SLAM для последовательностей этих наборов данных и измерить его СКО (среднеквадратичное отклонение) построенной траектории движения от истинной.

2 Запустить алгоритм SLAM с фильтрацией и оценить его точность так же, как в пункте 1.

3 Оценить долю отброшенных сканов.

Результаты по набору данных MIT представлены в таблице 4.1. Из нее ясно, что точность каждого алгоритма SLAM с фильтром такая же, как точность этого же алгоритма без фильтра. При этом отбрасывается более половины сканов. Важно отметить, что параметры для 7 первых последовательностей отличаются от параметров для 3 последних последовательностей. Они сгруппированы по средней скорости робота.

Средняя скорость робота в первой группе 0,022 м за квант. Квант – это промежуток времени между сканированиями. Для данного лазерного сканера он равен 0,025 сек. Тогда размер окна должен быть равен 5. При этом коэффициент корреляции для пары сканирований равен 0,96, поскольку робот движется быстро. Следовательно, P_{common} равен 0,8. Последний параметр для расчета – количество столбцов (что также является эвристическим) – сильно связан с P_{pair}. Он равняется 30, чтобы гистограмма была чувствительна к возможным изменениям окружающей среды. Для трех последних последовательностей средняя скорость робота близка к 0,017 м за квант. Эта скорость ниже и поэтому размер окна равен 9, $P_{pair} = 0{,}99$, $P_{common} = 0{,}9$, количество столбцов равно 15.

Таблица 4.1 Среднеквадратичное отклонение траектории, построенное алгоритмами SLAM с фильтром и без него от истинной траектории движения робота.

Номер последовательности	**СКО vinySLAM**	**СКО vinySLAM с фильтром**	**СКО cartographer**	**СКО cartographer с фильтром**	**Отброшено, %**
2011-01-20-07-18-45	0.062 ± 0.004	0.078 ± 0.004	0.131 ± 0.058	0.119 ± 0.041	59
2011-01-21-09-01-36	0.080 ± 0.018	0.089 ± 0.013	0.153 ± 0.072	0.163 ± 0.094	56
2011-01-24-06-18-27	0.096 ± 0.007	0.111 ± 0.013	0.183 ± 0.015	0.181 ± 0.014	59
2011-01-25-06-29-26	0.094 ± 0.006	0.100 ± 0.002	0.176 ± 0.010	0.179 ± 0.012	63
2011-01-27-07-49-54	0.170 ± 0.019	0.121 ± 0.006	0.248 ± 0.014	0.251 ± 0.007	52
2011-03-11-06-48-23	0.534 ± 0.085	0.543 ± 0.034	0.586 ± 0.174	0.642 ± 0.191	58
2011-03-18-06-22-35	0.090 ± 0.020	0.090 ± 0.003	0.130 ± 0.025	0.119 ± 0.017	52
2011-04-06-07-04-17	0.183 ± 0.014	0.213 ± 0.027	0.188 ± 0.011	0.185 ± 0.011	51
2011-01-19-07-49-38	0.305 ± 0.174	0.289 ± 0.181	0.188 ± 0.004	0.189 ± 0.005	50
2011-01-28-06-37-23	0.361 ± 0.175	0.348 ± 0.152	0.378 ± 0.025	0.399 ± 0.030	47

4.6 Выводы по четвертой главе

В этой главе представлен алгоритм фильтрации массивов данных больших размеров, основанный на корреляции этих массивов друг с другом. Если новый массив похож на несколько предыдущих, он может быть отброшен без потери точности. Эксперименты проводились с наборами данных MIT, представляющих собой лазерные сканы, для vinySLAM и Cartographer. Они показывают, что можно отбросить больше половины сканов, а иногда достаточно оставить только одну треть от общего количества сканов в последовательности данных.

Описан алгоритм детектора коридоров. Его точность зависит от концепции коридора и может отличаться в конкретных обстоятельствах. В наборе данных MIT коридор между офисными помещениями обнаруживается с помощью этого алгоритма в каждой последовательности без ложноотрицательных ошибок.

Эксперименты показывают, что предложенный алгоритм фильтрации работает несравнимо быстрее, чем алгоритм скан матчинга ($5.9 \cdot 10^{-5}$ секунд – для

фильтрации, $12.9 \cdot 10^{-3}$ – для скан матчера vinySLAM). Следовательно, процесс фильтрации в целом экономит вычислительные ресурсы. Сэкономленное время вместо скан матчинга может быть использовано для других возможных нужд робота. Применение этого фильтра сокращает среднее время обработки скана более чем на 40%.

Высокая производительность достигается за счет уменьшения количества данных в лазерном скане путем создания гистограммы. Необходимо создать несколько типов гистограмм, например, количество точек на определенном расстоянии от сканера или средний диапазон для определенного угла обзора. Затем гистограммы последовательных сканов сравниваются друг с другом, и вычисляется коэффициент корреляции Пирсона. Если корреляция высока, скан необходимо отбросить.

5 Качественные и количественные показатели могоагентного алгоритма

В этой главе описана программная реализация алгоритма, описанного в главе 3. Предложена методика тестирования качественных и количественных показателей разработанного алгоритма многоагентного SLAM. Для оценки границ применимости разработанного программного обеспечения (ПО) на роботах, оснащенных низкими вычислительными мощностями, произведено измерение времени и точности на некоторых аппаратных платах (Raspberry Pi).

5.1 Программная реализация разработанного алгоритма

Программное обеспечение, реализующее предложенный алгоритм входит в состав открытого фреймворка, имеющего название slam-constructor [26]. Назначение этого фреймворка – предоставить пользователю гибкую систему настройки SLAM алгоритма, как в конструкторе. Разделение логики на классы и интерфейсы выполнено таким образом, чтобы предоставить пользователю возможность как использовать некоторые уже собранные алгоритмы, так и заменять в них любые компоненты. К таким компонентам относятся: строение карты занятости, модель ячейки в карте занятости, скан матчер, фильтр частиц и др.

Реализация программного обеспечения, описывающего рассмотренный в главе 3 многоагентный алгоритм, была выполнена на базе этого фреймворка. Кроме того, необходимо добавить, что фреймворк slam-constructon разработан с использованием другого фреймворка, который носит название Robot Operating System (ROS) [44]. ROS выполняет роль системы передачи сообщений DDS, о которой подробнее сказано в пункте 3.1.2.

Обобщенная схема фреймворка представлена на рисунке 5.1.

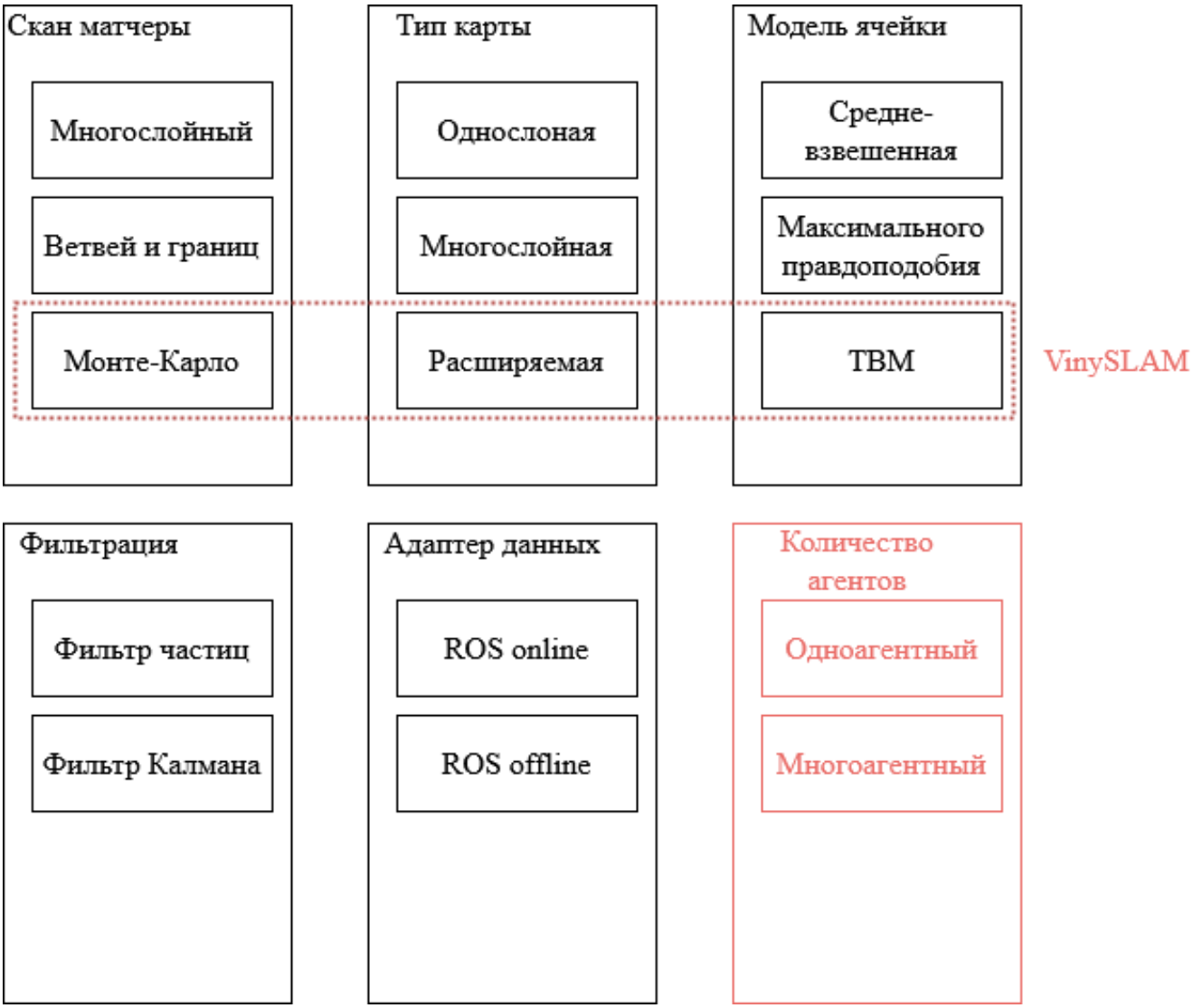


Рисунок 5.1 – Схема компонентов в фреймворке slam-constructor.

Схема реализованного многоагентного алгоритма решения задачи SLAM представлена на рисунке 5.2.

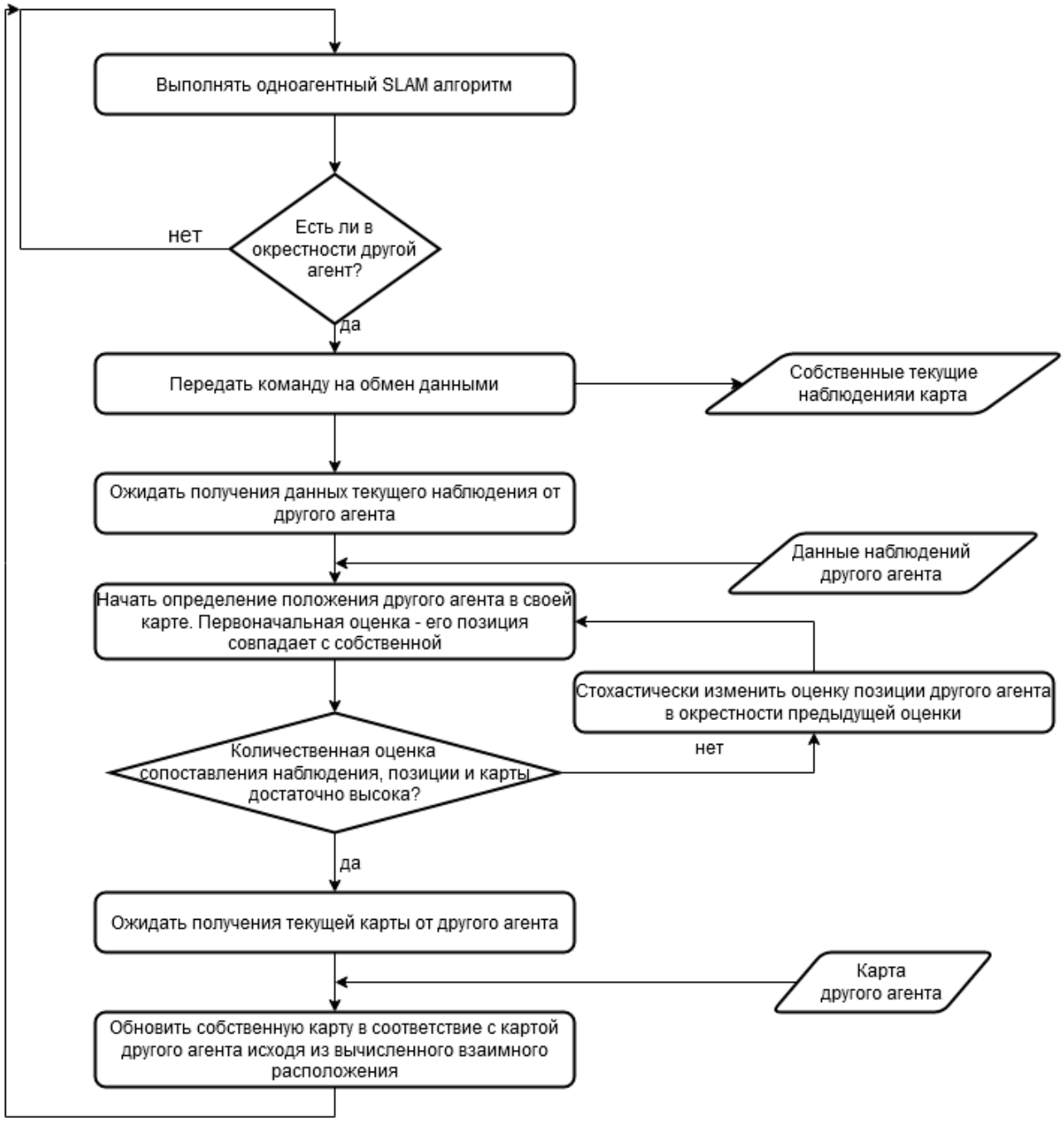


Рисунок 5.2 – Схема разработанного алгоритма.

5.2 Архитектура ПО общего назначения

Архитектура разработанного программного обеспечения продиктована многоагентным алгоритмом SLAM, описанным в разделе 3.3. Упрощенная диаграмма базовых классов, используемых в разработанном программном обеспечении, представлена на рисунке 5.3.

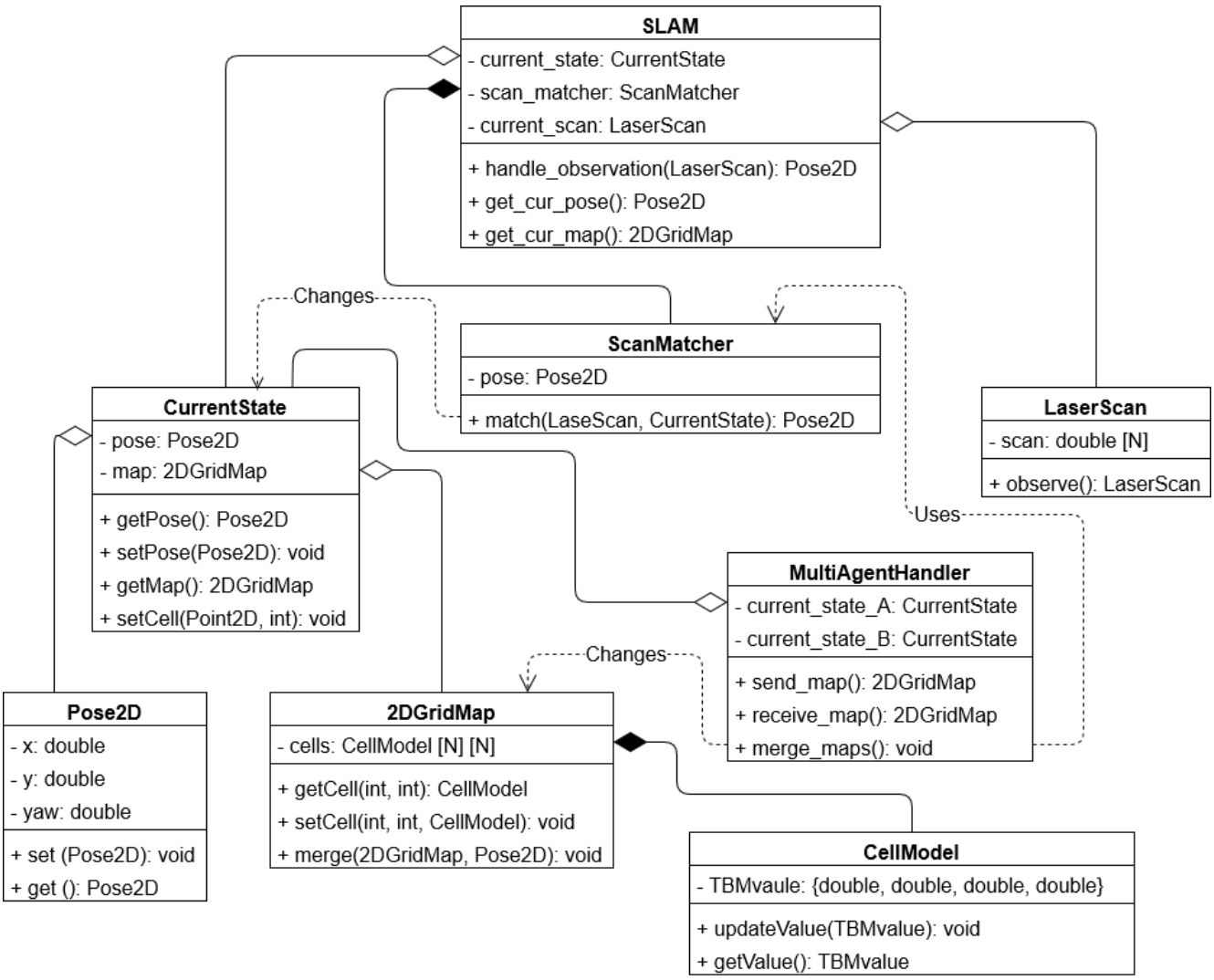


Рисунок 5.3 – UML диаграмма базовых классов, входящих в разработанное ПО

Основным интерфейсом, ответственным за начало работы, является класс SLAM и его наследники. Благодаря структуре фреймворка slam-constructor, на его базе можно создавать множество SLAMалгоритмов, отличающихся деталями реализации. В рамках представленной работы были использованы уже реализованные классы фреймворка slam-constructor, но существенно дополненные в функциональности.

Как видно на рисунке 5.3, класс MultiAgentHandler затрагивает все основные классы, задействованные в работе одноагентного алгоритма SLAM: и скан-матчер, и класс, содержащий информацию о текущей позиции и карте. Ответственность за наследников этого класса несет метод определения близости агентов, прежде чем они начнут обмен информацией. Как только определено, что агенты, выполняющие многоагентный алгоритм SLAM, находятся в окрестности

друг друга, то согласно методу, описанному в главе 3, начинается процесс вычисления взаимного расположения агентов. Этим занимается наследник класса ScanMatcher. Глобальная задача этого класса в одноагентном алгоритме вычислить новую позицию робота, когда известна карта, построенная во время нахождения робота в предыдущей известной позиции, а также – новый лазерный скан. Этот же класс с его полной функциональностью можно использовать для вычисления взаимного расположения агентов с использованием последней известной позиции и карты одного агента, а также наблюдением другого агента.

Класс CellModel отвечает за такие параметры ячейки, как метод вычисления занятости ячейки, метод соответствия точки лазерного скана и ячейки карты, метод объединения ячеек во время слияния карт. На практике все эти задачи решают различные наследники класса CellModel. Новизна работы заключается в применении теории Демпстера-Шафера к методу хранения информации о занятости ячейки. На рисунке 5.3 это продемонстрировано членом-данным TBMvalue, состоящим из четырех чисел с плавающей запятой. TBM – Transferrable Belief Model – дополнение к теории Демпстера-Шафера, которое предполагает, что масса конфликта (о массах в теории Демпстера-Шафера подробнее говорилось в разделе 3.2.2) всегда должна быть равна нулю. TBM предлагает способ устранения конфликта путем пропорционального разделения массы конфликта на остальные массы.

Поток данных, которым обмениваются основные узлы в разработанном программном обеспечении, можно увидеть на диаграмме последовательности, представленной на рисунке 5.4. Для упрощения на этой диаграмме показаны только два основных узла – скан матчер (реализованный в классе ScanMacher) и детектор (MultiAgentHandler). Остальные узлы опущены, поскольку взаимодействие с ними можно считать атомарными операциями, не требующими дополнительных действий. Также можно считать, что скан матчер владеет всеми сведениями о текущей карте, положении робота и текущем наблюдении.

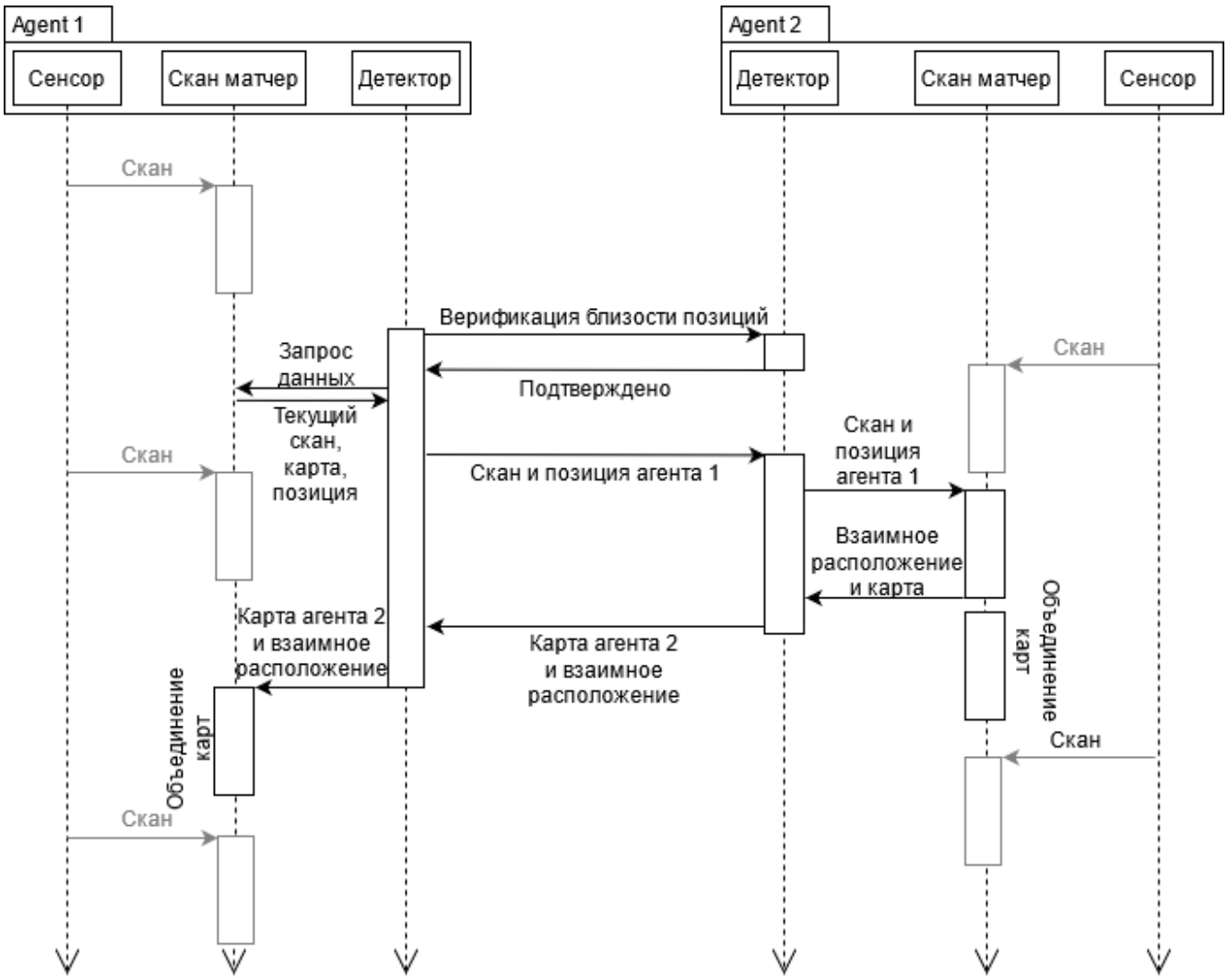


Рисунок 5.4 – UML диаграмма последовательности в разработанном ПО

Из рисунка 5.4 видно, что взаимное расположение вычисляет только один агент. Это поведение не определено методом, описанным в главе 3, в данной реализации оно служит для ускорения работы программы. Агент, не выполняющий вычисление взаимного расположения, может в это время продолжать выполнять SLAMалгоритм.

5.3 Методика тестирования точности разработанного алгоритма

Наиболее весомым способом определения точности любого SLAM алгоритма является запуск алгоритма на настоящем роботе, оснащенном физическими датчиками. Другим распространенным методом является программирование симуляции, где данные, получаемые роботом, синтетически генерируются и сопровождаются белым шумом. Использование настоящих

роботов выходило за рамки данной работы, а использование симуляции неизбежно влияло бы на точность работы алгоритма. Поэтому в данной работе для оценки точности алгоритма, используются наборы данных, полученные с физических роботов и опубликованные в открытом доступе.

Среди открытых наборов данных, на которых обычно тестируются лазерные алгоритмы SLAM, можно выделить наборы данных, предоставляемые лабораторией университета MIT [17], набор данных Willow Garage [38]. Они оба использовались при оценке качества алгоритмов, входящих в состав slam-constructor. В набор данных входит последовательность наблюдений лазерного дальномера и оценка одометрии в каждый момент времени.

5.3.1 Выбор характеристики измерения точности алгоритма

Для того, чтобы оценить точность работы SLAM алгоритма, необходимо количественно сравнить артефакты, полученные в ходе его работы: карту и траекторию движения робота с истинными картой и траекторией. Количественное сравнение карт – это задача анализа изображений. Характеристики такого сравнения достаточно многогранны и требуют специфических знаний об истинной карте: масштаб, допустимая погрешность при построении изображения и другие. Также необходимо вычислять, какую часть истинной карты робот успел пронаблюдать.

Поэтому для оценки качества работы SLAM алгоритма используется сравнение траектории как последовательности позиций, полученной в ходе выполнения алгоритма SLAM с истинной траекторией. К рассматриваемым наборам данных не прилагается истинная траектория. Однако набор данных университета MIT поставляется вместе с таблицей, которую можно использовать в качестве входных данных для решения задачи локализации (другими словами, вместе с утилитой, которая позволяет вычислить траекторию движения робота в записанном наборе данных, исходя из решения задачи локализации). Набор данных Willow Garage не предоставляет такой возможности, поэтому он не

использовался для определения точности работы многоагентного SLAM алгоритма.

Поскольку известна истинная траектория движения робота для каждого набора данных MIT, а именно позиция робота и временная метка, соответствующие этой позиции, то можно вычислить отклонение в каждый момент времени. Исходя из такого массива отклонений, можно вычислить среднеквадратичное отклонение как характеристику качества работы SLAM алгоритма. Недостатком предложенного подхода является вероятность ошибки вычисления истинной траектории при решении задачи локализации. Однако это самый надежный способ определения количественных показателей точности алгоритма.

Недостаточно просто вычислить среднеквадратичное отклонение от истинной траектории, необходимо также определить, насколько оно мало. Поэтому необходим надежный алгоритм решения задачи SLAM и сравнение среднеквадратичных отклонений различных алгоритмов. В качестве оценивающего алгоритма часто используется gmapping [15] или cartographer [24]. В данной работе сравнение будет производиться с алгоритмом cartographer, а также с алгоритмом vinySlam, входящим в состав slam-constructor.

5.3.2 Применение набора данных MIT для тестирования многоагентного алгоритма SLAM

Рассмотренные наборы данных могут быть применены для запуска одноагентных алгоритмов SLAM. В рамках этой работы данные в наборах необходимо адаптировать для многоагентного алгоритма. За исключением технических проблем с переименованием меток в наборе данных необходимо определить момент, когда роботы оказываются в окрестности друг друга. Это является проблемой, поскольку при записи наборов данных робот был один. Поэтому, даже при запуске одновременно двух одноагентных алгоритмов на разных последовательностях данных внутри одного датасета, определять, что роботы находятся близко друг к другу, необходимо в ручном режиме.

Таким образом, первый эксперимент, позволяющий вычислить точность многоагентного алгоритма, состоит из следующих шагов.

1 Запустить алгоритм на одной из последовательностей данных из набора данных MIT, эмулирующей одного робота из стаи.

2 Запустить алгоритм на другой последовательности данных из этого же набора данных MIT. Необходимо заранее проследить, чтобы эти последовательности были записаны с учетом существования момента времени, когда роботы из обеих последовательностей находятся вблизи друг от друга.

3 В момент, когда роботы оказываются вблизи друг от друга мануально послать команду на обмен текущими наблюдениями и картами.

4 После обмена данными и обновления карт агенты продолжают выполнение алгоритма до конца записанной последовательности данных.

5 После завершения воспроизведения набора данных каждый агент сравнивает построенную им траекторию с траекторией движения, построенной одноагентным алгоритмом на той же последовательности данных

Цель эксперимента № 1 показать, что использование карты другого робота не уменьшает точность работы по сравнению с одноагентным алгоритмом, а также позволяет дополнить карту агента участками, которые он до сих пор не посещал. Это позволяет ускорить работу за счет сопоставления лазерного скана и уже построенной карты – алгоритмически менее затратной операцией, чем встраивание скана в неизвестную карту.

Пример карт, полученных в ходе выполнения эксперимента 1, показан на рисунке 5.5.

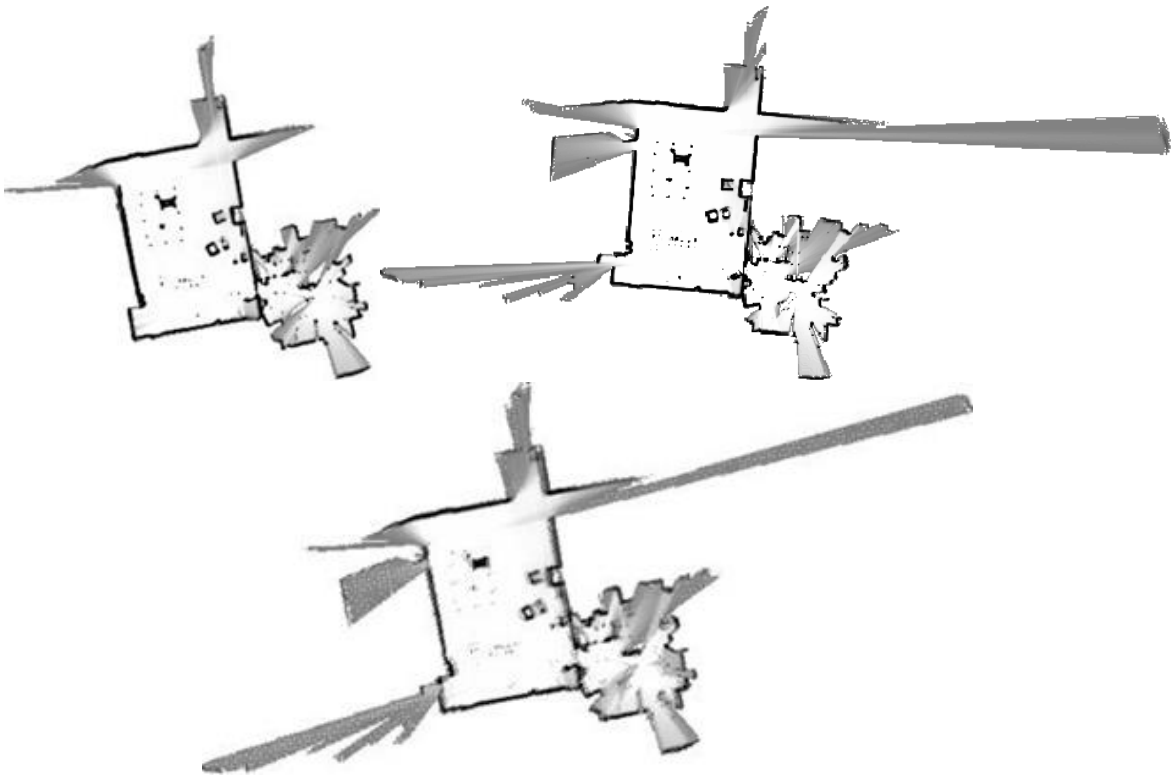


Рисунок 5.5 – Объединение карт, построенных в ходе решения задачи SLAM, на разных последовательностях данных MIT

Второй эксперимент заключается в запуске одной и той же последовательности одновременно два раза. Одна копия воспроизводится без изменений, а вторая – задом наперед. Такой случай гарантирует, что в середине последовательности найдется точка, в которой одновременно окажутся оба агента. Последовательность шагов в эксперименте 2 выглядит следующим образом.

1 Запустить алгоритм на одной из последовательностей данных из MIT.

2 Запустить алгоритм на этой же последовательности, воспроизведенной в другую сторону.

3 Когда роботы окажутся в одной точке, послать им сигнал обменяться текущими наблюдениями и картами.

4 После обмена данными и обновления карт агенты продолжают выполнение алгоритма до конца записанной последовательности данных.

5 После завершения воспроизведения набора данных каждый агент сравнивает построенную им траекторию с траекторией движения, построенной одноагентным алгоритмом на этой же последовательности данных.

Ключевая особенность такого эксперимента в демонстрации, насколько использование именно того участка карты, по которому робот будет двигаться в будущем, влияет на точность результатов. Примеры карт, строящихся в ходе исполнения эксперимента № 2, показаны на рисунке 5.6.

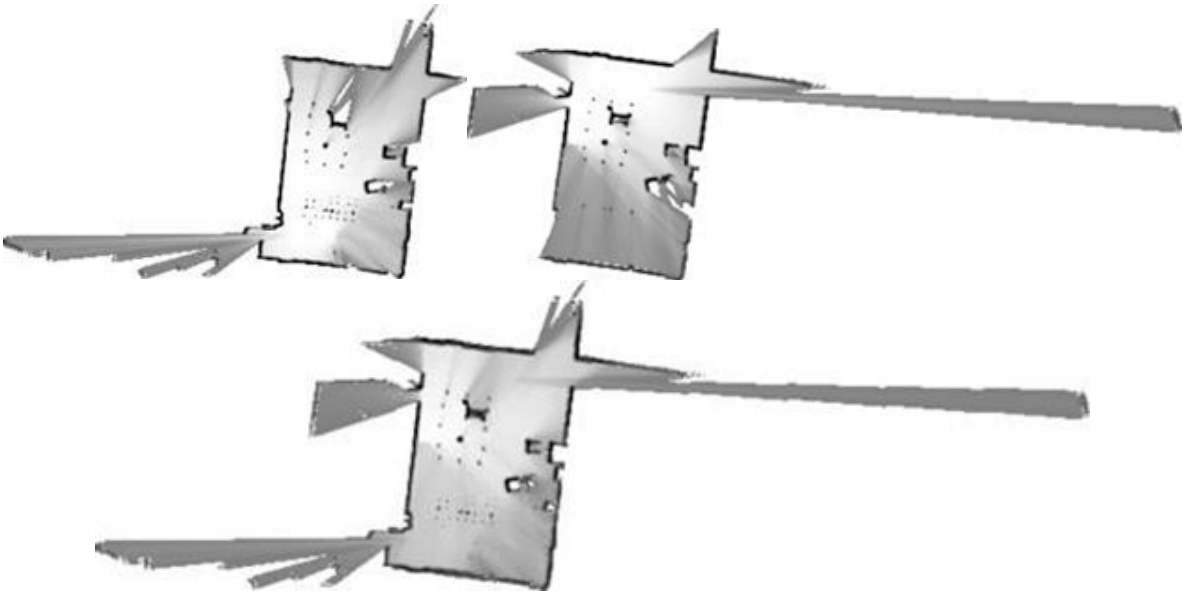


Рисунок 5.6 – Карты, построенные различными агентами на одной и той же последовательности, воспроизведенной в прямую и обратную сторону, а также результат их объединения

В ходе выполнение каждого из описанных экспериментов вычисляются траектории движения каждого робота. Для определения их истинности необходимо вычислить среднеквадратичное отклонение каждой точки траектории от точек, в которых робот был в действительности в момент измерения. Другими словами, каждая последовательность данных сопровождается набором координат, через которые проходил робот, и временной меткой, когда каждая координата была посещена. Построенная в результате работы SLAM алгоритма траектория также состоит из координат и соответствующих им временных меток.

Для вычисления среднеквадратичного отклонения траекторий необходимо воспользоваться формулой:

$$СКО = \sqrt{\sum_{i=1}^{n} \left(x_i - x_i^* \right)^2}, \tag{5.1}$$

где x_i – вычисленная позиция робота в момент времени i,

x_i^* – истинная позиция робота в момент времени i,

n – количество измеренных позиций робота.

Поскольку некоторые компоненты многоагентного алгоритма SLAM являются стохастическими, то такие измерения СКО траектории необходимо вычислить несколько раз и найти среднее СКО по формуле

$$\overline{СКО} = \frac{1}{n}\sum_{i=1}^{n} СКО_i \,, \tag{5.2}$$

где n – количество независимых итераций.

К тому же необходимо вычислить дисперсию СКО по формуле

$$D_{СКО} = \sqrt{\sum_{i=1}^{n}\left(СКО_i - \overline{СКО}_i\right)^2} \tag{5.3}$$

5.4 Оценка точности разработанного многоагентного SLAM алгоритма

Оценка производилась на настольном компьютере со следующими характеристиками: процессор: Intel Core i7-860 4x2.8GHz, оперативная память DDR3 8GB, операционная система Ubuntu Xenial x64.

Далее приведены таблицы, демонстрирующие среднее среднеквадратичное отклонение и его дисперсию согласно экспериментам 1 и 2, описанным в разделе 5.3. В таблице 5.1 указано среднеквадратичное отклонение, полученное агентами на последовательностях данных MIT по сравнению с одноагентными алгоритмами vinySLAM и catographer. Таблица 5.2 показывает среднее СКО и его дисперсию для эксперимента 2.

Таблица 5.1 Среднеквадратичное отклонение для эксперимента 1 в сравнении с одноагентным алгоритмом из ядра, а также алгоритмом google cartographer

Последовательность данных	Длина пути, м	СКО, м	СКО алгоритма из ядра, м	СКО алгоритма cartographer, m
2011-01-20-07-18-45	76	0.045 ± 0.005	0.062 ± 0.004	0.131 ± 0.058
2011-01-21-09-01-36	87	0.080 ± 0.018	0.080 ± 0.018	0.153 ± 0.072
2011-01-24-06-18-27	87	0.096 ± 0.011	0.097 ± 0.007	0.183 ± 0.015
2011-01-25-06-29-26	109	0.094 ± 0.009	0.094 ± 0.006	0.176 ± 0.010
2011-01-28-06-37-23	145	0.395 ± 0.190	0.361± 0.175	0.201 ± 0.011
2011-01-27-07-49-54	94	0.167 ± 0.018	0.170 ± 0.019	0.248 ± 0.014

Таблица 5.2 Среднеквадратичное отклонение для эксперимента 2

Последовательность данных	Длина пути, м	СКО в «прямой» последовательности, м	СКО в «обратной» последовательности, м
2011-01-20-07-18-45	38 + 24	0.044 ± 0.015	0.021 ± 0.005
2011-01-21-09-01-36	43 + 31	0.080 ± 0.011	0.068 ± 0.012
2011-01-24-06-18-27	43 + 41	0.106 ± 0.009	0.091 ± 0.003
2011-01-25-06-29-26	33 + 61	0.033 ± 0.013	0.097 ± 0.016
2011-01-28-06-37-23 часть 1	41 + 39	0.184 ± 0.093	0.231 ± 0.040
2011-01-28-06-37-23 часть 2	41 + 39	0.311 ± 0.187	0.272 ± 0.195
2011-01-27-07-49-54	31 + 62	0.142 ± 0.019	0.161 ± 0.022

В графическом виде данные, собранные в таблице 5.1, представлены на рисунке 5.7.

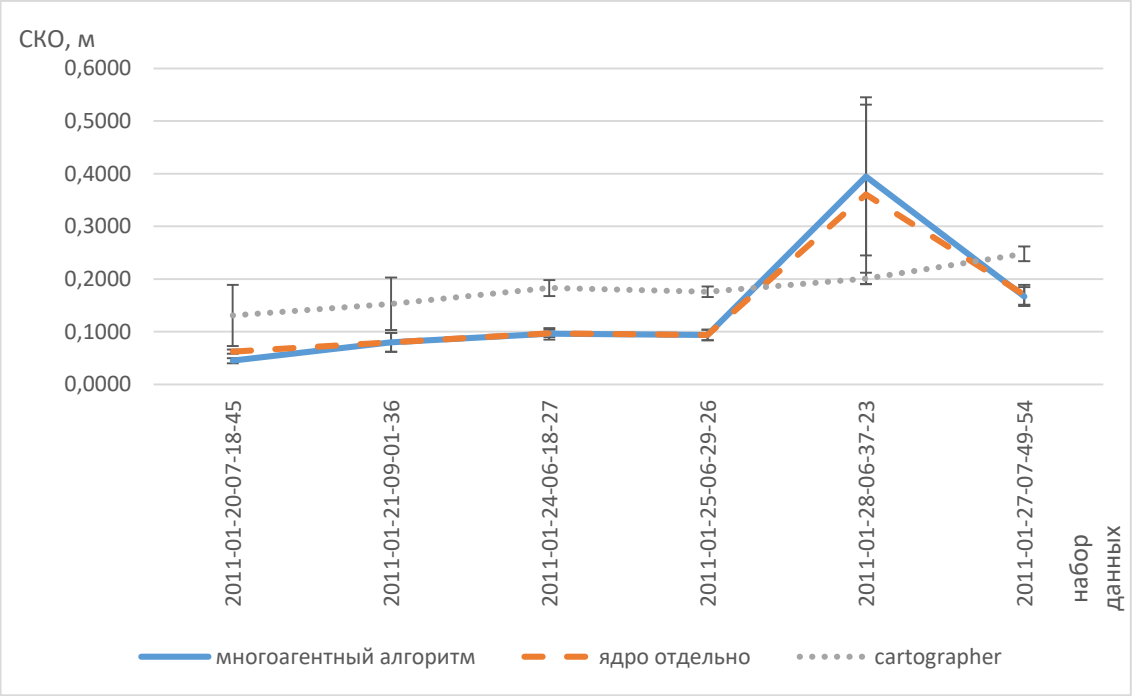


Рисунок 5.7 – Визуализация СКО многоагентного алгоритма, алгоритма ядра и алгоритма cartographer

Точность многоагентного алгоритма соответствует точности одноагентного алгоритма, находящегося в ядре. Это связано с тем, что количество раз, когда агенты обмениваются данными существенно меньше, чем количество итераций,

проделанных независимо. При таком подходе важную роль играет детерминированность алгоритма в ядре. Если алгоритм в ядре является детерминированным, то использование карты других агентов не увеличивает и не уменьшает точность выходных данных. В рассмотренном эксперименте алгоритм vinySLAM не детерминированный, поскольку в нем присутствует стохастический скан матчер. В этом случае любое подмножество агентов в системе может обладать недостоверной картой. Например, каждый агент мог оценить собственную позицию неверно в одном и том же участке пространства. Тогда все агенты, допустившие эту ошибку, будут обладать одинаковыми или близкими ошибочными картами. Использование таких карт агентом, который не посещал этого места, приведет к еще одной неверно построенной карте. Избежать этого невозможно, не внося изменения в ядро алгоритма. Однако стохастическая природа ядра позволяет рассматривать случай одновременной ошибки всех агентов как промах. Следовательно, в среднем даже ошибочные карты некоторых агентов могут быть исправлены другими агентами, которые построили карту правильно. Эта ситуация продемонстрирована в эксперименте на последовательности 2011-01-20-07-18-45. Здесь многоагентный алгоритм показал более высокую точность, чем одноагентный. Это связано с тем, что одноагентный алгоритм имеет самую большую погрешность во время поворота робота. Точка встречи агентов была выбрана как раз на месте, где агент выполняет поворот; поэтому каждый агент передает друг другу свою часть карты, а точное вычисление взаимного положения позволяет избежать ошибки при повороте.

Результаты доказывают точность предложенного алгоритма, поскольку среднеквадратичная ошибка всегда не превышает 0,5 м, а также почти всегда ниже, чем у графового алгоритма google cartographer. Этот результат в основном основан на точности одноагентной версии алгоритма vinySlam. Наибольшая погрешность может возникнуть, если при слиянии карт результатом вычисления взаимного расположения агентов стала ошибка во вращении. Несмотря на то, что в этом случае выходная карта будет согласованной, выходная траектория может заметно отличаться. Справедливо заметить, что эта проблема возникает в любом

мультиагентном алгоритме; и даже графовый алгоритм может исправить ее только в определенных условиях, когда агенты посещают наиболее противоречивую часть окружающего мира несколько раз.

Качественная оценка точности предложенного алгоритма представлена на рисунках 5.8 – 5.10. Рисунок 5.8 – карта, построенная одним агентом. Рисунок 5.9 – фрагмент карты, который был независимо создан другим агентом. Агенты встречаются друг с другом в первый раз и начинают процесс обмена данными. Результат объединения карт представлен на рисунке 5.10. Для карт таких больших размеров качество результирующей карты сильно зависит от работы скан матчера, сопоставляющего чужие наблюдения. Видно, что общая часть на обеих картах не содержит артефактов. Это происходит потому что правило соединения ТВМ позволяет сгладить части карт, которые содержат противоречивую информацию.

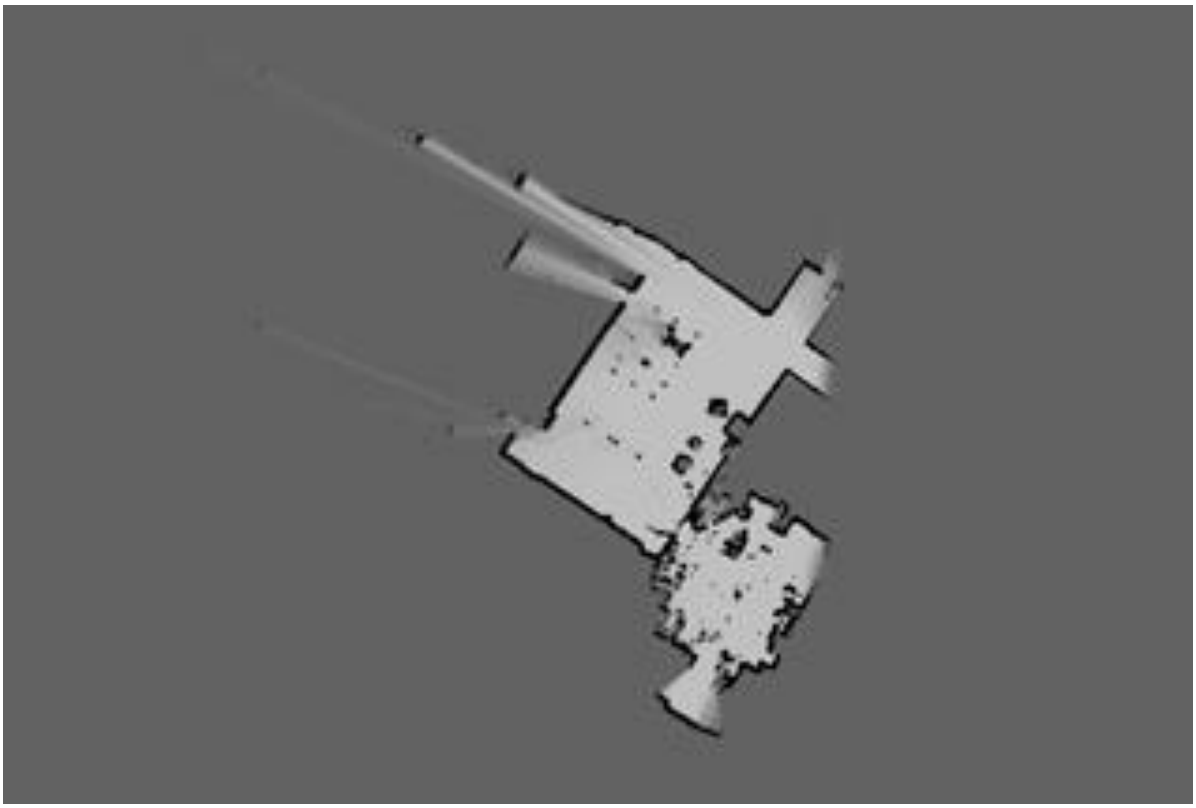


Рисунок 5.8 – Карта, независимо построенная одним агентом

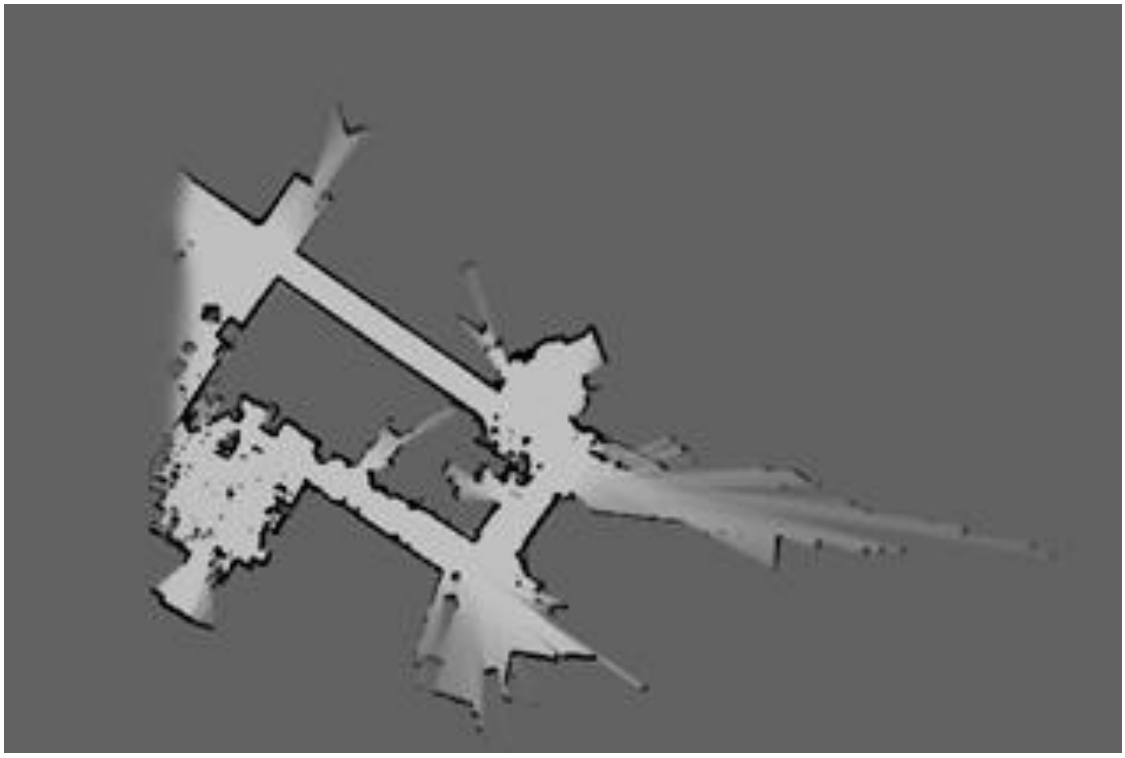


Рисунок 5.9 – Карта, независимо построенная другим агентом

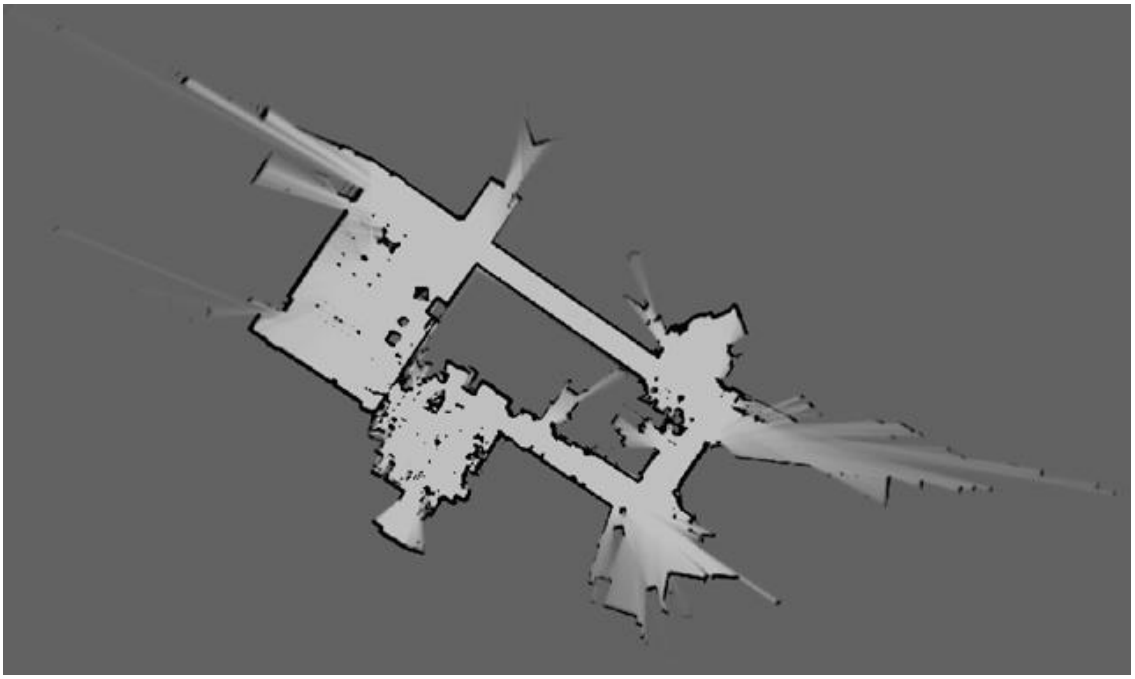


Рисунок 5.10 - Результат объединения независимо построенных карт

5.5 Оценка производительности и потребляемых ресурсов

Помимо оценки точности необходимо определить скорость работы многоагентного алгоритма на различных аппаратных конфигурациях. Цель этого эксперимента определить границы применимости на низко производительных

вычислительных устройствах. Измерения скорости работы необходимо разделить на два этапа:

1) определение частоты обработки измерений во время выполнения одноагентной части алгоритма;

2) определение скорости обмена данными при встрече, а также скорости определения взаимного расположения и обновления карты.

Измерение скорости работы производилось на следующих конфигурациях.

1 Raspberry Pi 3 Model B (Процессор Broadcom BCM2837, оперативная память LPDDR2 1GB, операционная система Ubuntu Xenial x64).

2 Raspberry Pi 3 Model B+ (Процессор Quad Core 1.2GHz Broadcom BCM2837B0, оперативная память LPDDR2 1GB, операционная система Ubuntu Xenial x64).

3 Raspberry Pi 4 Model B (Процессор Quad Core 1.5GHz Broadcom BCM2711, оперативная память LPDDR4 2GB, операционная система Ubuntu Xenial x64).

4 Персональный компьютер (Процессор: Intel Core i7-860 4x2.8GHz, оперативная память DDR3 8GB, операционная система Ubuntu Xenial x64).

Оценка производилась на платах Raspberry Pi [47], поскольку они являются недорогими и весьма популярными вычислительными устройствами, используемыми в робототехнике. Эксперимент также проводился на виртуальных машинах, обладающих ресурсами, сопоставимыми с ресурсами Raspberry Pi. Результаты оказались слишком похожими, поэтому они не представлены на отдельном графике. Поэтому можно считать, что для получения рассматриваемой скорости работы вместо продуктов Raspberry можно использовать другие платы с аналогичными ресурсами.

На рисунке 5.11 показано количество сканов в секунду, которые могут быть обработаны на перечисленных конфигурациях. За условную границу принята частота обработки входных данных 30 сканов в секунду, что сопоставимо с частотой обработки человеческого глаза. Поэтому условно можно считать, что

если алгоритм выполняется с частотой больше, чем 30 сканов в секунду, то он работает в режиме реального времени.

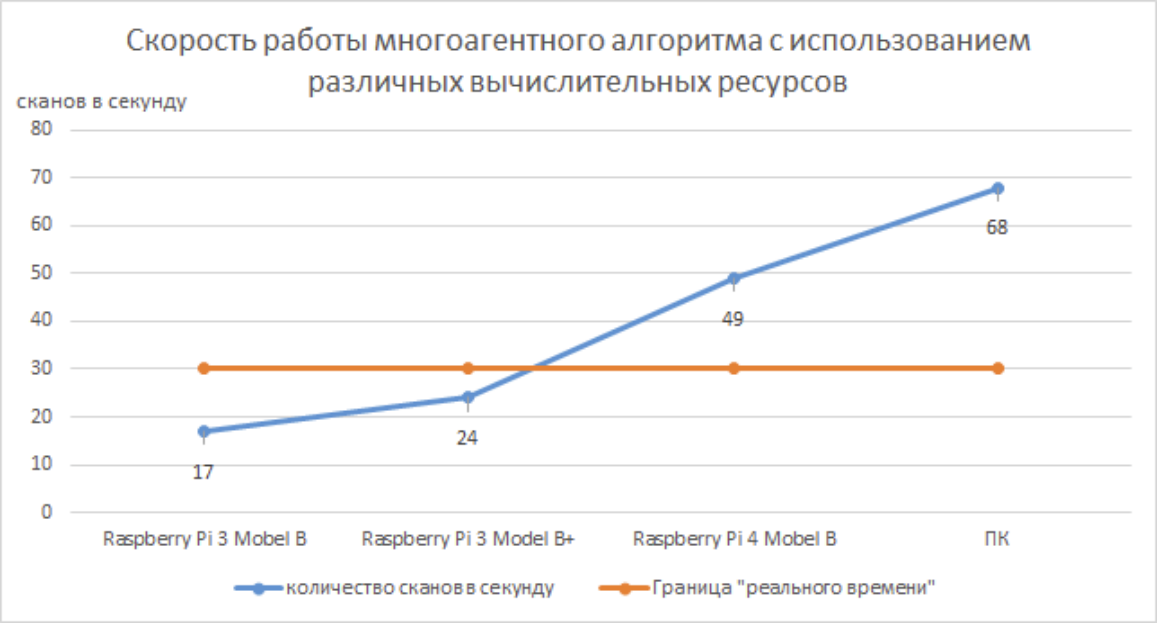


Рисунок 5.11 – График, демонстрирующий количество обрабатываемых сканов в секунду на различных вычислительных конфигурациях.

На рисунке 5.12 представлено сравнение скорости работы предложенного алгоритма и алгоритма cartographer, с которым выполнялось сравнение точности в разделе 5.4.

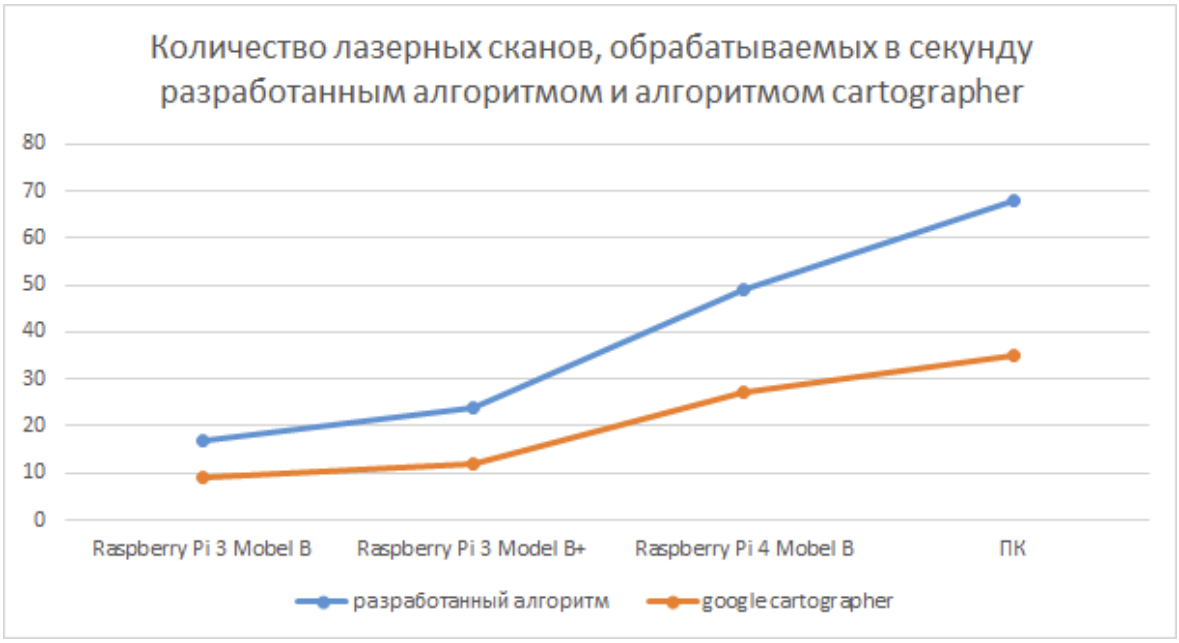


Рисунок 5.12 – График, демонстрирующий сравнение количества обрабатываемых сканов в секунду на различных вычислительных конфигурациях для разработанного алгоритма и алгоритма Google cartographer.

Очевидно, что процесс обмена данными занимает больше времени, чем обработка одного наблюдения. Этот процесс состоит из трех этапов.

1 Передача данных наблюдений (впоследствии и карты).

2 Обработка наблюдений и скан матчинг.

3 Обновление карты в соответствии с чужой картой.

Обработка наблюдений и скан матчинг являются частью одноагентного алгоритма и должны выполняться так же быстро. Однако скан матчер обладает заранее заданными параметрами поиска. В одноагентном алгоритме он настроен на поиск в локальной области, а в многоагентном – область гораздо шире. Поэтому время, которое требуется на обмен данными и слияние карт значительно больше, чем время на обработку одного наблюдения.

График, показанный на рисунке 5.13, демонстрирует время, которое потребовалось тестируемым вычислительным конфигурациям, чтобы провести обмен картами и наблюдениями. Все агенты, выполняющие многоагентный алгоритм, запускались на одной вычислительной конфигурации, поэтому особенности сетевого взаимодействия не рассматривались в эксперименте.

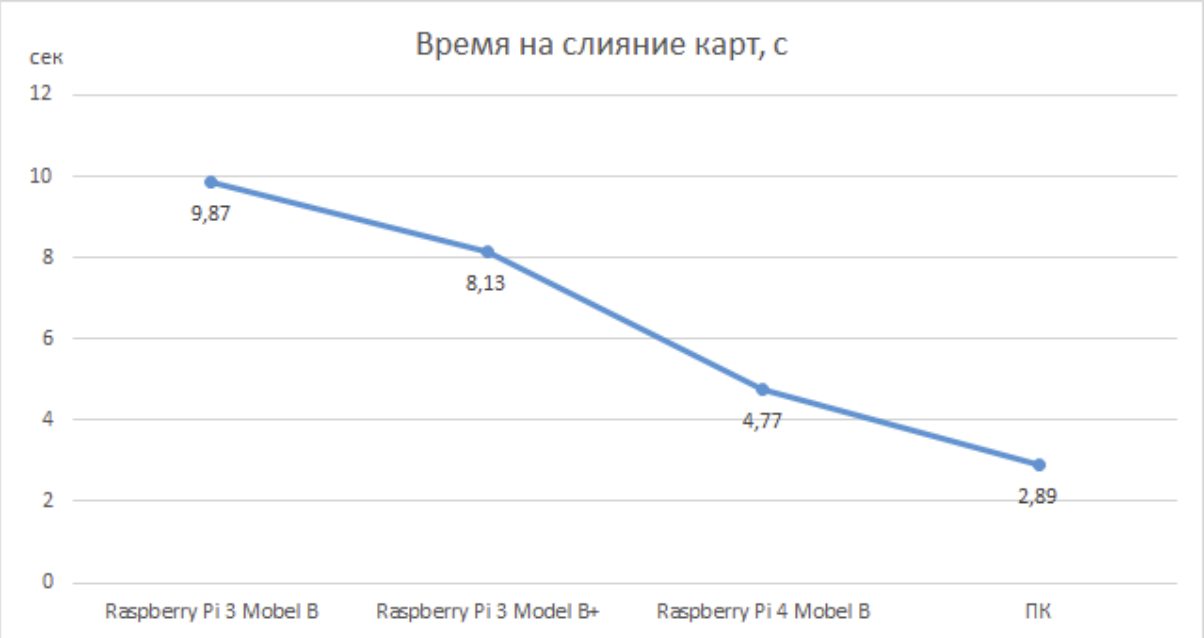


Рисунок 5.13 – График, демонстрирующий время, затраченное на вычисление взаимного расположения и слияние карт на различных вычислительных конфигурациях.

5.6 Выводы по пятой главе

В этой главе рассмотрено тестирование характеристик точности и производительности разработанного программного обеспечения на основе многоагентного алгоритма, представленного в главе 3. Программная реализация основана на фреймворках ROS и slam-constructor. По результатам тестирования среднеквадратичная ошибка многоагентного алгоритма на дистанции 100 метров, пройденной одним агентом (при условии однократной синхронизации с другим агентом) составила 9,4 см. Исследование производительности показало, что разработанный алгоритм может обрабатывать до 24 лазерных сканов в секунду на платах модели Raspbery Pi 3 Model B+, обладающих ограниченными вычислительными ресурсами; а время на синхронизацию карт значительно превышает время обработки одного лазерного скана. Следовательно, во время слияния карт роботам необходимо выделить все ресурсы на синхронизацию и приостановить наблюдение за окружающей средой, что можно отнести к недостатку разработанного алгоритма.

Заключение

Итогом проведенных теоретических и экспериментальных исследований является решение актуальной задачи одновременной локализации и построения карты стаей мобильных роботов. **Поставленные задачи решены**, и можно сделать следующие выводы:

1 После проведения сравнительного анализа и классификации существующих одноагентных и многоагентных алгоритмов, решающих задачу SLAM, была построена классификация одноагентных алгоритмов по различным характеристикам.

2 Разработан масштабируемый алгоритм многоагентного решения задачи SLAM. Он имеет низкую вычислительную сложность за счет применения метода Монте-Карло к задаче скан-матчинга, а также отказа от графовой структуры алгоритма и применения теории Демпстера-Шафера. Горизонтальная масштабируемость алгоритма достигается за счет отсутствия разделения ролей выполняющих его агентов.

3 Разработан горизонтально масштабируемый алгоритм фильтрации двумерных лазерных сканов. Применение этого алгоритма позволяет уменьшить среднее время обработки лазерного скана более чем на 40% за счет быстрого определения корреляции скана с предыдущими.

4 Реализовано программное обеспечение, демонстрирующее работу разработанного алгоритма и позволяющее оценить качество алгоритма на практике. Получено свидетельство о регистрации программного обеспечения для ЭВМ.

5 Определены характеристики точности, а также границы применимости на вычислительных устройствах, обладающих ограниченными ресурсами. Используемый в рамках проделанной работы алгоритм оказался точнее, чем популярное решение от Google – графовый алгоритм cartographer. Среднеквадратичная ошибка многоагентного алгоритма на дистанции 100 метров, пройденной одним агентом, при условии однократной синхронизации с другим

агентом, составила 9,4 см. Погрешность определения позиции робота сопоставима с его габаритами, что свидетельствует о высокой точности разработанного алгоритма.

Исходя из полученных результатов можно сделать вывод о том, что цель работы достигнута: разработаны масштабируемые алгоритмы для многоагентного решения задачи SLAM, применимые для низко производительных роботов. В дальнейшем возможно применение как теории Демпстра-Шафера к алгоритму фильтрации данных, так и графовых алгоритмов определения лидера для увеличения точности объединения карт (в случаях, когда в синхронизации одновременно участвует более двух агентов).

Словарь терминов

SLAM: англ. Simultaneous Localization And Mapping – задача одновременного определения собственного положения и построения карты.

Лидар: сенсор, вычисляющий расстояния до окружающих объектов при помощи лазеров и оптических систем.

Мобильный робот: передвижная платформа, оснащенная вычислительным устройством, а также лидаром – датчиком, который может измерять расстояние до препятствий, окружающих робота

Одометрия: сведения о перемещении мобильного робота, напрямую измеренные датчиками перемещения, установленными на роботе

Скан: набор точек, представленный массивом расстояний от лидара до препятствий на пути лазера, пущенными под углами, отличающимися на одинаковую величину.

Скан матчинг: процесс сопоставления сканов друг с другом или сканов с построенной картой для определения точной позиции робота, с которой снимался скан.

Скан матчер: алгоритм, реализующий скан матчинг.

Слияние карт: алгоритм построения глобальной карты, включающей в себя все, участвующие в слиянии, карты

Стая: набор автономных роботов, имеющих сопоставимое техническое оснащение для наблюдения окружающей среды в ограниченной окрестности, не имеющих иерархии и обменивающихся информацией друг с другом для совместной разметки окружающей среды.

Фильтр частиц: алгоритм численного решения задачи фильтрации путём создания нескольких гипотез о фильтруемой величине и оценки правдоподобности каждой гипотезы при получении новых наблюдений.

Фреймворк: программа, обеспечивающая и облегчающая разработку и объединение разных компонентов большого программного проекта.

Список использованной литературы

1. Филатов А. Ю. Сравнение современных лазерных алгоритмов SLAM. Филатов А. Ю., Филатов А.Ю., Гулецкий А.Т, Карташов Д. А., Кринкин К. В. // Известия ЛЭТИ. – 2018. – № 7. – С. 66-73.

2. Филатов А. Ю. Методы сравнения качества 2D-SLAM-алгоритмов. Филатов А. Ю., Филатов А. Ю., Кринкин К. В., Чен Б., Молодан Д. // Известия ЛЭТИ. – 2018. – № 7. – С. 87-95.

3. Филатов А. Ю., Кринкин К. В. Программа для вычисления взаимного расположения агентов, выполняющих алгоритм решения задачи SLAM. Санкт-Петербургский Государственный Электротехнический университет (ЛЭТИ). – ПрЭВМ в Роспатенте, 15.01.2020г. № 2020610524.

4. Akaike H. Likelihood and the Bayes procedure //Selected papers of Hirotugu Akaike. – Springer, New York, NY, 1998. – С. 309-332.

5. Andersson L. A. A., Nygards J. C-SAM: Multi-robot SLAM using square root information smoothing //2008 IEEE International Conference on Robotics and Automation. – IEEE, 2008. – С. 2798-2805.

6. Bailey T. Consistency of the EKF-SLAM algorithm //2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. – IEEE, 2006. – С. 3562-3568.

7. Bay H., Tuytelaars T., Van Gool L. Surf: Speeded up robust features //European conference on computer vision. – Springer, Berlin, Heidelberg, 2006. – С. 404-417.

8. Birk A., Carpin S. Merging occupancy grid maps from multiple robots //Proceedings of the IEEE. – 2006. – Т. 94. – №. 7. – С. 1384-1397.

9. Carpin S. Fast and accurate map merging for multi-robot systems //Autonomous Robots. – 2008. – Т. 25. – №. 3. – С. 305-316.

10. Castellanos J. A. Robocentric map joining: Improving the consistency of EKF-SLAM //Robotics and autonomous systems. – 2007. – Т. 55. – №. 1. – С. 21-29.

11. Castro D. A., Morales C. F., De la Rosa R. F. Multi-robot slam on client-server architecture //2012 Brazilian Robotics Symposium and Latin American Robotics Symposium. – IEEE, 2012. – С. 196-201.

12. Cieslewski T., Choudhary S., Scaramuzza D. Data-efficient decentralized visual SLAM //2018 IEEE International Conference on Robotics and Automation (ICRA). – IEEE, 2018. – С. 2466-2473.

13. Cole D. M., Newman P. M. Using laser range data for 3D SLAM in outdoor environments //Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. – IEEE, 2006. – С. 1556-1563.

14. Dempster A. P. The Dempster–Shafer calculus for statisticians //International Journal of approximate reasoning. – 2008. – Т. 48. – №. 2. – С. 365-377.

15. Doucet A. Rao-Blackwellised particle filtering for dynamic Bayesian networks //arXiv preprint arXiv:1301.3853. – 2013.

16. Duckietown: сайт. – URL: https://www.duckietown.org/ (дата обращения 11.05.2020). – Текст: электронный

17. Fallon M. The mit stata center dataset //The International Journal of Robotics Research. – 2013. – Т. 32. – №. 14. – С. 1695-1699.

18. Filatov A., Krinkin K. A Simplistic Approach for Lightweight Multi-Agent SLAM Algorithm. // International Journal of Embedded and Real-Time Communication Systems (IJERTCS). – 2020. – Т. 11. – №. 3. – С. 67-83.

19. Filatov A., Krinkin K. Multi-Agent SLAM Approaches for Low-Cost Platforms //2019 24th Conference of Open Innovations Association (FRUCT). – IEEE, 2019. – С. 89-95.

20. Forster C. Collaborative monocular slam with multiple micro aerial vehicles //2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. – IEEE, 2013. – С. 3962-3970.

21. Fox D. Monte carlo localization: Efficient position estimation for mobile robots //AAAI/IAAI. – 1999. – Т. 1999. – №. 343-349. – С. 2-22.

22. Grisettiyz G., Stachniss C., Burgard W. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling

//Proceedings of the 2005 IEEE international conference on robotics and automation. – IEEE, 2005. – С. 2432-2437.

23. Grisetti G. A tutorial on graph-based SLAM //IEEE Intelligent Transportation Systems Magazine. – 2010. – Т. 2. – №. 4. – С. 31-43.

24. Hess W. Real-time loop closure in 2D LIDAR SLAM //2016 IEEE International Conference on Robotics and Automation (ICRA). – IEEE, 2016. – С. 1271-1278.

25. Hol J. D., Schon T. B., Gustafsson F. On resampling algorithms for particle filters //2006 IEEE nonlinear statistical signal processing workshop. – IEEE, 2006. – С. 79-82.

26. Huletski A., Kartashov D. A slam research framework for ros //Proceedings of the 12th Central and Eastern European Software Engineering Conference in Russia. – 2016. – С. 1-6.

27. Huletski A., Kartashov D., Krinkin K. Vinyslam: an indoor slam method for low-cost platforms based on the transferable belief model //2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – IEEE, 2017. – С. 6770-6776.

28. Karrer M., Schmuck P., Chli M. CVI-SLAM—collaborative visual-inertial SLAM //IEEE Robotics and Automation Letters. – 2018. – Т. 3. – №. 4. – С. 2762-2769.

29. Kerl C., Sturm J., Cremers D. Dense visual SLAM for RGB-D cameras //2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. – IEEE, 2013. – С. 2100-2106.

30. Kim B. Multiple relative pose graphs for robust cooperative mapping //2010 IEEE International Conference on Robotics and Automation. – IEEE, 2010. – С. 3185-3192.

31. Krinkin K., Filatov A. Correlation Filter of 2D Laser Scans for Indoor Environment. // Robotics and Autonomous Systems. – 2021. – Т. 142 – С. 91-99.

32. Krinkin K. Data distribution services performance evaluation framework //2018 22nd Conference of Open Innovations Association (FRUCT). – IEEE, 2018. – C. 94-100.

33. Krinkin K. Evaluation of modern laser based indoor slam algorithms //2018 22nd Conference of Open Innovations Association (FRUCT). – IEEE, 2018. – C. 101-106.

34. Krinkin K. The scan matchers research and comparison: Monte-carlo, olson and hough //2016 19th Conference of Open Innovations Association (FRUCT). – IEEE, 2016. – C. 99-105.

35. Konolige K. Map merging for distributed robot navigation //Proceedings 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)(Cat. No. 03CH37453). – IEEE, 2003. – T. 1. – C. 212-217.

36. Lazaro M. T. Multi-robot SLAM using condensed measurements //2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. – IEEE, 2013. – C. 1069-1076.

37. Leonard J. J., Durrant-Whyte H. F. Simultaneous map building and localization for an autonomous mobile robot //IROS. – 1991. – T. 3. – C. 1442-1447.

38. Mason J., Marthi B. An object-based semantic world model for long-term change detection and semantic querying //2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. – IEEE, 2012. – C. 3851-3858.

39. Montemerlo M. FastSLAM: A factored solution to the simultaneous localization and mapping problem //Aaai/iaai. – 2002. – T. 593598.

40. Montemerlo M. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges //IJCAI. – 2003. – C. 1151-1156.

41. Moras J., Cherfaoui V., Bonnifait P. Credibilist occupancy grids for vehicle perception in dynamic environments //2011 IEEE International Conference on Robotics and Automation. – IEEE, 2011. – C. 84-89.

42. Mur-Artal R., Montiel J. M. M., Tardos J. D. ORB-SLAM: a versatile and accurate monocular SLAM system //IEEE transactions on robotics. – 2015. – Т. 31. – №. 5. – С. 1147-1163.

43. Nettleton E. Decentralised SLAM with low-bandwidth communication for teams of vehicles //Field and Service Robotics. – Springer, Berlin, Heidelberg, 2003. – С. 179-188.

44. O'Kane J. M. A gentle introduction to ROS. – 2014.

45. Özkucur N. E., Akın H. L. Cooperative multi-robot map merging using fast-SLAM //Robot Soccer World Cup. – Springer, Berlin, Heidelberg, 2009. – С. 449-460.

46. Pfingsthorn M., Slamet B., Visser A. A scalable hybrid multi-robot SLAM method for highly detailed maps //Robot Soccer World Cup. – Springer, Berlin, Heidelberg, 2007. – С. 457-464.

47. Richardson M., Wallace S. Getting started with raspberry PI. – " O'Reilly Media, Inc.", 2012.

48. Rublee E. ORB: An efficient alternative to SIFT or SURF //2011 International conference on computer vision. – Ieee, 2011. – С. 2564-2571.

49. Schmuck P., Chli M. Multi-uav collaborative monocular slam //2017 IEEE International Conference on Robotics and Automation (ICRA). – IEEE, 2017. – С. 3863-3870.

50. Scovanner P., Ali S., Shah M. A 3-dimensional sift descriptor and its application to action recognition //Proceedings of the 15th ACM international conference on Multimedia. – 2007. – С. 357-360.

51. Smith R., Self M., Cheeseman P. Estimating uncertain spatial relationships in robotics //Autonomous robot vehicles. – Springer, New York, NY, 1990. – С. 167-193.

52. Statistics. Gartner. Autonomous vehicles: сайт. – URL : https://www.gartner.com/en/information-technology/glossary/autonomous-vehicles (дата обращения 11.05.2020). – Текст: электронный

53. Trusted advisor for enterprises Gartner: сайт. – URL:https://www.gartner.com/en (дата обращения 11.05.2020). – Текст: электронный

54. Thrun S., Montemerlo M. The graph SLAM algorithm with applications to large-scale mapping of urban structures //The International Journal of Robotics Research. – 2006. – Т. 25. – №. 5-6. – С. 403-429.

55. Turtlebot: сайт. – URL: https://www.turtlebot.com/ (дата обращения 11.05.2020). – Текст: электронный

56. Willow garage. Обзор робота pr2-bot : сайт – URL: http://www.willowgarage.com/pages/pr2/overview (дата обращения 11.05.2020). – Текст: электронный

57. Yager R. R. On the Dempster-Shafer framework and new combination rules //Information sciences. – 1987. – Т. 41. – №. 2. – С. 93-137.

58. Zhou X. S., Roumeliotis S. I. Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case //2006 IEEE/RSJ international conference on intelligent robots and systems. – IEEE, 2006. – С. 1785-1792.

Приложение 1

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2020610524

Программа для вычисления взаимного расположения агентов, выполняющих алгоритм решения задачи SLAM

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)» (СПбГЭТУ «ЛЭТИ») (RU)***

Авторы: ***Филатов Антон Юрьевич (RU),***
Кринкин Кирилл Владимирович (RU)

Заявка № **2019667191**
Дата поступления **23 декабря 2019 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***15 января 2020 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев