Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»
Физтех-школа Прикладной Математики и Информатики
Научно-образовательный центр когнитивного моделирования

Направление подготовки / специальность: 09.04.01 Информатика и вычислительная техника
Направленность (профиль) подготовки: Методы и технологии искусственного интеллекта

АЛГОРИТМ ПОСТРОЕНИЯ ГРАФА ВИДИМОСТИ ДЛЯ БЫСТРОГО ПЛАНИРОВАНИЯ ТРАЕКТОРИИ МОБИЛЬНОГО РОБОТА В СРЕДЕ С ПРЕПЯТСТВИЯМИ

(магистерская диссертация)

Студент:
Касмынин Кирилл Анатольевич

_____
(подпись студента)

Научный руководитель:
Миронов Константин Валерьевич,
канд. техн. наук

_____
(подпись научного руководителя)

Консультант *(при наличии)*:
_____
_____
(подпись консультанта)

Москва 2025

Содержание

1 Введение **5**

2 Постановка задачи **8**

3 Обзор литературы и теоретические основы **10**

3.1 Методы планирования пути 10

3.1.1 Методы на основе графов 11

3.1.2 Методы клеточной декомпозиции 12

3.1.3 Методы потенциальных полей 13

3.1.4 Интеллектуальные методы 14

3.2 Граф видимости и его применение 15

3.3 Методы выделения препятствий и их аппроксимации 16

3.4 Оптимизация вычислений и векторизация 18

4 Методология построения графа видимости **20**

4.1 Программная архитектура 20

4.2 Извлечение и аппроксимация препятствий 21

4.3 Построение графа видимости 23

4.3.1 Алгоритм построения графа видимости 26

4.3.2 Подходы к удалению лишних ребер 29

4.3.3 Преимущества графа видимости в планировании пути 32

4.3.4 Механизм перепланирования 33

4.4 Реализация узла ROS 34

5 Методология векторизации проверки пересечений **35**

5.1 Векторизация операций проверки пересечений 35

5.2 Подходы реализации: AVX и NumPy 39

6 Эксперименты и анализ результатов **41**

6.1 Сравнение методов планирования: влияние аппроксимации и векторизации 41

6.2 Сравнение с Probabilistic Roadmaps (PRM) 49

6.3 Сравнение производительности NumPy и AVX 51

6.4 Анализ производительности при перепланировании 52

7 Заключение **52**

Аннотация

В работе исследовано ускорение планирования траектории в средах с препятствиями на основе графа видимости. Предложен векторизованный подход к проверке пересечений ребер, позволяющий одновременно обрабатывать множество ребер и тем самым существенно сокращать время вычислений. Дополнительно интегрированы различные подходы к извлечению и аппроксимации полигонов препятствий, включая классические и нейросетевые подходы. Эксперименты подтвердили ускорение построения графа до десятков миллисекунд на картах с большим числом препятствий. Проведено сравнение с методами PRM, A* и Theta*, показавшее превосходство векторизованного графа видимости по скорости построение и перепланирования при сохранении гарантии получения кратчайшего пути. Предложенный подход обеспечивает ускорение в 5-60 раз по сравнению с другими методами планирования. Реализация на Python/NumPy и C++/AVX, а также интеграция в ROS-узел подтверждают практическую применимость подхода для задач планирования траекторий мобильных роботов.

1 Введение

Планирование траектории — это ключевая задача автономной робототехники. Проблема планирования пути в условиях среды с препятствиями является актуальным и сложным направлением исследований [38]. Для решения этой проблемы используются алгоритмы планирования безопасных, то есть свободных от столкновений траекторий для объекта среди множества препятствий [37]. Один из наиболее перспективных подходов — граф видимости [5], который позволяет решить эту проблему, но его построение и извлечение полигонов препятствий из карт сопряжены со значительными вычислительными трудностями.

Мобильные роботы применяются для автономной доставки, выполнения бытовых и сервисных задач, а также для проведения работ в труднодоступных или опасных зонах. Надежная навигация и оперативное планирование траекторий в среде с препятствиями имеют решающее значение для повышения эффективности промышленной автоматизации, обеспечения безопасности беспилотных транспортных средств и расширения функциональных возможностей роботизированных систем.

Планирование маршрута является одним из наиболее важных аспектов навигационных процессов. Оно включает в себя определение маршрута без столкновений в заданных условиях. Системы планирования маршрута должны создавать маршруты от начальной точки до конечной точки безопасно и эффективно, учитывая такие факторы, как время, расстояние, энергозатраты и другие ограничения [36].

Граф видимости, как концепция, существует уже несколько десятилетий. Он предоставляет метод планирования маршрута, отображая взаимосвязи видимости между точками в заданной среде. Преимущество графа видимости заключается в его способности гарантировать оптимальный путь.

Алгоритмы планирования маршрута направлены на определение наиболее эффективного маршрута или его разумного приближения, оптимизируя одну или несколько целевых функций. Например, маршрут может сократить время в пути, что имеет решающее значение для таких задач, как поисково-спасательные операции, или снизить энергопотребление, обеспечивая эффективное использование ресурсов робота. Наряду с поиском эффективного пути, не менее важна способность быстро его планировать [41].

Граф видимости [5] основан на построении графа, в котором узлы представляют вершины препятствий, а ребра соединяют видимые вершины, обеспечивая пути без

столкновений.

С точки зрения вычислений одним из наиболее сложных этапов построения графа видимости является процесс проверки пересечений. Он включает в себя определение: пересекаются ли ребра видимости, соединяющие вершины препятствий, с ребрами препятствий. Основная проблема заключается в том, что необходимо проводить попарное сравнение большого количества возможных ребер видимости и ребер препятствий — это приводит к вложенным циклам и значительным потерям производительности. Для решения данной проблемы в работе предложен подход векторизации проверок пересечений, что уменьшает зависимость от итерационных циклов и минимизирует вычислительные затраты.

Необходимо обеспечить точное обнаружение и соответствующую аппроксимацию полигонов, извлеченных из сетки занятости или изображения, будь то с использованием классических методов обработки изображений или нейронных сетей, таких как PolyWorld [43], или путем предварительного сегментирования препятствий, например, с помощью U-Net [21], для получения бинарной маски, а затем извлечение многоугольных контуров. Правильное представление полигонов имеет решающее значение для эффективного и точного построения графа видимости.

Планирование траектории в среде с препятствиями может опираться на различные алгоритмы. В данном исследовании основной упор сделан на граф видимости в сочетании с подходами к его ускорению, а также с точно заданными и эффективно аппроксимированными полигонами. Интеграция графа видимости с векторизованными операциями проверки пересечений решает задачи, связанные с вычислительной эффективностью и большими временными затратами при планировании траекторий. Таким образом, наш подход сравнивается не только с популярными решеточными алгоритмами, такими как A* [2] и Theta* [14], но и с методами на основе графов, например, метод построения вероятностных дорожных карт — Probabilistic Roadmaps [10].

Цель данной работы заключается в разработке и реализации эффективного алгоритма построения графа видимости, позволяющего быстро планировать кратчайшие траектории в среде с препятствиями, а также исследовании и применении новых методов ускорения построения графа видимости.

Для достижения основной цели были поставлены и решены следующие задачи:

- Разработан метод частичной и полной векторизации операций проверки пересе-

чений при построении графа видимости, что позволило существенно сократить время выполнения алгоритма.

- Выполнена интеграция алгоритма построения графа видимости с ROS, обеспечено использование стандартных представлений данных о среде и положении робота, а также реализован интерфейс для задания целевых точек и визуализации полученных траекторий.

- Реализованы различные версии и проведено сравнение этих реализаций алгоритма с использованием библиотек NumPy на Python и AVX-инструкций на языке C++, оценена эффективность каждого подхода.

- Предложены и реализованы два подхода к удалению лишних ребер графа видимости: метод на основе анализа выпуклости углов препятствий и метод генерации вспомогательных ребер для проверки корректности соединений.

- Выполнена модификация структуры и оптимизация реализации алгоритма построения графа видимости, что позволило существенно улучшить его вычислительную производительность.

- Реализован механизм перепланирования, показано, что добавление новых стартовых и целевых вершин выполняется за десятки миллисекунд, что значительно быстрее полного пересчета маршрута.

- Проведено сравнение скорости построения графа видимости с использованием различных подходов к извлечению и аппроксимации полигонов препятствий. Проведены эксперименты, которые показали преимущество предложенного подхода по сравнению с популярными методами клеточной декомпозиции (A*, Theta*) и вероятностным (Probabilistic Roadmaps) методом.

Таким образом, в работе предложен комплексный подход, включающий алгоритмы ускоренного построения графа видимости и эффективные методы обработки исходных данных, а также выполнено экспериментальное сравнение с существующими популярными методами, подтверждающее практическую эффективность разработанного решения.

2 Постановка задачи

Проблема определения кратчайшего пути возникла довольно давно и охватывает множество алгоритмов и методов. Выбор методов для решения этой задачи зависит от конкретных условий постановки, области применения и доступных инструментов для разработки. Исследование новых методов остается актуальным в связи с развитием робототехники, что требует более оперативных и точных решений. Кроме того, многие существующие алгоритмы и методы решения задачи поиска маршрута характеризуются избыточной вычислительной сложностью и требовательностью к точным алгебраическим моделям препятствий [23].

Задача, рассматриваемая в этом исследовании, заключается в разработке эффективного алгоритма построения графа видимости для планирования траектории мобильного робота в среде с препятствиями. Основная цель работы состоит в ускорении процедуры построения графа видимости путем применения методов векторизации операций проверки пересечений и аппроксимации препятствий.

Среди методов планирования траекторий мобильных роботов граф видимости обеспечивает нахождение кратчайшего пути между стартовой и целевой точками. Однако высокая вычислительная сложность построения графа видимости ранее существенно ограничивала его применение. Рост вычислительных мощностей, а также разработка методов векторизации проверок пересечений и аппроксимации препятствий позволяют значительно ускорить построение графа видимости.

Задача планирования траектории сводится к решению трех взаимосвязанных подзадач:

1) Извлечение полигонов препятствий.
2) Построение графа видимости.
3) Поиск кратчайшего пути на этом графе.

Далее приведены основные определения, используемые в постановке задачи.

Карта занятости — двумерная дискретная сетка, где каждая ячейка обозначена как занятая (принадлежит препятствию) или свободная. Обычно представляется матрицей значений 0 и 1, где 1 указывает на наличие препятствия, а 0 на свободное пространство.

Препятствие — связная область ячеек (или пикселей) карты занятости или бинарной маске, помеченных как занятые. Область препятствия объединяет все смежные ячейки со значением 1.

Свободное пространство — множество всех ячеек карты занятости со значением 0. Именно в этой области робот может перемещаться.

Полигон препятствия — упрощенная полигональная модель препятствия. На карте занятости контуры каждого препятствия аппроксимируются многоугольником P_i, чьи вершины лежат на границе исходного препятствия.

Для извлечения полигонов препятствий необходимо определить их области. Пусть на вход алгоритму подается карта или изображение I, на котором необходимо идентифицировать множество препятствий $O = \{O_1, O_2, ..., O_n\}$, где каждое препятствие представлено набором пикселей или ячеек сетки, соответствующих занятым областям.

Результатом работы алгоритма является множество полигонов препятствий $P = \{P_1, P_2, ..., P_m\}$ и построенный на их основе граф видимости $G(V, E)$, используемый для поиска кратчайшего пути.

Каждое препятствие $O_i \subset \mathcal{E}_{\text{obs}}$, первоначально заданное в виде множества пикселей или ячеек карты, заменяется на многоугольник P_i. Многоугольник P_i является упрощенным полигоном исходного препятствия O_i.

Определим среду $\mathcal{E}$ следующим образом. Среда состоит из двух непересекающихся подмножеств: множества свободного пространства $\mathcal{E}_{\text{free}} \subset \mathcal{E}$, где робот может безопасно перемещаться, и множества препятствий $\mathcal{E}_{\text{obs}} \subset \mathcal{E}$, где движение невозможно. При этом выполняется условие: $\mathcal{E} = \mathcal{E}_{\text{free}} \cup \mathcal{E}_{\text{obs}}$, $\mathcal{E}_{\text{free}} \cap \mathcal{E}_{\text{obs}} = \emptyset$.

Задача сводится к построению графа видимости $G = (V, E)$, вершины которого включают в себя начальную точку s, целевую точку g, а также вершины многоугольников препятствий V_{poly}, так что $V = \{s, g\} \cup V_{\text{poly}}$. Множество ребер E состоит из соединений между вершинами, которые не пересекают препятствия среды $\mathcal{E}_{\text{obs}}$.

Ключевая вычислительная сложность при построении графа видимости заключается в необходимости проверки пересечения каждого кандидата ребра видимости $e = (v_i, v_j)$ с препятствиями:

$$\text{Intersect}(e, \mathcal{E}_{\text{obs}}) = \begin{cases} \text{True}, & \text{если } e \cap \mathcal{E}_{\text{obs}} \neq \emptyset, \\ \text{False}, & \text{иначе.} \end{cases}$$

Для решения указанной вычислительной проблемы в рамках исследования предлагается подход, включающий извлечение препятствий $P = \{P_1, P_2, \ldots, P_n\}$, где каждый P_i является многоугольником, определяемым его вершинами $\{p_1, p_2, \ldots, p_k\}$. Затем их эффективную аппроксимацию для уменьшения вычислительных затрат, а

после чего полную векторизацию операции проверки пересечений, позволяющую отказаться от последовательных попарных сравнений и тем самым ускорить построение графа видимости.

Таким образом, граф видимости можно определить так:

$$G = (V, E), \quad \text{где } E = \{(v_i, v_j) \mid v_i, v_j \in V,\ \text{Intersect}((v_i, v_j), P_k) = \text{False } \forall k\}.$$

Конечной целью построения графа видимости является определение кратчайшего пути от начальной точки s до целевой точки g по ребрам графа, учитывающим имеющиеся препятствия в заданной рабочей области U. Кратчайший путь на графе видимости — последовательность вершин $P = (v_0, \ldots, v_k)$, где $v_0 = s$, $v_k = g$ и суммарная длина ребер $\sum \|v_i - v_{i+1}\|$ минимальна. Все ребра графа соединяют пары взаимно видимых точек без пересечения препятствий и формируют тем самым сеть возможных маршрутов. Рабочая область представляет собой двумерное пространство, ограниченное прямоугольной областью с заданными границами $x_{\min} \leqslant x \leqslant x_{\max}$ и $y_{\min} \leqslant y \leqslant y_{\max}$.

В следующих разделах будут подробно описаны методы извлечения полигонов, построения графа видимости и поиска кратчайшего маршрута.

3 Обзор литературы и теоретические основы

Разработка эффективных алгоритмов планирования траекторий в среде с препятствиями является ключевой задачей в области автономной робототехники. Существуют различные подходы к решению этой задачи, каждый из которых обладает своими преимуществами и ограничениями. В данном разделе рассматриваются основные методы планирования траекторий, особенности графа видимости, методы сегментации препятствий, а также современные подходы к оптимизации вычислений.

3.1 Методы планирования пути

Путь должен прокладываться от исходной точки к целевой точке и обеспечивать движение робота, учитывая возможные препятствия на пути. Далее среди всех потенциальных маршрутов требуется выбрать такой, который, в определенном контексте, будет оптимальным [28].

Задача планирования пути включает в себя определение последовательности состояний для подвижного объекта, обеспечивающих его перемещение из текущего

положения в целевое. Под состоянием понимается минимально необходимый набор переменных, точно определяющий положение объекта в момент времени t [44].

Переход в целевое состояние требует соответствия физическим ограничениям, включая время, управляющие воздействия и отсутствие препятствий. Оптимальность траектории подразумевает минимизацию или максимизацию критерия качества, такого как время движения или длина пути [44].

Существует множество методов планирования пути, и их можно классифицировать по различным характеристикам. По условиям окружающей среды методы планирования можно разделить по применению в статической и динамической средах [31]. По полноте информации об окружающей среде существуют методы с полной информацией, когда применяется глобальное планирование пути, и — с неполной информацией, где среда известна только в непосредственной близости от робота, в этом случае применяется локальное планирование пути [28].

В литературе выделяют следующие методы планирования: методы на основе графов, методы на основе клеточной декомпозиции, методы потенциальных полей, методы на основе интеллектуальных технологий. Во многих из этих методов результатом планирования является последовательность путевых точек, соединяющих начальную и целевую позиции робота, что обеспечивает основу для последующего движения и управления [28].

3.1.1 Методы на основе графов

Планирование траектории на плоскости рассматривается как задача нахождения кратчайшего пути на взвешенном графе. В данном контексте вершины графа представляют координаты точек на плоскости, а ребра соответствуют элементарным траекториям, таким как отрезки прямых. Элементарные траектории ассоциируются с характеристиками, такими как длина, сложность и опасность, которые могут быть формализованы как веса ребер графа [20].

Общая схема алгоритма построения кратчайшей траектории между двумя точками на плоскости включает два основных шага: построение взвешенного графа и поиск кратчайшего пути на построенном графе [20].

Алгоритмы, основанные на графах, получили широкое распространение в задачах планирования траекторий [22]. К данной группе относятся методы построения графа видимости [5], диаграммы Вороного [9], быстрорастущие случайные деревья

(RRT) [13], а также метод вероятностных дорожных карт (PRM) [10].

Граф видимости представляет собой графовую модель, применяемую для решения задачи планирования траектории в пространствах с многоугольными непроходимыми областями. В данной модели вершины соответствуют вершинам многоугольников и ключевым точкам (начальная и конечная), а ребра графа соединяют эти вершины, избегая пересечения с внутренними областями многоугольников [20, 5].

Метод вероятностной дорожной карты (PRM) предлагает альтернативный подход, при котором сначала в свободном пространстве выбирается разреженный набор конфигураций, не допускающих столкновений, а затем они соединяются локальным планировщиком [10]. Этот метод применяется для решения задач как локального, так и глобального планирования. Процесс планирования включает в себя два этапа. На первом этапе создается карта путевых точек графа путем случайной генерации заданного числа узлов. Каждая новая точка проверяется на пересечение с препятствиями, и сохраняются только те точки, которые генерируются в области карты, свободной от препятствий. На втором этапе создаются свободные от столкновений ребра между заранее сгенерированными вершинами. Затем по полученному графу осуществляется поиск пути между любыми двумя узлами [44].

3.1.2 Методы клеточной декомпозиции

Наиболее широко используемый способ планирования движения заключается в использовании методов пространственной декомпозиции. Этот метод предполагает разделение областей сцены на простые регионы с помощью различных методов декомпозиции. В классическом понимании вся сцена покрывается сеткой с фиксированным шагом, где каждой ячейке присваивается состояние занята или свободна, после чего поиск маршрута происходит путем проверки соседних свободных ячеек и последовательному переходу между ними [22].

Одним из методов применения данного подхода является использование методов регулярной пространственной декомпозиции. В данном случае разделяется весь объем сцены с использованием сетки, в которой ячейки имеют фиксированный размер и ориентированы по осям координат. Каждой ячейке присваивается статус занятости, указывающий является ли ячейка занятой или свободной. Построение маршрута осуществляется анализом соседних свободных ячеек, выбором направлений для перемещения и проверкой доступности свободных ячеек сетки для перемещаемого объ-

екта. Таким образом, вместо использования исходной точной геометрической модели сцены применяется ее упрощенное дискретное представление [22].

Этот метод относительно легок в реализации, однако требует значительных вычислительных ресурсов из-за необходимости детализированной дискретизации всего пространства сцены и обеспечения приемлемой точности маршрута. Методы регулярной адаптивной декомпозиции пространства, например, использующие октодеревья занятости [16] обеспечивают более экономичное представление сцены благодаря рекурсивной процедуре декомпозиции.

В решении задачи планирования, использующем методы клеточной декомпозиции или графов, часто требуется дополнительный этап поиска путей на графе [28].

К стратегиям поиска относятся методы, такие как поиск в ширину, поиск в глубину, ограниченный поиск глубины, поиск в глубину с итеративным углублением и двунаправленный поиск. Стратегии описаны в данной работе [12].

Примером эвристической стратегии является алгоритм A* [2], представляющий собой расширение алгоритма Дейкстры [1]. Алгоритм A* достигает более высокой производительности по времени с помощью эвристики. Целевая функция алгоритма A* реализуется как сумма двух других: функции стоимости достижения рассматриваемой точки из начальной и эвристической функции оценки расстояния от рассматриваемой точки до конечной точки. Эвристическая функция не должна переоценивать расстояние до целевой точки. Без второй составляющей алгоритм A* сводится к алгоритму Дейкстры [28].

Однако, у алгоритмов на основе клеточной декомпозиции имеются существенные ограничения, такие как ограниченное количество направлений, вызванное структурой сетки. Разработаны различные методы преодоления этого ограничения, включая алгоритмы D*, который описан в данной работе [26] и алгоритм Theta*, описанный в работе [14].

3.1.3 Методы потенциальных полей

Метод искусственных потенциальных полей задает навигационное векторное поле через скалярный потенциал $U(q)$, представляющий собой сумму притягивающей составляющей, ориентированной на целевую точку, и отталкивающей, порождаемой препятствиями, а направление движения задается градиентом этой функции [7]. Простая суперпозиция приводит к появлению локальных минимумов, для их пре-

одоления вводятся случайные возмущения и динамическое демпфирование. Далее навигационные функции формируют потенциальную поверхность с единственным глобальным минимумом, опираясь на гармонические поля или супербиквадратные потенциалы [8].

В работе [33] проанализированы разные способы конструирования притягивающих полей, включая модификацию потенциала до постоянной по модулю силы, а также предложен новый алгоритм обхода препятствий.

Также решающее значение имеет настройка коэффициентов притяжения и отталкивания, учет формы препятствий при построении профиля поля и применение приемов сглаживания градиента для обеспечения плавности траекторий.

3.1.4 Интеллектуальные методы

Для решения задачи планирования траекторий во все большей степени применяются алгоритмы, основанные на методах искусственного интеллекта. Одним из таких подходов является использование генетических алгоритмов [18]. Параллельно развиваются методы, сочетающие генетические алгоритмы с нечеткой логикой, что позволяет более гибко настраивать правила принятия решений при неопределенности восприятия [28].

Нейронные сети позволяют обучать локальные навигационные политики. Так, в данной работе [30] PRM объединили с обучением с подкреплением для получения стратегий навигации в больших средах. В работе [34] представлен подход, в котором глобальное планирование сочетается с обучением с подкреплением для обхода динамических препятствий. Алгоритм строит первоначальную дорожную карту, после чего агент учится выбирать промежуточные точки маршрута, учитывая изменения окружающей среды и положение других роботов.

Нечеткая логика широко используется для учета неточностей сенсорных данных и принятия гибких решений. В работе [32] применен алгоритм Q-обучения для автоматической настройки нечетких контроллеров, которые отвечают за поиск цели и обход препятствий.

Таким образом, интеллектуальные методы охватывают широкий спектр техник — от эволюционных алгоритмов и нечеткой логики до современных подходов на основе обучения с подкреплением. Их объединяет способность обучаться на данных и адаптироваться к сложным и изменчивым условиям среды.

Рассмотрены многочисленные методы, применимые для решения задач планирования траекторий. Эти методы играют ключевую роль в управлении движением мобильных роботов, обеспечивая навигацию в средах с препятствиями.

3.2 Граф видимости и его применение

Графы видимости представляют собой графовую модель, применяемую для решения задачи планирования траектории в пространствах с многоугольными непроходимыми областями. В данной модели вершины соответствуют вершинам многоугольников и ключевым точкам (начальная и конечная), а ребра графа соединяют эти вершины, избегая пересечения с внутренними областями многоугольников.

Планирование в графе видимости начинается с рассмотрения мобильного робота и препятствий, которые представлены многоугольниками. Выбирается какая-то одна точка на роботе, называемая контрольной точкой, и рассматривается, где она может находиться так, чтобы избежать столкновений с препятствиями. Создается много запрещенных областей, которые представляют собой места, где контрольная точка не должна находиться, чтобы избежать столкновений. Таким образом, поиск безопасного пути осуществляется для одной точки, а не для всего рассматриваемого объекта. Происходит построение графа, который показывает, как эти запрещенные области связаны друг с другом. Затем происходит поиск пути через этот граф, который не проходит через запрещенные области [5].

При использовании графа видимости для рассмотрения движущегося объекта как точки, применяется подход увеличения областей препятствий на размер мобильного робота. Операция увеличения препятствий позволяет представить объект как точку и создать новый набор препятствий [5].

Граф видимости может применяться для планирования траектории различных типов роботов. Также в области существуют подходы, сочетающие графы видимости с другими структурами и алгоритмами. Например, в работе [39] описан алгоритм планирования маршрута для беспилотных надводных судов, основанный на комбинировании представления среды в виде квадродерева и графа видимости.

В статье [35] представлен метод планирования маршрутов на основе Essential Visibility Graph с динамическим обновлением графа в скользящем временном окне для учета движущихся объектов. Алгоритм выполняет поиск минимального пути в EVG и периодически перестраивает связи между вершинами при появлении новых

препятствий для безопасной навигации в реальном времени.

В работе [40] предложен гибридный метод планирования траектории двуногого робота в статической среде, сочетающий многоузловой RRT и граф видимости. Сначала формируются несколько деревьев RRT от начальной, целевой и промежуточных точек, что обеспечивает широкий охват пространства поиска, затем полученный маршрут сглаживается и оптимизируется с помощью графа видимости.

В работе [29] предложен алгоритм Minimal Construct, который для каждой пары точек на полигональной карте строит лишь ту часть графа видимости, что реально влияет на маршрут. Сначала эвристически отбираются перспективные ребра вокруг препятствий, затем на полученной частичной структуре выполняется поиск кратчайшего пути, что позволяет избежать полной генерации графа и сократить количество проверок столкновений.

Для безопасного планирования движения квадрокоптеров в среде с подвижными препятствиями разработана динамическая дорожная карта свободного пространства, основанная на концепции графа видимости. Метод непрерывно формирует и обновляет множество выпуклых областей свободного пространства, соединяя их вершины ребрами видимости [42].

В работе [38] предложен метод, динамически обновляющий глобальный граф видимости при исследовании среды. Система извлекает контурные точки препятствий, формирует полигоны и поддерживает две структуры: локальную, для быстрой генерации новых видимых ребер, и глобальную, которая хранит полный граф. Здесь удаляются ребра, которые закрываются вновь обнаруженными препятствиями и добавляются ребра при открытии новых областей.

Графы видимости обеспечивают гарантированную оптимальность маршрутов. Они доказали свою эффективность для планирования траекторий в различных условиях. Перспективным направлением развития является дальнейшая оптимизация вычислительной сложности при сохранении гарантий оптимальности маршрутов.

3.3 Методы выделения препятствий и их аппроксимации

В контексте извлечения полигонов препятствий из изображения, представляющего карту, было рассмотрено несколько подходов. Существует множество методов для получения ребер препятствий, подходящих для построения графов видимости. Классические методы компьютерного зрения извлекают контуры из бинарной кар-

ты занятости, а нейросетевые методы могут выполнять сегментацию препятствий и последующее выделение полигонов для получения требуемых вершин.

Один из подходов предполагает использование методов обработки изображений для извлечения полигонов из бинарного изображения. Сначала бинарная карта проецируется на модель среды, после чего производится расширение контурных точек с учетом габаритов мобильного робота. Затем из размытого изображения извлекаются граничные точки и анализируется топологическое распределение этих точек [6]. После чего извлеченные граничные точки используются для формирования замкнутых многоугольников, представляющих препятствия в окружающей среде.

После формирования замкнутых многоугольников начинается этап их аппроксимации. Этот этап направлен на снижение числа вершин при сохранении общей формы препятствий и анализ локальной кривизны на основе внутренних углов между смежными ребрами [4].

Эти методы обеспечивают систематический подход к извлечению полигонов препятствий из бинарных изображений и их аппроксимации для эффективного представления в алгоритмах планирования маршрута. Кроме того, исходное бинарное изображение обычно формируется с помощью моделей сегментации или других методов предварительной обработки для точного определения препятствий.

Также могут быть применены нейросетевые подходы, например, PolyWorld, который предназначен для прямого извлечения вершин зданий из изображений и соединения их в полигоны. Используя нейронную сеть, модель выделяет углы зданий и предсказывает силу связи между ними [43].

Нейросетевое выделение полигонов можно разделить на два подхода: использование моделей сегментации и методов, которые напрямую возвращают полигоны.

Модели сегментации, такие как U-Net [21], Mask R-CNN [25] и DeepLabv3+ [27], формируют детальные бинарные маски, позволяющие точно выделить область каждого препятствия. Для превращения полученной растровой границы в многоугольник применяется алгоритм обхода контура [6], после чего полученные полигоны аппроксимируются, наиболее известным из которых является алгоритм Дугласа–Пекера [4]. Такой подход дает высокую точность и универсальность, так как нейросетевые модели могут дообучаться под конкретные классы препятствий, однако такой подход может требовать нескольких этапов пост-обработки, а также затраты времени на обучение и инференс модели.

Также существует подход, который возвращает полигоны напрямую. Примером

такого решения является PolyWorld [43]. Используя эту модель сначала выявляются потенциальные углы зданий, затем оценивается вероятность существования ребра между каждой парой вершин. Полученный набор связей формирует замкнутые контуры, которые сразу можно рассматривать как полигональные модели препятствий. Экспериментальные результаты показывают, что прямой векторный вывод обеспечивает точное соответствие реальной форме объектов и сокращает количество ошибок, связанных с разрывами или избыточными вершинами.

Оба нейросетевых подхода решают общую задачу — получение компактного и геометрически корректного полигонального описания препятствий, пригодного для дальнейшего использования в алгоритмах планирования маршрута. Сегментационные сети обеспечивают гибкость и простоту обучения, а модели прямого выделения полигонов минимизируют этапы пост-обработки.

3.4 Оптимизация вычислений и векторизация

В задачах построения графа видимости основным узким местом остается операция проверки пересечений между ребрами и препятствиями. При увеличении числа вершин количество потенциальных пар для проверки квадратично возрастает, что делает необходимым применение методов оптимизации вычислений. Одним из таких методов является векторизация, позволяющая обрабатывать несколько операций одновременно.

Современные процессоры поддерживают наборы инструкций SIMD, позволяющие применять одну операцию сразу к нескольким элементам данных. Использование наборов команд, например, как AVX, дает возможность выполнять операции над векторизованными данными параллельно, что обеспечивает значительный прирост производительности по сравнению со стандартными реализациями [19]. В контексте проверки пересечений SIMD позволяет одновременно анализировать несколько пар ребер вместо последовательной обработки. В настоящем исследовании рассмотрены два наиболее практичных способа векторизации на CPU. Кроме того, существуют и альтернативные решения, включая перенос вычислений на GPU [15].

Реализация на основе NumPy. Благодаря массивам и функциям NumPy данный подход является платформонезависимым и эффективным в средах, где инструкции AVX недоступны. NumPy выполняет операции с массивами, обеспечивая быстрые и масштабируемые векторизованные вычисления.

Реализация на основе AVX инструкций. Расширенные векторные инструкции (Advanced Vector Extensions, AVX) значительно ускоряют вычисления за счет одновременной обработки нескольких операций с плавающей точкой в архитектуре SIMD (Single Instruction, Multiple Data). Метод особенно полезен для массовых проверок пересечений на совместимых процессорах.

Обе стратегии пригодны для построения графа видимости: NumPy обеспечивает простоту и гибкость, тогда как AVX дает дополнительный прирост скорости в средах с поддержкой соответствующего оборудования.

Для эффективного применения SIMD вычисления необходимо явно переформулировать таким образом, чтобы они работали параллельно с ребрами графа. Этого удается достичь путем преобразования координат всех ребер в векторные представления. Вместо попарного вычисления пересечений во вложенных циклах формируются массивы вершин ребер, которые сохраняются в виде массивов NumPy или массивов, выровненных под AVX.

Такое векторное представление позволяет использовать две стратегии реализации. Первая — реализация на основе NumPy, где расчеты пересечений выполняются с помощью функций NumPy. NumPy построен на оптимизированных C-интерфейсах, и многие операции уже выполняются с использованием SIMD-инструкций. Стандартные арифметические операции поддерживают встроенные функции AVX [24], поэтому векторизованный код на NumPy может обеспечивать скорости, сопоставимые с ручной реализацией на AVX.

Вторая стратегия опирается на ручную векторизацию с использованием AVX, когда данные отрезков явно загружаются в регистры AVX (например, `m256`), и все операции реализуются с помощью встроенных функций AVX. Такой подход предоставляет более точный контроль над выравниванием данных и производительностью на уровне регистров.

Advanced Vector Extensions (AVX) представляют собой набор расширений инструкций SIMD, введенных компанией Intel для повышения производительности приложений с высокой интенсивностью вычислений. AVX увеличивает ширину векторных регистров (до 256 бит в AVX и еще больше в AVX-512), позволяя выполнять параллельную обработку нескольких элементов данных за одну команду. Эта возможность позволяет ускорить выполнение арифметических операций [17].

Как реализация на основе NumPy, так и реализация на основе AVX обеспечивают эффективную векторизацию операций пересечения. Это значительно ускоряет

построение графа видимости. Предлагаемый в работе подход преобразует координаты ребер в векторные представления, что позволяет эффективно использовать реализацию с использованием NumPy и AVX.

4 Методология построения графа видимости

4.1 Программная архитектура

Задача планирования траектории на плоскости между неподвижными точками является одной из ключевых задач, изучаемых в робототехнике. Широкий спектр применений разработанных методов охватывает область разработки и проектирования беспилотных транспортных средств и роботизированных систем, способных к автономной работе в окружающей среде.

Общая схема алгоритма построения кратчайшей траектории между двумя точками на плоскости включает в себя два основных этапа: построение взвешенного графа и нахождение кратчайшего пути на построенном графе. В этом контексте длина построенной траектории рассматривается как основной критерий оптимальности [20].

Процесс планирования на основе графа видимости включает три последовательно выполняемых этапа. Сначала из исходного описания среды (карты занятости или изображения) извлекаются и аппроксимируются контуры препятствий в виде полигонов. Затем на основе полученных препятствий происходит построение графа видимости. На заключительном этапе по этому графу выполняется поиск кратчайшего пути от стартовой точки до цели.

Решение реализовано в виде набора взаимосвязанных модулей. Модуль извлечения препятствий отвечает за преобразование входных данных (OccupancyGrid, бинарное изображение или результаты сегментации) в список упрощенных полигональных моделей. Модуль построения графа видимости принимает эти полигоны и строит ребра между всеми парами взаимовидимых вершин, вес ребра равен длине между вершинами. При этом реализована векторизованная проверка пересечений для ускорения построения графа. Далее модуль интеграции ключевых точек добавляет в граф стартовую и целевую позиции, и на их основе расширяет уже построенный граф видимости. После чего модуль поиска пути, в данном случае алгоритм Дейкстры, находит кратчайший маршрут в этой структуре. В итоге выводим последовательность

точек найденного пути, а также, в зависимости от способа запуска, отображается построенный граф видимости и найденный кратчайший путь, используя Matplotlib или Rviz.

4.2 Извлечение и аппроксимация препятствий

Существует множество методов для получения границ препятствий, подходящих для построения графов видимости. Классические методы компьютерного зрения извлекают контуры из бинарных изображений или карт занятости, а нейросетевые методы могут выполнять сегментацию препятствий и последующее выделение полигонов для получения требуемых вершин.

Методы обработки изображений позволяют извлекать полигоны препятствий из бинарных изображений путем анализа краевых точек и их топологического распределения для формирования замкнутых полигонов [6]. Такие бинарные изображения могут быть сгенерированы на основе карт занятости, представляющих среду в виде дискретной решетки, где ячейки могут быть помечены как препятствия или свободное пространство.

В дополнение к обработке бинарных изображений, начальный этап часто включает в себя получение этих изображений с помощью моделей сегментации, например U-Net [21], или DeepLabv3+ [27], которые формируют детальные бинарные маски, позволяющие точно выделить область каждого препятствия. Также используются другие методы предварительной обработки для точного определения препятствий. Аппроксимация полигонов уменьшает количество вершин в выделенных полигонах, сохраняя при этом их форму, за счет анализа локальной кривизны и внутренних углов между ребрами [4].

Одним из нейросетевых методов извлечения полигонов из изображений описан в [43]. PolyWorld — это нейронная сеть, предназначенная для прямого извлечения вершин зданий из изображений и соединения их в полигоны.

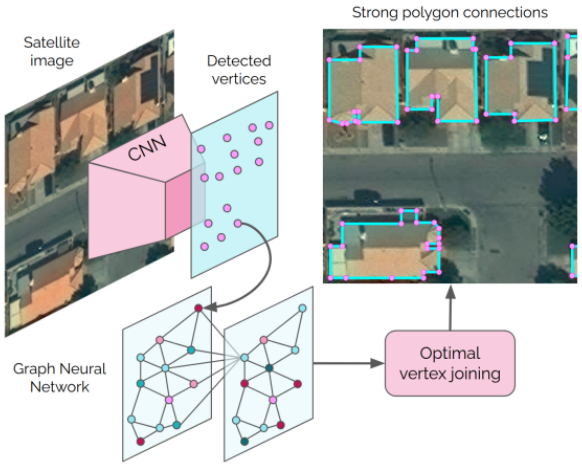


Рис. 1: Извлечение полигонов препятствий с помощью PolyWorld

PolyWorld включает в себя три ключевых компонента:

1. Vertex Detection Network: выявляет кандидатов на углы зданий на входном изображении.

2. Attentional Graph Neural Network: агрегирует информацию о вершинах и уточняет их положение с помощью графовых нейронных сетей.

3. Optimal Connection Network: формирует допустимые связи между вершинами на основе матрицы перестановок.

Архитектура PolyWorld обрабатывает входное изображение для сегментации углов зданий, затем с помощью графовой нейросети обновляет их координаты, а затем генерирует корректные связи для построения многоугольников. Этот подход позволяет одновременно получить координаты углов и матрицу соединений для построения полигонов [43].

Оценка эффективности подходов, основанных на классических методах и нейронных сетях была сосредоточена на количестве полученных вершин и скорости планирования. Кроме того, были проведены сравнения до и после векторизации операции проверки пересечений, чтобы оценить ее влияние на общую эффективность процесса.

Основной используемый подход был таким. В начале получаем бинарное изображение или карту занятости, которую преобразуем в бинарное изображение, при необходимости, применяем предварительное сглаживание (Gaussian blur). Далее приме-

няется поиск контуров в полученном бинарном образе с помощью стандартных методов компьютерного зрения (например, `findContours` в OpenCV), после чего каждый контур упрощается алгоритмом Дугласа–Пекера [4], что позволяет сократить число вершин без значительной потери геометрической точности. Также дополнительно можно выполнить фильтрацию, например, по минимальной площади и удаление избыточных вершин, которые близко расположены друг к другу. В отличие от классического подхода, нейросетевое решение (PolyWorld) может напрямую предсказывать вершины и их связи, их применение сопровождается высокой вычислительной нагрузкой на этапе использования и необходимостью переобучения под конкретную среду. Результатом этапа является набор полигональных моделей препятствий, готовых к построению графа видимости.

4.3 Построение графа видимости

Граф видимости — это графическая модель, применяемая для решения задачи планирования траектории в пространствах с многоугольниками препятствий. Процесс построения графа видимости основан на известных координатах вершин этих многоугольников. Высокая сложность построения графа видимости определяется необходимостью проверки на наличие пересечений с препятствиями всех прямых, попарно соединяющих вершины.

Среди методов планирования траекторий мобильных роботов граф видимости обеспечивает кратчайший путь, но ранее его вычислительная сложность ограничивала применение. Рост вычислительных мощностей, а также разработанные методы векторизации и аппроксимации препятствий позволяют значительно ускорить построение графа видимости.

Алгоритм построения кратчайшего пути в графе видимости сводит задачу движения сложного объекта к поиску безопасного маршрута для одной опорной точки. Сперва все препятствия представляются полигонами, после чего может быть выполнено расширение их границ на размер мобильного робота, так формируются области препятствий, в которых опорная точка не должна находиться. Далее строится граф видимости, где из полигонов получаем вершины, а ребра соединяют лишь пары точек, видимые друг другу без пересечений с полигонами. По полученному графу находится кратчайший путь, который обходит все запрещенные зоны [5].

Процесс построения графа видимости в двумерном пространстве:

1. Определяются полигоны препятствий на карте. Эти полигоны представляют области, в которых движение ограничено.

2. Используются вершины p, полученные из этих полигонов, при построении графа видимости. Эти вершины определяют граничные точки препятствий.

3. Каждая точка p в рабочей области U, включая начальную (s) и конечную (g) точки, соединяются. Это необходимо для формирования начальной структуры графа.

4. Для каждой пары точек p_1 и p_2 проверяется, пересекается ли ребро p_1p_2 с какими-либо препятствиями и не идут ли внутрь препятствий. Если нет, то ребро добавляется в граф видимости.

Этот процесс позволяет построить граф, отражающий доступные пути в среде с учетом препятствий, обеспечивая основу для последующего поиска кратчайшего пути.

Однако, чтобы построить такой граф в первую очередь нужно получить препятствия из карты, для этого рассмотрим рабочую область U, которая содержит набор препятствий $O = \{O_1, O_2, \ldots, O_n\}$. Каждое препятствие O_i представляется в виде выпуклого (или не выпуклого) многоугольника P_i, заданного набором вершин, которые в свою очередь задаются координатами на плоскости:

$$P_i = \{(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), \ldots, (x_{im}, y_{im})\}.$$

Получение препятствий заключается в следующем:

1. Получаем карту или изображение, например, из файла.

2. На полученной карте выделяем препятствия O_i используя классические, либо нейросетевые методы выделения полигонов.

3. Аппроксимируем полученные полигоны препятствий.

4. Для каждого препятствия P_i извлекается список вершин, задающих его границу.

5. Вершины представляются в виде точек $p = \{p_1, p_2, \ldots, p_m\}$, где каждая точка $p_j = (x_j, y_j)$ является вершиной многоугольника.

6. Для каждого препятствия P_i формируется набор вершин, и весь набор добавляется в список препятствий P.

Таким образом, после загрузки препятствий у нас имеется набор многоугольников $P = \{P_1, P_2, \ldots, P_n\}$, где каждый многоугольник представляет собой препятствие на плоскости.

После формирования набора полигональных моделей препятствий $P = \{P_1, P_2, \ldots, P_n\}$ следующим этапом является непосредственно построение графа видимости. Вершины будущего графа составляются из всех вершин этих полигонов. Сначала строится полный граф, в котором каждая пара вершин соединена потенциальным ребром:

$$G_{\text{full}} = (V, E_{\text{full}}), \quad E_{\text{full}} = \{(v_i, v_j) \mid v_i, v_j \in V,\ i < j\}.$$

Далее каждое ребро $e = (v_i, v_j)$ подвергается проверкам на пересечение с препятствиями и на вхождение внутрь полигона:

$$\text{Intersect}(e, P_k) = \begin{cases} \text{True}, & e \cap P_k \neq \emptyset, \\ \text{False}, & \text{иначе}, \end{cases}$$

а также, если v_i и v_j принадлежат одному препятствию P_k, применяется фильтрация по разрешенному и запрещенному диапазону построения.

В результате формируется граф видимости

$$G = (V, E), \quad E = \{(v_i, v_j) \mid \text{Intersect}((v_i, v_j), P_k) = \text{False}\ \forall k\}.$$

Таким образом, после вызова функции построения графа видимости она возвращает множество ребер видимости s, соединяющих видимые вершины. Затем добавляются начальная точка s и конечная точка g, которые задаются так:

$$s = (x_s, y_s), \quad g = (x_g, y_g).$$

Далее функция `add_points` добавляет ребра от начальной точки и конечной точки к существующему графу видимости, возвращая множество ребер $s2$. Объединяя s и $s2$ получаем расширенный целевыми точками и их ребрами граф видимости:

$$\text{visible_edges} = s \cup s2.$$

Для поиска кратчайшего пути в полученном разреженном графе $G = (V, \mathcal{E})$ применяется алгоритм Дейкстры [1]. Благодаря разреженной структуре графа Дейкстра позволяет быстро найти кратчайший путь на графе. Пусть s и g стартовая и целевая вершины в V. Результатом является последовательность вершин, образующая минимальный по длине маршрут от s к g. Алгоритм находит маршрут

$$P = (v_0, v_1, \ldots, v_k), \quad v_0 = s,\ v_k = g,$$

минимизирующий суммарный вес

$$L(P) = \sum_{i=0}^{k-1} w(v_i, v_{i+1}),$$

где

$$w(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

— евклидово расстояние между вершинами.

Таким образом, применение алгоритма Дейкстры к графу видимости позволяет эффективно находить кратчайший путь в среде с препятствиями.

4.3.1 Алгоритм построения графа видимости

Здесь приведен алгоритм построения графа видимости. Основные этапы заключаются в формировании препятствий, и их выпуклых оболочек. Затем генерации всех ребер и проверке их допустимости с учетом углов и пересечений, а затем в добавлении стартовой и целевой точек.

Алгоритм 4.1 Построение графа видимости (обработка препятствий)

ПРОЦЕДУРА `ObstacleBuild`$\mathcal{O}$
$\mathcal{E}_O \leftarrow \emptyset$
$\mathcal{V} \leftarrow \emptyset$
для всех $O_i \in \mathcal{O}$
    для $j = 1$ **to** $m_i - 1$
        $e \leftarrow (v_{i,j}, v_{i,j+1})$
        $\mathcal{E}_O \leftarrow \mathcal{E}_O \cup \{e\}$
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_{i,j}, v_{i,j+1}\}$
return $\mathcal{E}_O, \mathcal{V}$

ПРОЦЕДУРА `Convex`$\mathcal{O}$
convex_obs $\leftarrow \emptyset$
для всех $O_i \in \mathcal{O}$
    convex_obs $\leftarrow$ convex_obs $\cup$ GrahamScan(O_i)
return convex_obs

Алгоритм 4.2 Построение графа видимости

ПРОЦЕДУРА `VisGraph` $(\mathcal{V}, \mathcal{E}_O, \texttt{convex_obs})$
$\mathcal{E}_V \leftarrow \emptyset$
для всех $v \in \mathcal{V}$
  для всех $v' \in \mathcal{V} \setminus \{v\}$
    $e \leftarrow (v, v')$
    is_valid_edge $\leftarrow$ True
    для всех $O_i \in \mathcal{O}$
      если $v, v' \in O_i$ **то**
        $p_{\text{prev}} \leftarrow \text{PreviousVertex}(O_i, v)$
        $p_{\text{next}} \leftarrow \text{NextVertex}(O_i, v)$
        если $\neg\text{IsInsideAngle}(e, p_{\text{prev}}, p_{\text{next}}, \text{convex_obs})$ **то**
          is_valid_edge $\leftarrow$ False
          break
    если is_valid_edge **то**
      $\mathcal{E}_V \leftarrow \mathcal{E}_V \cup \{e\}$
$\mathcal{E}_V \leftarrow \text{IsEdgeIntersecting}(\mathcal{E}_V, \mathcal{E}_O)$
return $\mathcal{E}_V$

ПРОЦЕДУРА `AddPoints` $(p_O, p_E, \mathcal{V}, \mathcal{E}_O)$
$\mathcal{E}'_V \leftarrow \emptyset$
для всех $v \in \mathcal{V}$
  $e_O \leftarrow (p_O, v)$
  $e_E \leftarrow (p_E, v)$
  если $\neg\text{IsEdgeIntersecting}(e_O, \mathcal{E}_O)$ **то**
    $\mathcal{E}'_V \leftarrow \mathcal{E}'_V \cup \{e_O\}$
  если $\neg\text{IsEdgeIntersecting}(e_E, \mathcal{E}_O)$ **то**
    $\mathcal{E}'_V \leftarrow \mathcal{E}'_V \cup \{e_E\}$
$\mathcal{E}'_V \leftarrow \text{IsEdgeIntersecting}(\mathcal{E}'_V)$
return $\mathcal{E}'_V$

Здесь множество препятствий O состоит из полигонов O_i, каждый из которых с m_i вершинами. Препятствие задается последовательностью вершин $v_{i,1}, v_{i,2}, \ldots, v_{i,m_i}$. Процедура `ObstacleBuild` перебирает все препятствия и последовательно соединя-

ет каждую пару соседних вершин ребром, формируя в результате множества $\mathcal{E}_O$ (ребер препятствий) и $\mathcal{V}$ (вершин). Далее процедура `Convex` для каждого препятствия строит его минимальную выпуклую оболочку методом Грэхема и сохраняет ее в `convex_obs`. Основная процедура `VisGraph` создает полный граф видимости, где все пары вершин из $\mathcal{V}$ рассматриваются как кандидаты на ребро e. Также если обе вершины ребра $e = (v, v')$ принадлежат одному препятствию, то дополнительно проверяется его вхождение в запрещенный угловой сектор на основе выпуклых оболочек, используя функцию `IsInsideAngle`. После этого все отфильтрованные ребра проверяются на пересечения с $\mathcal{E}_O$ (с учетом эффективной векторизованной реализации функции `IsEdgeIntersecting` и добавляются в итоговое множество $\mathcal{E}_V$ (ребра видимости). Затем процедура `AddPoints` расширяет граф новыми ребрами от заданных точек старта p_O и финиша p_E ко всем вершинам $\mathcal{V}$, отбрасывая вновь пересекающиеся с $\mathcal{E}_O$ связи, и возвращает окончательный набор ребер $\mathcal{E}'_V$.

Таким образом, результате выполнения алгоритма формируется набор всех допустимых ребер $\mathcal{E}_V$, которые соединяют попарно все видимые вершины, при этом проверка на пересечения с препятствиями векторизованно, а затем граф расширяется за счет стартовой и целевой точек. Этот алгоритм обеспечивает быстрое построение графа видимости, на котором затем можем производить поиск кратчайшего пути.

4.3.2 Подходы к удалению лишних ребер

При построении графа видимости, ребра помимо соединения с вершинами другими препятствиями также могут соединять вершины одного препятствия, что может привести к построению ребер внутри препятствий. Для решения этой проблемы были реализованы два подхода: угловой подход и метод дополнительных ребер.

Также есть наиболее простой подход, который можно использовать, например, при условии, что все препятствия выпуклые. Он также может служить базовым решением для исключения внутренних ребер без дополнительных вычислительных затрат. Для этого при построении полного графа для каждого потенциального ребра проверяется условие — принадлежат ли обе вершины одному и тому же препятствию. При выполнении условия ребро не добавляется в итоговый граф, а во всех остальных случаях ребра включаются в дальнейший этап проверки пересечений. Однако такой подход мог приводить к потере некоторых ребер в рамках одного препятствия.

Для решения этой проблемы был разработан метод на основе анализа углов.

Угловой подход решает эту задачу, в первую очередь происходит построения минимальных выпуклых оболочек для каждого препятствия P, используя алгоритм Грэхэма [3]. Алгоритм Грэхема используется для поиска минимально выпуклой оболочки многоугольника: из множества вершин сначала выбирается опорная точка с минимальной координатой по оси y (и при равенстве – по оси x), после чего все остальные вершины сортируются по полярному углу относительно нее, а затем в стек сохраняются только те точки, которые обеспечивают поворот против часовой стрелки, удаляя внутренние выемки.

Таким образом, для каждого препятствия P_i вычисление минимальной выпуклой оболочки $\mathrm{conv}(P_i)$. Суть метода заключается в выборе опорной точки по наименьшей координате y (а при равенстве — по наименьшей x), после чего все остальные точки p_i сортируются по углу относительно этой опорной точки p_{start}:

$$\theta(p_i) = \mathrm{atan2}(y_p - y_{\text{start}}, x_p - x_{\text{start}}).$$

Отсортированные точки последовательно обрабатываются как стек: каждая новая точка сравнивается с двумя последними, так проверяем угол поворота относительно двух последних точек в массиве:

$$\mathrm{rotate}(a, b, c) = (x_b - x_a)(y_c - y_a) - (y_b - y_a)(x_c - x_a).$$

Если результат поворота указывает на вогнутый угол или коллинеарность $\mathrm{rotate} < 0$, то вершина удаляется, что обеспечивает удаление внутренних точек. Если $\mathrm{rotate} > 0$, угол выпуклый, и точка добавляется в S. Результатом алгоритма является множество точек S, которые образуют вершины выпуклой оболочки в порядке обхода против часовой стрелки [3].

Далее для каждого кандидата на ребро, соединяющего две вершины P_{start} и P_{end}, вычисляются углы между векторами от опорной вершины к соседним вершинам полигона и вектором к концу ребра. Полученные углы нормализуются в диапазон $[0, 360)$ и сравниваются с угловым сектором, определяемым соседними вершинами. Если исследуемый угол оказывается направлен внутрь полигона, соединение отвергается. В противном случае ребро сохраняется, поскольку оно не проходит внутри препятствия.

После получения выпуклых оболочек препятствий для каждого кандидата на ребро, соединяющего две вершины P_{start} и P_{end}, вычисляются углы между векторами

от опорной вершины к соседним вершинам полигона и вектором к концу ребра. Полученные углы нормализуются в диапазон $[0, 360)$ и сравниваются с угловым сектором, определяемым соседними вершинами. Если исследуемый угол оказывается направлен внутрь полигона, соединение отвергается. В противном случае ребро сохраняется, поскольку оно не проходит внутри препятствия. С этой целью вычисляются три угла: θ_1 — угол от вектора, направленного к предыдущей вершине контура, θ_3 — угол к следующей вершине, и θ_2 — угол к проверяемому концу ребра. Углы определяются стандартной функцией

$$\theta = \left(\tfrac{180}{\pi}\right) \operatorname{atan2}(\Delta y, \Delta x) \bmod 360.$$

Если опорная вершина лежит на выпуклой части оболочки, то разрешенным считается сектор, лежащий между θ_1 и θ_3 по направлению обхода против часовой стрелки; любой θ_2, выходящий за этот сектор, свидетельствует о том, что ребро заходит внутрь полигона, и такое соединение отклоняется. В случае вогнутой вершины разрешенный сектор охватывает все углы вне промежутка между θ_1 и θ_3, поскольку внутренняя область ограничивается этим вогнутым углом. Таким образом, на основании относительных положений θ_1, θ_2 и θ_3 производится удаление ребер, идущих внутрь препятствия. Данная проверка ребер проводится только в случае, когда начало и конец ребра находится в пределах одного препятствия.

Таким образом, для реализации угловой проверки каждый набор вершин препятствия хранится вместе с соответствующей выпуклой оболочкой, вычисленной заранее по алгоритму Грэхема. При обходе полного графа для текущей вершины v_c и ее кандидата v, если обе точки принадлежат одному препятствию O_i, по индексу извлекаются предыдущая и следующая вершины v_{prev} и v_{next} из списка вершин O_i и сама выпуклая оболочка $\text{conv}(O_i)$. Затем тройка $(v_{\text{prev}}, v_c, v_{\text{next}})$ и проверяемое ребро (v_c, v) передаются в функцию проверки углов, которая на основании сравнения полярных углов $\theta_1, \theta_2, \theta_3$ устанавливает, лежит ли направление ребра внутри разрешенного сектора.

Для определения типа угла в вершине v_c сначала проверяется, лежит ли она на границе выпуклой оболочки. Для этого создаем функцию, которая проверяет, лежит ли точка $p = (x, y)$ на отрезке, соединяющем точки $p1 = (x_1, y_1)$ и $p2 = (x_2, y_2)$. Это можно выразить так, что точка p лежит на отрезке $p1p2$, если она коллинеарна точкам $p1$ и $p2$. Коллинеарность проверяется через вычисление векторного произве-

дения:

$$\text{cross_product} = (y_2 - y_1) \cdot (x - x_1) - (x_2 - x_1) \cdot (y - y_1).$$

Также точка p лежит на отрезке $p1p2$, если она находится в пределах отрезка:

$$\min(x_1, x_2) \leqslant x \leqslant \max(x_1, x_2) \quad \text{и} \quad \min(y_1, y_2) \leqslant y \leqslant \max(y_1, y_2).$$

После того как для вершины установлено, выпуклый угол или вогнутый, производится фильтрация ребер. Для выпуклых точек допускаются только направления внутри вычисленного сектора, а для вогнутых — наоборот, за его пределами.

Другим подходом является метод дополнительных ребер, которые строятся параллельно ребрам полигонов препятствий с небольшим смещением внутрь (для outdoor-задач) или наружу (для indoor-задач). Пусть ребро исходного полигона задано двумя точками $P = (x_1, y_1)$ и $Q = (x_2, y_2)$. Обозначим $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$ и вычислим единичную нормаль

$$\mathbf{n} = \frac{1}{\sqrt{\Delta x^2 + \Delta y^2}} (-\Delta y,\ \Delta x).$$

Тогда новые концы дополнительного ребра получают координаты

$$P' = P \pm \delta\, \mathbf{n}, \quad Q' = Q \pm \delta\, \mathbf{n},$$

где δ — заданный сдвиг, знак <<±>> выбирается в зависимости от типа задачи (смещение внутрь для outdoor или наружу для indoor). Эти вспомогательные ребра не включаются непосредственно в граф видимости, но используются для проверки пересечений. Если какое-либо ребро основного графа пересекает вспомогательные ребра, оно считается недопустимым и исключается из графа.

Применение описанных подходов позволяет эффективно контролировать формирование корректного графа видимости, исключая лишние ребра, что улучшает надежность и точность планирования траектории.

4.3.3 Преимущества графа видимости в планировании пути

Граф видимости решает задачу планирования пути за счет построения сети допустимых маршрутов, соединяя вершины препятствий. В отличие от методов, основанных на сетке, которые делят окружение на ячейки, графы видимости соединяют все взаимно видимые вершины многоугольных препятствий, а также начальную и

конечную точки. В результате получается структура графа, в которой каждое ребро представляет собой прямую линию без столкновений.

Одним из главных преимуществ этого подхода является разреженность результирующего графа. Поскольку графы видимости состоят только из вершин препятствий и ребер видимости между ними, количество узлов значительно меньше, чем в средах представленных в виде сетки, которые могут содержать тысячи отдельных ячеек. Это приводит к более эффективному обходу графа на этапе поиска пути, например, с использованием Дейкстры.

Кроме того, граф видимости обеспечивает построение кратчайших путей. В отличие от методов, основанных на дискретной сетке, таких как A*, которые ограничены движением по ячейкам сетки и часто приводят к неэффективным траекториям, граф видимости позволяет находить оптимальные пути с точным обходом препятствий.

Другим важным преимуществом является разделение этапа построения графа и этапа поиска пути. Как только граф видимости построен, его можно многократно использовать для различных запросов. Это делает подход особенно эффективным при перепланировании, где изменяются только начальная или конечная точки. В таких случаях структура графа остается актуальной, и необходимо только пересчитать кратчайший путь, что выполняется за миллисекунды. В отличие от этого, другие методы, например Theta* и A*, вынуждены пересчитывать весь путь при каждом новом запросе.

Графы видимости, обладая разреженной структурой, высокой точностью и способностью к оперативному перепланированию, представляют собой производительное и масштабируемое решение задачи поиска пути в средах с препятствиями. В следующем разделе дается описание реализованного механизма перепланирования.

4.3.4 Механизм перепланирования

При применении мобильных роботов начальная или целевая точка могут изменяться в ходе работы, что требует оперативного пересчета траектории. Для решения этой проблемы была разработана возможность перепланирования, которая значительно сокращает время на повторное планирование в той же среде, используя уже построенный граф видимости и дополняя его только новыми начальными и конечными вершинами и соответствующие им ребра.

Планировщики на графах, такие как граф видимости или PRM, позволяют повторно использовать и обновлять существующий граф, добавляя только новые начальные или конечные вершины и инцидентные им ребра, что обеспечивает быстрое перепланирование с минимальными вычислениями. Напротив, методы, основанные на дискретной сетке (например, A*, Theta*), при каждом новом запросе вынуждены заново формировать карту и пересчитывать связность ее ячеек, что значительно увеличивает объем вычислений.

Для реализации перепланирования необходимо один раз построить базовый граф видимости на основе полигонов препятствий и сохранить его для дальнейшего использования. Этот исходный граф, определяемый набором вершин V_0 (все углы препятствий) и набором ребер E_0 (все попарные ребра видимости). Как только этот базовый граф будет построен и сохранен в памяти, его не нужно будет перестраивать до тех пор, пока геометрия препятствий остается неизменной.

При поступлении нового запроса, содержащий стартовую и целевую точку (s, g), сохраненный граф дополняется путем добавления двух новых вершин s и g, а затем генерируются только те ребра, исходящие из них, которые не пересекаются с препятствиями. Формально:

$$V = V_0 \cup \{s, g\},$$

$$E = E_0 \ \cup \ \{(s, v) \mid v \in V_0, \text{ visible}(s, v)\} \ \cup \ \{(g, v) \mid v \in V_0, \text{ visible}(g, v)\}.$$

Однократное выполнение алгоритма Дейкстры на (V, E) позволяет вычислить обновленный кратчайший путь за миллисекунды. При последующем изменении только начальной или только целевой точки та же процедура повторяется на неизменном базовом графе, что потребует минимальных затрат.

4.4 Реализация узла ROS

Планировщик на основе графа видимости реализован в виде ROS-узла. Узел подписывается на карту занятости (например, топик `/occupancy_grid_map/grid_map`), преобразует ее в бинарное изображение и отбрасывает мелкие шумовые области по порогу минимальной площади. Контуры извлекаются с помощью `cv2.findContours`, после чего упрощаются алгоритмом Дугласа–Пекера [4] для получения упрощенных полигонов препятствий. Дополнительный фильтр удаляет вершины, которые находятся ближе минимального расстояния между друг другом, что обеспечивает разреженную, но точную модель.

Координаты вершин полигонов переводятся из пикселей в метрические единицы с учетом разрешения и смещения карты. На основе преобразованных вершин единожды строится базовый граф видимости с использованием векторизованной проверки пересечений. При каждом запросе перепланирования к графу добавляются текущая позиция робота (полученная из `TF`) и пользовательская целевая точка (из топика `/move_base_simple/goal`), после чего рассчитываются видимые ребра от этих точек к уже сохраненным вершинам. Для поиска кратчайшего пути используется алгоритм Дейкстры, после чего последовательность промежуточных точек публикуется в виде сообщения `nav_msgs/Path` и отображается в RViz.

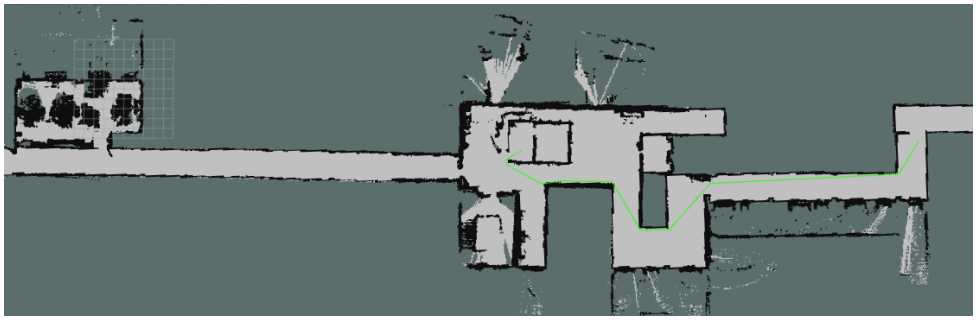


Рис. 2: Визуализация маршрута в RViz

Реализация планировщика в виде ROS-узла показала свою работоспособность и высокую производительность.

5 Методология векторизации проверки пересечений

5.1 Векторизация операций проверки пересечений

Построение графа видимости включает в себя проверку пересечений между ребрами препятствий и потенциальными ребрами графа. Этот процесс может потребовать больших вычислительных затрат из-за большого количества требуемых попарных проверок. Чтобы решить эту проблему, был разработан и внедрен подход векторизации проверок пересечений.

Для проверки пересечений используем метод Хелда [11], который основан на вычислении определителя, чтобы определить, находится ли точка слева или справа от заданного отрезка. Этот метод эффективен, так как его можно векторизовать для

одновременной проверки пересечений множества точек. Пусть требуется проверить пересечение двух отрезков AB и CD. Для этого необходимо убедиться, что концы одного отрезка находятся по разные стороны от прямой, заданной вторым отрезком, и наоборот. Для трех точек a, b и p определяется детерминант:

$$a = (a_x, a_y), \quad b = (b_x, b_y), \quad p = (p_x, p_y),$$

$$\det(a, b, p) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ p_x & p_y & 1 \end{vmatrix}.$$

Если $\det(a, b, p) > 0$, точка p лежит слева от направленного отрезка ab; при $\det < 0$ — справа, а при нулевом значении — лежит на одной прямой с отрезком.

Согласно методу Хелда, отрезки AB и CD пересекаются, когда одновременно выполняются условия

$$\det(A, B, C)\ \det(A, B, D) < 0 \quad \text{и} \quad \det(C, D, A)\ \det(C, D, B) < 0.$$

В векторизованной реализации все четыре массива определителей вычисляются параллельно для всех возможных пар ребер, создавая булевую маску, которая определяет ребра без пересечений.

Векторизованный подход предполагает представление ребер препятствий и потенциальных ребер графа в виде массивов NumPy, что обеспечивает их эффективную обработку. Вместо выполнения последовательных проверок каждой пары ребер метод использует быстрые операции над массивами, позволяющие одновременно вычислять пересечения для всех ребер.

Для формирования всех $N \times M$ комбинаций пар ребер в векторной форме применяются два приема дублирования массивов. Сначала массив ребер препятствий длины M расширяется до длины $N \times M$ путем соединения N полных копий самого себя через `np.tile`, так что каждое ребро препятствия появляется для каждого из N возможных ребер графа видимости. Затем массив возможных ребер графа видимости N расширяется до той же длины $N \times M$ путем последовательного повторения каждого элемента M раз через `np.repeat`, что гарантирует попарную проверку каждого ребра кандидата с каждым ребром препятствия. В результате образуются четыре одномерных вектора одинаковой длины $N \times M$, задающие поэлементное соответствие всех возможных пар ребер препятствий и ребер кандидатов, и обеспечивающие одновременное вычисление детерминант для проверки пересечений.

Таким образом, векторизованная процедура проверки пересечений состоит из следующих шагов:

1. Координаты ребер препятствий повторяются для каждого ребра графа с использованием `np.tile`, который дублирует массив с координатами ребер препятствий N раз, создавая одномерный массив длиной $N \times M$, где N — количество потенциальных ребер видимости, а M — это количество ребер препятствия. Это гарантирует сопоставление каждого ребра графа со всеми ребрами препятствий.

2. Аналогично, координаты потенциальных ребер графа повторяются для каждого ребра препятствия с помощью `np.repeat`, который дублирует данные о ребрах графа M раз, также получая вектор длиной $M \times N$, где M — число ребер препятствия. Это позволяет создавать все возможные комбинации ребер.

3. Далее вычисляются определители для этих пар ребер путем обработки их как отдельных векторов. Эти определители определяют относительное расположение конечных точек ребер, чтобы определить, пересекаются ли они. Все вычисления выполняются непосредственно над векторами, что позволяет избежать необходимости в явных циклах. Такой подход обеспечивает высокую производительность за счет использования векторизированных операций в NumPy, что и позволяет избегать итеративных проверок пересечений.

4. Полученный булев вектор длины $N \times M$, преобразуется размера $N \times M$, где элемент (i, j) указывает, пересекает ли ребро видимости i ребро препятствия j. Этот массив сворачивается в логическую маску длиной N, идентифицирующую видимые ребра — те, которые не пересекаются с каким-либо препятствием.

Такой векторизованный подход устраняет необходимость вложенных циклов, значительно ускоряя процесс проверки пересечений. Результатом является набор ребер, которые не пересекаются с препятствиями. Реализация на Python использует NumPy для работы с массивами и вычисления определителей, обеспечивая высокую производительность.

Ниже приведен псевдокод векторизованной операции проверки пересечений ребер:

Алгоритм 5.1 Векторизованная операция проверки пересечений

ПРОЦЕДУРА `IsEdgeIntersecting(edges)`
$N \leftarrow$ число потенциальных ребер видимости
$M \leftarrow$ число ребер препятствий
$obstacle_edges \leftarrow$ массив ребер препятствий (координаты концов)
$all_edges \leftarrow edges$ — массив всех проверяемых ребер видимости
$x_1 \leftarrow \mathrm{Tile}(obstacle_edges[:, 0],\ N)$
$y_1 \leftarrow \mathrm{Tile}(obstacle_edges[:, 1],\ N)$
$x_2 \leftarrow \mathrm{Tile}(obstacle_edges[:, 2],\ N)$
$y_2 \leftarrow \mathrm{Tile}(obstacle_edges[:, 3],\ N)$
$x_3 \leftarrow \mathrm{Repeat}(all_edges[:, 0],\ M)$
$y_3 \leftarrow \mathrm{Repeat}(all_edges[:, 1],\ M)$
$x_4 \leftarrow \mathrm{Repeat}(all_edges[:, 2],\ M)$
$y_4 \leftarrow \mathrm{Repeat}(all_edges[:, 3],\ M)$
$D_1 \leftarrow (x_1\, y_2 + x_2\, y_3 + x_3\, y_1) - (y_1\, x_2 + y_2\, x_3 + y_3\, x_1)$
$D_2 \leftarrow (x_1\, y_2 + x_2\, y_4 + x_4\, y_1) - (y_1\, x_2 + y_2\, x_4 + y_4\, x_1)$
$D_3 \leftarrow (x_3\, y_4 + x_4\, y_1 + x_1\, y_3) - (y_3\, x_4 + y_4\, x_1 + y_1\, x_3)$
$D_4 \leftarrow (x_3\, y_4 + x_4\, y_2 + x_2\, y_3) - (y_3\, x_4 + y_4\, x_2 + y_2\, x_3)$
$intersects \leftarrow \big(\mathrm{sign}(D_1) \cdot \mathrm{sign}(D_2) < 0\big) \wedge \big(\mathrm{sign}(D_3) \cdot \mathrm{sign}(D_4) < 0\big)$
$intersects \leftarrow \mathrm{Reshape}(intersects,\ (N, M))$
$visible_mask \leftarrow \neg\, \mathrm{any}(intersects,\ \mathrm{axis} = 1)$
$visible_edges \leftarrow \mathrm{Filter}(edges,\ mask = visible_mask)$
$visible_edges \leftarrow$ Фильтрация $edges$ используя $visible_edges_mask$
return $visible_edges$

Процедура `IsEdgeIntersecting` выполняет полностью векторизованную проверку пересечений ребер графа видимости с ребрами препятствий. Координаты ребер препятствий многократно дублируются командой `np.tile`, N раз дублируя каждый элемент, а координаты ребер видимости расширяются командой `np.repeat` M раз дублируя массив полностью, где N и M — соответственно числа ребер видимости и ребер препятствий. В результате образуются массивы длиной $N \times M$, в которых каждая пара ребер расположена на одной и той же позиции. Для всех таких пар параллельно вычисляются четыре детерминанта и по методу Хелда устанавливается пересекают-

ся ли они. Полученный булев вектор пересечений сворачивается в маску длины N, оставляющую только ребра, не пересекающиеся с препятствиями. Такой подход дает значительное ускорение по сравнению с последовательными алгоритмами проверки пересечений.

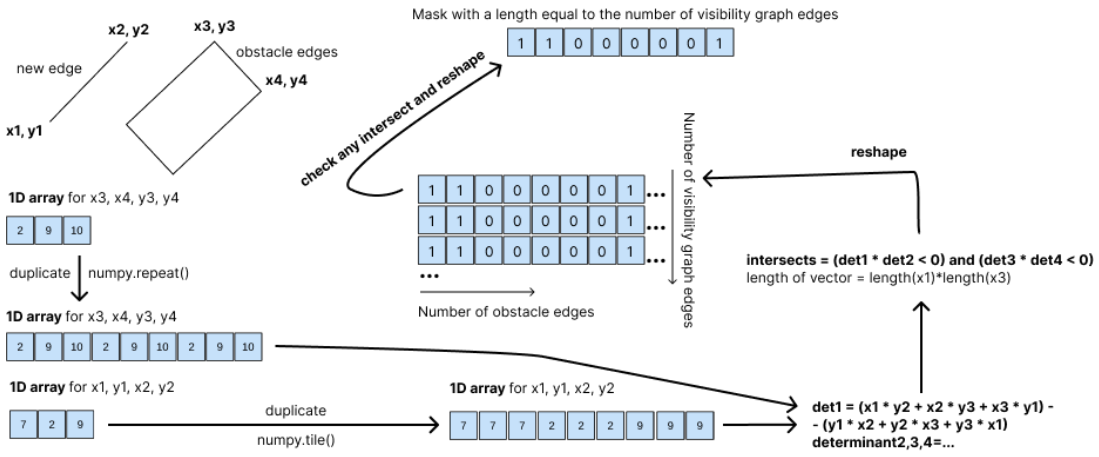


Рис. 3: Схема векторизованной проверки пересечений

5.2 Подходы реализации: AVX и NumPy

Современные процессоры поддерживают SIMD-инструкции (Single Instruction, Multiple Data), позволяющие одной инструкции одновременно обрабатывать несколько элементов данных. Это дает существенный прирост производительности по сравнению со стандартными последовательными реализациями [19]. В задаче проверки пересечений SIMD-подход позволяет одновремено проверять сразу все или несколько пар ребер вместо поочередных циклов. Такой подход особенно актуален для построения графов видимости, где число потенциальных пересечений растет квадратично с увеличением числа вершин.

Однако, для эффективного применения SIMD необходимо перестроить вычисления так, чтобы они могли выполняться параллельно по всему набору ребер. Добиться этого можно, представив координаты всех ребер в виде векторных массивов. Вместо вычисления попарных пересечений во вложенных циклах создаются массивы координат концов ребер и сохраняются в виде NumPy-массивов, либо как выровненные под AVX структуры. Помимо CPU, для подобных расчетов применяются также GPU с поддержкой CUDA или OpenCL [15], однако в настоящей работе акцент сделан на двух стратегиях, реализуемых на процессоре через NumPy и через AVX.

Первой рассмотрим реализацию на основе NumPy, в которой вычисления пересечений выполняются с использованием функций NumPy. Сама библиотека построена на оптимизированных C-интерфейсах, и многие операции выполняются с использованием SIMD-инструкций. Стандартные арифметические операции уже поддерживаются встроенными функциями AVX [24]. Таким образом, векторизованный NumPy код может обеспечить скорость выполнения, сравнимую с AVX кодом, с инструкциями, написанными вручную.

Второй подход предполагает явную векторизацию с помощью AVX, при которой данные каждого ребра загружаются прямо в регистры AVX (например, тип m256), а все операции реализуются с помощью встроенных функций AVX. Такой подход обеспечивает более точный контроль за выравниванием данных и производительностью на уровне регистров.

Реализация на основе NumPy не зависит от платформы и работает даже в средах, где инструкции AVX недоступны. NumPy хранит координаты в массивах и сам выполняет все арифметические операции и вычисления детерминантов, обеспечивая быстрые и масштабируемые векторизованные вычисления.

Реализация на основе AVX (Advanced Vector Extensions) обеспечивает ускорение за счет одновременной обработки нескольких операций с плавающей запятой с использованием архитектуры SIMD (Single Instruction, Multiple Data). Этот метод особенно эффективен при массовых проверках пересечений.

Оба метода пригодны для построения графа видимости. NumPy реализация выигрывает за счет простоты и универсальности, а AVX-вариант способен дать дополнительное ускорение на оборудовании с аппаратной поддержкой.

Advanced Vector Extensions (AVX) представляют собой набор расширений команд SIMD (Single Instruction, Multiple Data), представленных Intel для повышения производительности вычислительно тяжелых задач. AVX увеличивает ширину векторных регистров (до 256 бит в AVX и еще больше в AVX-512), что позволяет одновременно обрабатывать несколько элементов данных в рамках одной инструкции. Эта возможность позволяет ускорить выполнение арифметических операций и повышает пропускную способность, что делает ее полезной технологией для решения высокопроизводительных вычислений [17].

Как реализация на основе NumPy, так и реализация на основе AVX обеспечивают эффективную векторизацию операций проверки пересечения, что существенно ускоряет построение графа видимости. Предлагаемый подход преобразует коорди-

наты ребер в векторные представления, позволяя эффективно использовать их при реализации с использованием NumPy и AVX.

Запланированные эксперименты позволят сравнить эти подходы в различных условиях, чтобы оценить их эффективность.

6 Эксперименты и анализ результатов

В этом разделе представлена серия экспериментов, направленных на оценку эффективности нашего подхода к построению графов видимости и сравнение его с другими популярными методами планирования пути.

Для оценки предлагаемого алгоритма рассматриваются несколько показателей. Первый показатель — это вычислительная сложность, которая измеряет время, необходимое для поиска пути. Второй показатель — это качество пути, которое оценивает длину найденного пути. Наконец, оценивается масштабируемость, отражающая способность алгоритма работать со средами большего размера и с большим количеством препятствий.

6.1 Сравнение методов планирования: влияние аппроксимации и векторизации

Исследование влияния применения алгоритма аппроксимации контуров и метода векторизации проверки пересечений на производительность построения графа видимости.

Первоначальные реализации алгоритма построения графа видимости выполнялись на сгенерированных полигонах без применения упрощения полигонов и без векторизации проверки пересечений. Структуры данных в виде списков Python и вложенные циклы для попарных проверок приводили к долгим вычислениям. Например, при 13 препятствиях общее время построения графа с визуализацией могло достигать 400 секунд. В первую очередь, проведен рефакторинг неэффективных операций, например, если обнаружено хоть одно пересечение ребра с препятствием, то нет необходимости в дальнейших проверках этого ребра. Также для сокращения были заменены списки на множества и словари, что позволило сократить время на планирование более чем в два раза. Однако, это было еще довольно много.

Поэтому был проведен анализ производительности построения графа видимо-

сти, который показал, что основной проблемой является проверка пересечений между ребрами видимости и ребрами препятствиями. Основное время затрачивается именно на эту операцию. Таким образом, ускорение и сокращение количества таких проверок могло бы значительно ускорить процесс построения графа видимости.

Построение графа видимости на сгенерированных полигонах, при проведенном рефакторинге, а также дополнительно использование очередей, и исключение проверок ребер, которые строятся внутри препятствий, сократило время до 4 секунд для 13 препятствий. Поиск пути по уже построенному графу занимает около 1 миллисекунды. Поскольку скорость поиска пути высока, используется алгоритм Дейкстры. Поиск пути на сгенерированных препятствиях показан на рисунке 4.

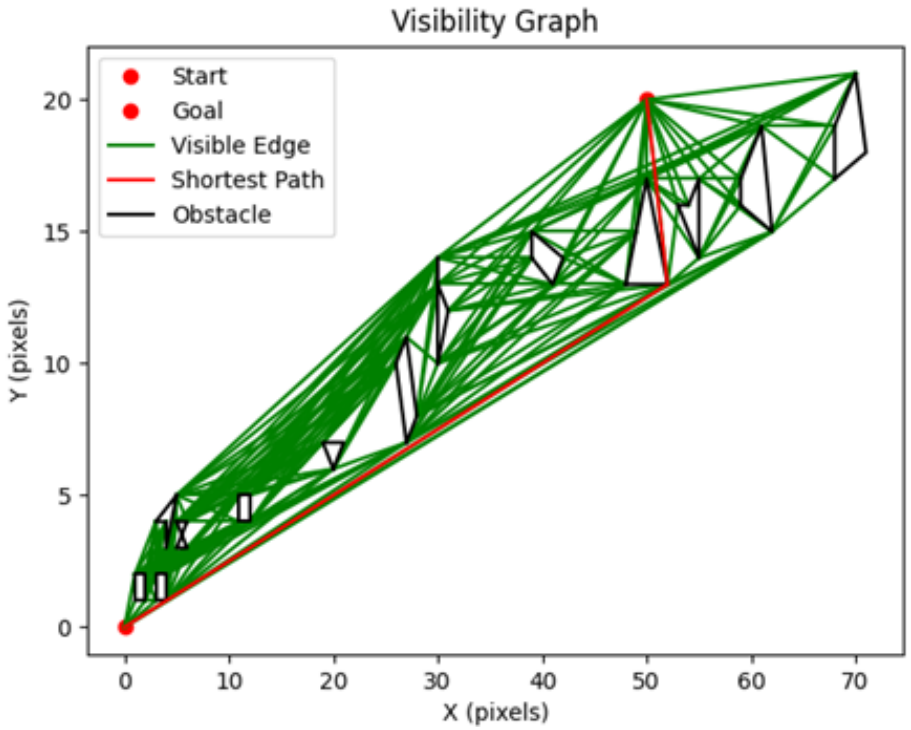


Рис. 4: Граф видимости с сгенерированными препятствиями

Далее переходим к экспериментам на реальных картах, где препятствия извлекаются напрямую из изображений, OccupancyGrid или бинаризованных изображений. Рассмотрим подходы с использованием и без использования аппроксимации полигонов. Оценим влияние частичной и полной векторизации проверки пересечений на общую производительность.

В первую очередь сравним подходы к аппроксимации, используя классические и нейросетевые подходы. После интеграции графа видимости с PolyWorld. На одном

из примеров изображение содержало 10 препятствий и 85 вершин (включая стартовую и целевую точки). Поиск пути с помощью алгоритма Дейкстры занимал около 1-1.5 миллисекунды, тогда как построение графа видимости требовало примерно 10.2 секунды. Получение информации о препятствиях занимало около 2 секунд на изображение для выделения контуров и представления их в виде списков вершин. Результат построение графа видимости с использованием PolyWorld показан на рисунке 5.

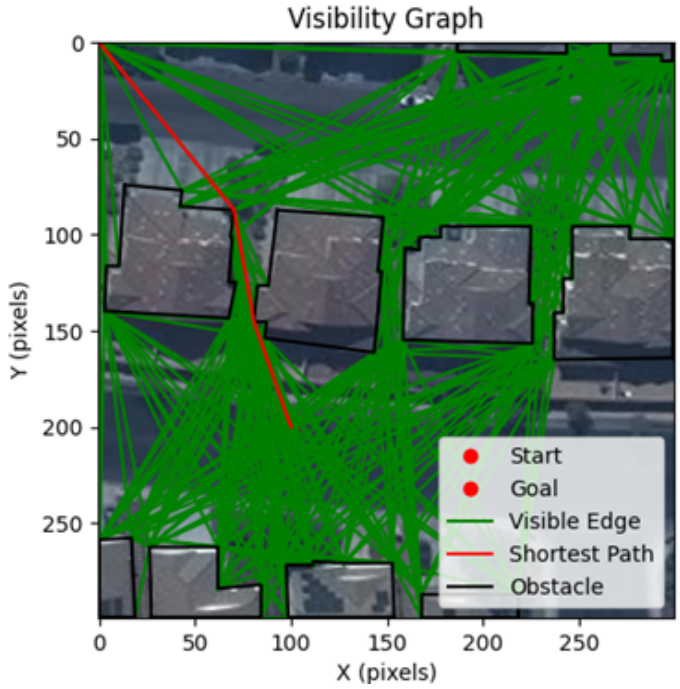


Рис. 5: Граф видимости и PolyWorld

Эти эксперименты проводились без векторизации с целью оценки различных способов извлечения полигонов и их влияния на вычислительную эффективность и точность. Затем тот же процесс был проанализирован с векторизацией для сравнения прироста производительности.

Также проведены эксперименты по извлечению полигонов с использованием классического подхода. Для извлечения вершин препятствий из бинарного изображения без аппроксимации полигонов использовалась функции `cv2.findContours()`, которая находит контуры и размещает точки вдоль них. Далее полученные вершины сохранялись и преобразовывались в формат, подходящий для построения графа видимости. В этом случае построение графа занимало примерно 170-200 секунд, в результате получался граф из 208 вершин (включая стартовую и целевую точки). Результат показан на рисунке 6.

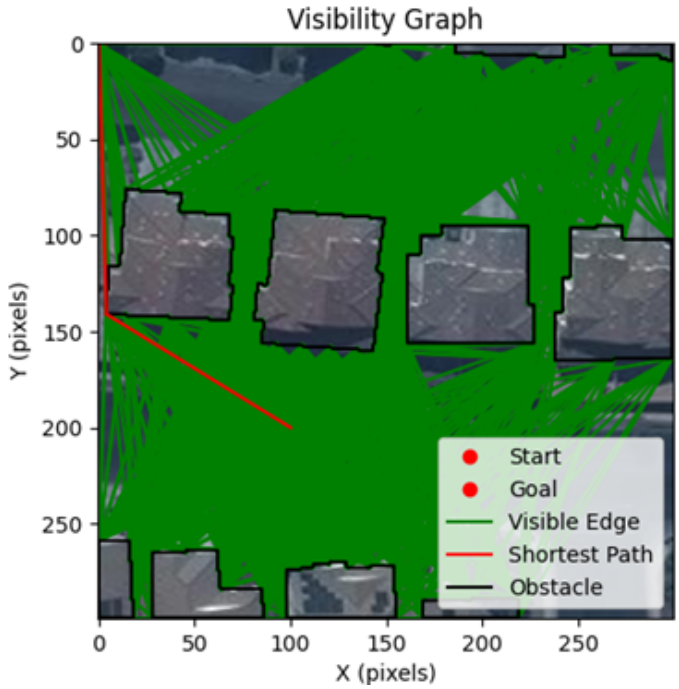


Рис. 6: Граф видимости, построенный без аппроксимаций

Используя нейросетевую реализацию построение графа видимости заняло 10.2 секунды. При этом Polyworld получал полигоны напрямую из изображения. Результат представлен на рисунке 5. При аппроксимации полигонов извлечение вершин препятствий из бинарного изображения происходило с помощью функции `cv2.findContours()`, которая выделяла контуры, а функция `cv2.approxPolyDP()` упрощала их, используя алгоритм Дугласа–Пекера. После аппроксимации построение графа видимости занимает примерно 6.4 секунды 7.

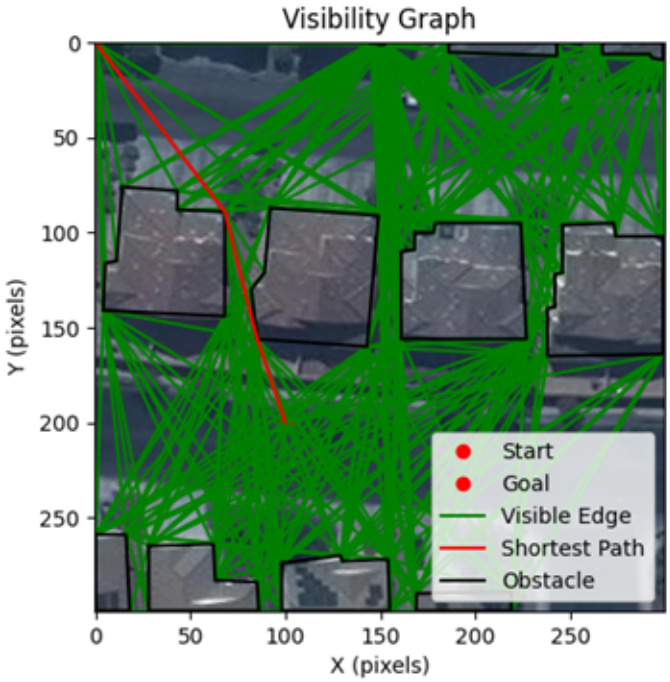


Рис. 7: Граф видимости, построенный с аппроксимацией

Цель экспериментов заключалась в сравнении различных методов извлечения полигонов с точки зрения их применимости к задачам планирования траекторий. Наиболее быстрым оказался классический подход, поскольку он превосходит нейросетевой по скорости и предоставляет более гибкие возможности аппроксимации. Сначала контуры упрощаются алгоритмом Дугласа–Пекера, затем из полученных вершин могут быть удалены избыточные точки, которые находятся близко друг к другу, а также может быть произведена фильтрация по минимальной площади полигона.

Для проверки пересечений между ребрами видимости и ребрами препятствий были проведены эксперименты с двумя подходами к векторизации. На первом этапе применялась частичная векторизация, где каждое ребро видимости сравнивалось сразу со всеми ребрами препятствий. В итоговом варианте реализована полная векторизация, при которой сразу все ребра графа проверяются одновременно со всеми ребрами препятствий по описанному выше механизму. Это позволило добиться существенного ускорения операции проверки пересечений.

Операция проверки пересечений была векторизована с использованием ранее описанного алгоритма, что позволило при сочетании этого подхода с PolyWorld получить такой же граф видимости как на рисунке 5, но сократило время построения с 10.2 секунд до 2.32 секунд для той же среды.

Далее были проведены эксперименты для сравнения различных алгоритмов планирования маршрута, включая граф видимости, Theta*, PRM и A*, в идентичных условиях. Окружающая среда была представлена в виде списка вершин препятствий, выделенных из изображения размером 300×300. Результаты эксперимента приведены в таблице 1.

Таблица 1: Сравнение времени выполнения различных методов

Эксперимент	**Метод**	**Время выполнения, с**
1	PolyWorld + Граф видимости	12.2
2	Граф видимости	170-200
3	Граф видимости + аппроксимация	6.4
Частичная векторизация		
4	VecVG-P + PolyWorld	4.8
5	VecVG-P-A	**1.53**
6	VecVG-P	36.05
Полная векторизация		
7	VecVG-F + PolyWorld	2.1
8	VecVG-F-A	**0.05**
9	VecVG-F	2
10	PRM, 65 точек, $k = 20$	0.223
11	Theta*	3.142
12	A*	6.335

Разработанный в рамках работы метод построения графа видимости с векторизацией обозначен сокращением «VecVG», к которому добавляются суфиксы, указывающие на режим векторизации и применение аппроксимации. Буква «P» (partial) означает частичную векторизацию проверок пересечений, «F» (full) — полную векторизацию, а суффикс «A» указывает на аппроксимацию полигонов препятствий. Так, запись VecVG-P-A соответствует версии с частичной векторизацией и аппроксимацией, VecVG-P — частичной векторизации без аппроксимации, VecVG-F-A — полной векторизации с аппроксимацией и VecVG-F — полной векторизации без аппроксимации. В таблице 1 также приведены результаты для исходных вариантов построения графа видимости (без указания векторизации и аппроксимации), для нейросетевой версии извлечения полигонов (PolyWorld) и для алгоритмов PRM, Theta* и A*,

причем во всех случаях поиск кратчайшего пути по графу выполнялся алгоритмом Дейкстры.

Полная векторизация значительно сокращает время построения графа видимости по сравнению с предыдущими подходами.

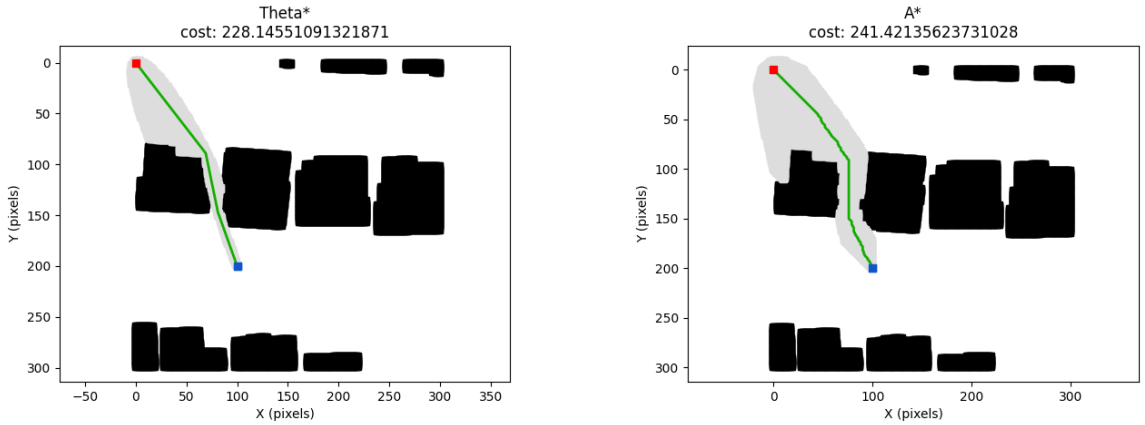
Сокращение числа вершин путем аппроксимации полигонов позволяет уменьшить время построения графа видимости с примерно 170-200 секунд до 6.4 секунд (эксперимент 2). Введение частичной векторизации проверки пересечений ребер дополнительно сокращает это время до 1.53 секунды при использовании аппроксимации (эксперимент 5). Наиболее существенный прирост дает полная векторизация всех операций проверки пересечений, где время построения графа на аппроксимированных препятствиях снижается до 0.05 секунд (эксперимент 8). Эти результаты демонстрируют, что сначала упрощение полигонов, а затем полная векторизация операций проверок пересечений является эффективным подходом для планирования пути на графе видимости.

Нейросетевой подход на базе PolyWorld показала время построения графа 12.2 секунды без векторизации (эксперимент 1), сокращенное до 4.8 секунд при частичной векторизации (эксперимент 4) и до 2.1 секунд при полной векторизации (эксперимент 7).

Кроме того, в таблице приведены данные эксперимента с Probabilistic Roadmap (PRM) с 65 случайными вершинами и $k = 20$ соседними соединениями (эксперимент 10). Здесь k обозначает число попыток соединения каждой точки с ближайшими свободными от препятствий соседями, тогда как в графе видимости каждая вершина пытается соединиться со всеми остальными. В реализации PRM алгоритм поиска соседей и проверка пересечений ребер также были ускорены за счет использования функций NumPy.

Также для сравнения приведены результаты популярных алгоритмов поиска пути: Theta (эксперимент 11) и A (эксперимент 12). Были взяты популярные реализации этих алгоритмов из наиболее популярных репозиториев. Они также были дополнительно ускорены за счет замены списков для открытых и закрытых вершин на хэш-множества. Они показывают большую вычислительную нагрузку по сравнению с методами на основе графа видимости с полной векторизацией.

Для прямого сравнения графа видимости с другими методами на рисунке 8 представлены траектории, полученные с использованием алгоритмов Theta* и A* на той же карте.

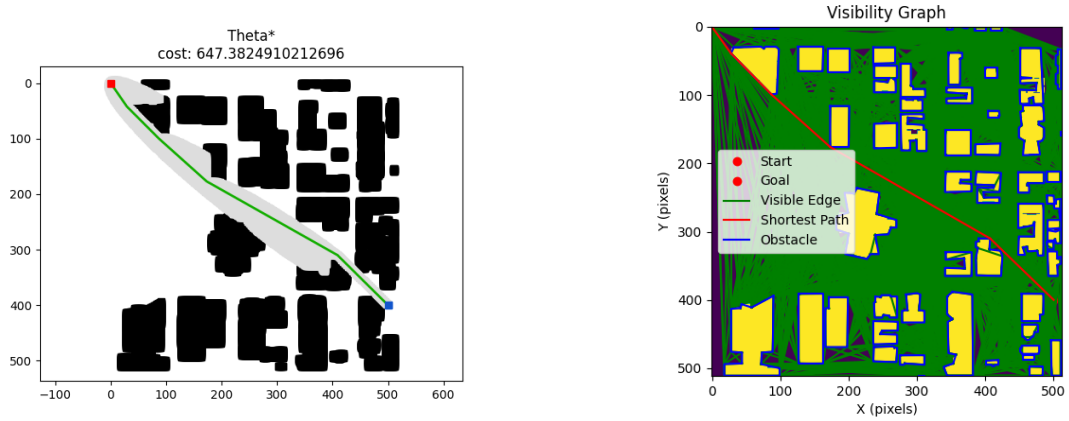


а) Путь Theta*

б) Путь A*

Рис. 8: Сравнение Theta* и A*

Также был проведен эксперимент на карте большего размера с большим количеством препятствий, чтобы оценить масштабируемость и эффективность графа видимости по сравнению с Theta*. Результаты продемонстрировали значительное преимущество графа видимости в работе со сложными средами. Алгоритму Theta* потребовалось приблизительно 20.2 секунды для построения пути. Граф видимости был построен за 4 секунды. Кроме того, после построения графа видимости сам процесс поиска пути занял всего 1 миллисекунду. Сравнение подходов на большой карте изображено на рисунке 9.



а) Theta*

б) Граф видимости

Рис. 9: Сравнение Theta* (а) и Графа видимости (б) на большой карте

Проведенные эксперименты продемонстрировали, что последовательное применение аппроксимации полигонов и векторизации операций проверки пересечений,

позволяет обеспечить наиболее быстрое планирование пути с использованием графа видимости.

6.2 Сравнение с Probabilistic Roadmaps (PRM)

В данном разделе проведена сравнение эффективности векторизованного графа видимости с методом вероятностных дорожных карт (PRM) в задачах планирования пути.

В Probabilistic Roadmaps (PRM) для построения графа выбираются случайные точки (конфигурации) в свободных областях карты, а затем происходит соединение каждой такой точки с ее k ближайшими соседями. В отличие от этого, граф видимости строится на основе вершин многоугольников препятствий и пытается соединить каждую пару вершин друг с другом, что гарантирует поиск кратчайшего пути. Для получения сходного по качеству пути с помощью PRM необходимо увеличивать и число сгенерированных точек, и параметр k, что приводит к значительному росту числа ребер и вычислительных затрат.

В экспериментах сравнивались время построения графа и качество пути. На рисунке 10 показана зависимость времени построения от относительной погрешности длины пути для обоих методов на малой и большой картах. Граф видимости всегда обеспечивает оптимальное решение (ошибка 0 %): на малой карте его построение занимает около 50 миллисекунд, а на большой — 4091 миллисекунд. В то же время параметры PRM (количество точек и k) существенно влияют и на время построения, и на точность, что видно на рисунке 10 и в таблице 2.

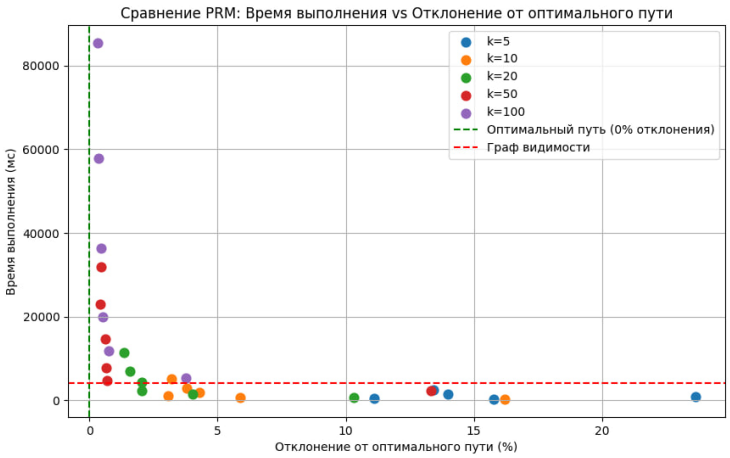
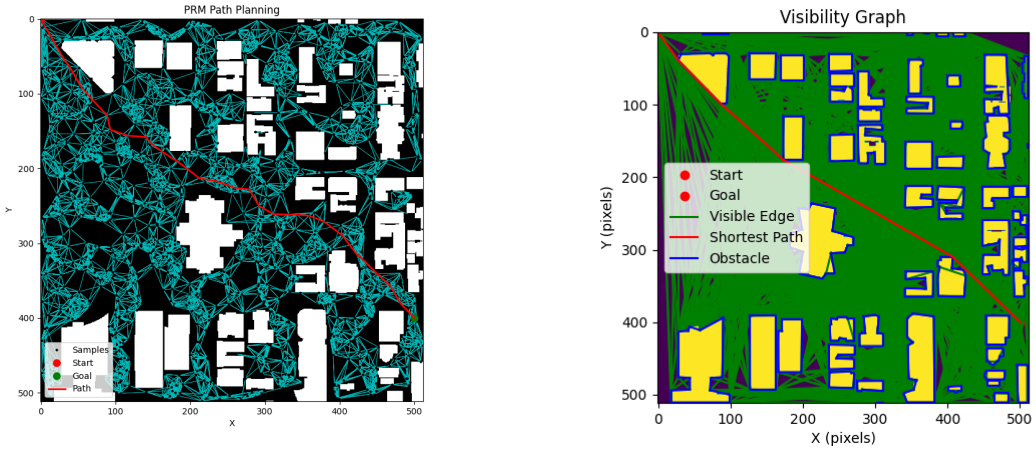


Рис. 10: Время построения и отклонение длины пути

Таблица 2: Время построения PRM и отклонения длины пути в сравнении с графом видимости

Метод	**Время (мс)**	**Длина пути**	**Соседей (k)**
Маленькая карта			
Граф видимости	50	**100%**	—
PRM (50 точек)	80	117%	10
PRM (100 точек)	112	107%	10
PRM(65 точек)	223	110%	20
Большая карта			
Граф видимости	4 091	**100%**	—
PRM (500 точек)	359	104-111%	10
PRM (1 000 точек)	1 289	102-107%	15
PRM (5 000 точек)	3 128	104-106%	10
PRM (5 000 точек)	4 574	102-103%	15

Сравнение PRM и графа видимости на большой карте показано на рисунке 11.



а) Probabilistic roadmap

б) Граф видимости

Рис. 11: Сравнение PRM (а) и графа видимости (б)

Граф видимости строится на основе вершин многоугольников препятствий и гарантирует поиск кратчайшего пути, тогда как PRM формирует граф путем случайной выборки точек в свободном пространстве и соединения каждой точки с k ближайшими соседями, что дает лишь приближенное решение. Применение аппроксимации контуров и полной векторизации проверок пересечений существенно ускоря-

ет построение графа видимости, делая его более эффективным по сравнению с PRM даже в сложных средах. Оба метода сначала конструируют граф, а затем выполняют поиск пути и позволяют быстро перепланировать маршрут. Благодаря высокой скорости за счет векторизации и упрощения полигонов, граф видимости может заменить PRM в задачах планирования пути.

Результаты экспериментов подтверждают, что для задач, требующих оптимальной длины пути, граф видимости превосходит PRM как по времени построения (на маленькой карте), так и по качеству пути (на обеих картах). PRM может достичь оптимальной длины только при очень больших выборках за счет гораздо более длительного времени построения.

6.3 Сравнение производительности NumPy и AVX

В рамках экспериментов было проведено сравнение производительности полностью векторизованных операций проверки пересечений как на Python с использованием NumPy, так и на C++ с применением AVX. На небольшой карте размером 300×300 версия на Python/NumPy завершала построение графа в среднем за 52 миллисекунды, тогда как C++/AVX требовал около 104 миллисекунд. На большой плотной карте оба подхода дали практически одинаковый результат: 4091 миллисекунда для Python/NumPy и 4119 миллисекунд для C++/AVX.

Кроме того, были проведены эксперименты для сравнения двух вариантов удаления лишних ребер. Первый подход основан на анализе углов, второй на генерации дополнительных ребер. Проведены эксперименты для оценки их производительности. Python/NumPy с дополнительными ребрами на малой карте показал время 32 миллисекунды, тогда как при угловом методе оно составило 52 миллисекунды. На большой карте угловой метод продемонстрировал 4091 миллисекунду, а метод дополнительных ребер занял 10100 миллисекунд. В C++/AVX реализация с дополнительными ребрами заняла 48-50 миллисекунд на малой карте и 7515 миллисекунд на большой, а угловой метод 104 миллисекунды и 4119 миллисекунд соответственно. Таким образом, для большинства карт предпочтителен угловой метод, но в данном случае метод с дополнительными ребрами показал результат лучше на малой карте.

Таким образом, NumPy реализация может сравниться или даже обогнать реализацию с AVX инструкциями. На малых картах преимущества NumPy проявились особенно сильно, а на больших обе реализации показали почти идентичное время

работы. Таким образом, выбор между Python/NumPy и C++/AVX в задачах планирования траекторий может базироваться на удобстве разработки и доступности инструментов, а не на заметной разнице в быстродействии.

6.4 Анализ производительности при перепланировании

В рамках экспериментов было выполнено измерение времени перерасчета маршрута при изменении стартовой или целевой точки с использованием сохраненного графа. Перепланирование на основе уже построенного графа видимости выполняется с высокой скоростью. Далее представлены результаты измерений времени перепланирования на различных картах и сравнение с подходом на основе Probabilistic Roadmaps (PRM).

Перепланирование на большой карте с большим количеством препятствий показало, что первоначальное построение графа видимости занимает примерно 3.9 секунд, тогда как последующие запросы на перепланирование выполняются всего за 34-37 миллисекунд. Для сравнения в случае с PRM время перепланирования составило 38-62 миллисекунды, при этом длина маршрутов оказалась на 1-3.4 % больше оптимальной.

В экспериментах в ROS построение графа видимости на большой карте занимало 2-3 секунды, а каждый шаг перепланирования в среднем требовал около 100 миллисекунд. Эти результаты подтверждают, что после однократного построения графа видимости все последующие перепланирования происходят довольно быстро.

7 Заключение

В рамках работы был разработан и реализован эффективный алгоритм планирования траектории на основе графа видимости с применением аппроксимации препятствий и полной векторизации проверки пересечений.

Эксперименты показали превосходство предложенного подхода над другими популярными методами. Сравнение с методами клеточной декомпозиции (A*, Theta*) и методом случайных дорожных карт (PRM) продемонстрировало более быстрое выполнение при гарантии оптимальности путей. Дополнительная интеграция методов извлечения и аппроксимации полигонов, еще больше ускоряет построение графа видимости.

После первоначального построения базового графа видимости, перепланирование для новых стартовых и целевых точек занимает лишь десятки миллисекунд. Реализации на Python/NumPy и C++/AVX, а также интеграция планировщика в виде ROS-узла, подтверждают его практическую гибкость и применимость в реальных системах.

Результаты работы опубликованы в статье Vectorized Visibility Graph Planning with Neural Polygon Extraction/ Kasmynin K., Mironov K. //International Conference on Interactive Collaborative Robotics. — Cham : Springer Nature Switzerland, 2024. — С. 265-280 [45].

Список литературы

1. *Dijkstra E.* A Note on Two Problems in Connexion with Graphs // Numerische Mathematik. — 1959. — Т. 1. — С. 269—271.
2. *Hart P. E.*, *Nilsson N. J.*, *Raphael B.* A formal basis for the heuristic determination of minimum cost paths // IEEE transactions on Systems Science and Cybernetics. — 1968. — Т. 4, № 2. — С. 100—107.
3. *Graham R. L.* An efficient algorithm for determining the convex hull of a finite planar set // Info. Proc. Lett. — 1972. — Т. 1. — С. 132—133.
4. *Douglas D. H.*, *Peucker T. K.* Algorithms for the reduction of the number of points required to represent a digitized line or its caricature // Cartographica: the international journal for geographic information and geovisualization. — 1973. — Т. 10, № 2. — С. 112—122.
5. *Lozano-Pérez T.*, *Wesley M. A.* An algorithm for planning collision-free paths among polyhedral obstacles // Communications of the ACM. — 1979. — Т. 22, № 10. — С. 560—570.
6. Topological structural analysis of digitized binary images by border following / S. Suzuki [и др.] // Computer vision, graphics, and image processing. — 1985. — Т. 30, № 1. — С. 32—46.
7. *Khatib O.* Real-time obstacle avoidance for manipulators and mobile robots // The international journal of robotics research. — 1986. — Т. 5, № 1. — С. 90—98.
8. *Rimon E.* Exact robot navigation using artificial potential functions. — Yale University, 1990.
9. *Aurenhammer F.* Voronoi diagrams—a survey of a fundamental geometric data structure // ACM computing surveys (CSUR). — 1991. — Т. 23, № 3. — С. 345—405.
10. Probabilistic roadmaps for path planning in high-dimensional configuration spaces / L. E. Kavraki [и др.] // IEEE transactions on Robotics and Automation. — 1996. — Т. 12, № 4. — С. 566—580.
11. *Held M.* Erit—a collection of efficient and reliable intersection tests // Journal of Graphics Tools. — 1997. — Т. 2, № 4. — С. 25—44.

12. Framed-quadtree path planning for mobile robots operating in sparse environments / A. Yahja [и др.] // Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146). — 1998. — С. 650—655.

13. *LaValle S.* Rapidly-exploring random trees: A new tool for path planning // Research Report 9811. — 1998.

14. Theta*: Any-angle path planning on grids / K. Daniel, A. Nash, S. Koenig, A. Felner // Journal of Artificial Intelligence Research. — 2010. — Т. 39. — С. 533—579.

15. *Fang J.*, *Varbanescu A. L.*, *Sips H.* A comprehensive performance comparison of CUDA and OpenCL // 2011 International Conference on Parallel Processing. — IEEE. 2011. — С. 216—225.

16. *Kazakov K.*, *Semenov V.*, *Zolotov V.* Topological Mapping Complex 3D Environments Using Occupancy Octrees // Графикон'2011. — 2011. — С. 111—114.

17. *Lomont C.* Introduction to intel advanced vector extensions // Intel white paper. — 2011. — Т. 23, № 23. — С. 1—21.

18. *Darintsev O. V.*, *Migranov A. B.* Comparative Analysis of Intelligent Planning Methods // Proceedings of the R.R. Mavlyutov Institute of Mechanics, Ufa Scientific Center of the Russian Academy of Sciences. — 2012. — Т. 9, № 2. — С. 53—57.

19. Performance of SSE and AVX instruction sets / H. Jeong, S. Kim, W. Lee, S.-H. Myung // arXiv preprint arXiv:1211.0820. — 2012.

20. *Yakovlev K. S.*, *Baskin E. S.* Graph Models in Trajectory Planning on the Plane // Artificial Intelligence and Decision Making. — 2013. — № 1. — С. 5—12.

21. *Ronneberger O.*, *Fischer P.*, *Brox T.* U-net: Convolutional networks for biomedical image segmentation // Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18. — Springer. 2015. — С. 234—241.

22. *Kazakov K. A.*, *Semenov V. A.* Overview of Modern Motion Planning Methods // Proceedings of the Institute for System Programming of the RAS. — 2016. — Т. 28, № 4. — С. 241—293.

23. *Заева К. А.*, *Семенов А. Б.* Метод маршрутизации с препятствиями на основе параллельных вычислений // Вестник Тверского государственного университета. Серия: Прикладная математика. — 2016. — Т. 3. — С. 85—95.

24. Accelerating Scientific Python with Intel Optimizations. / O. Pavlyk, D. Nagorny, A. Guzman-Ballen, A. A. Malakhov, H. Liu, E. Totoni, T. A. Anderson, S. Maidanov // SciPy. — 2017. — С. 106—112.

25. Mask r-cnn / K. He, G. Gkioxari, P. Dollár, R. Girshick // Proceedings of the IEEE international conference on computer vision. — 2017. — С. 2961—2969.

26. *Sadiq A. T.*, *Raheem F. A.*, *Abbas N. A. F.* Optimal trajectory planning of 2-DOF robot arm using the integration of PSO based on D* algorithm and cubic polynomial equation // First Int. Conf. Eng. Res. — 2017. — С. 458—467.

27. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation / L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam // ECCV. — 2018.

28. *Liu V.* Path Planning Methods in Obstacle Environment (Review) // Mathematics and Mathematical Modeling. — 2018. — № 1. — С. 15—58.

29. *Missura M.*, *Lee D. D.*, *Bennewitz M.* Minimal construct: Efficient shortest path finding for mobile robots in polygonal maps // 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — IEEE. 2018. — С. 7918—7923.

30. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning / A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, J. Davidson // 2018 IEEE international conference on robotics and automation (ICRA). — IEEE. 2018. — С. 5113—5120.

31. *Afanasov A. L.* Analysis of Path Planning Methods for Autonomous Mobile Devices // Issues of Science and Education. — 2019. — 17 (64). — С. 9—17.

32. *Cherroun L.*, *Boumehraz M.*, *Kouzou A.* Mobile robot path planning based on optimized fuzzy logic controllers // New Developments and Advances in Robot Control. — 2019. — С. 255—283.

33. *Филимонов А. Б.*, *Филимонов Н. Б.*, *Барашков А. А.* Вопросы построения потенциальных полей в задачах локальной навигации мобильных роботов // Автометрия. — 2019. — Т. 55, № 4. — С. 65—70.

34. Mobile robot path planning in dynamic environments through globally guided reinforcement learning / B. Wang, Z. Liu, Q. Li, A. Prorok // IEEE Robotics and Automation Letters. — 2020. — Т. 5, № 4. — С. 6932—6939.

35. Path planning and real-time collision avoidance based on the essential visibility graph / L. Blasi, E. D'Amato, M. Mattei, I. Notaro // Applied Sciences. — 2020. — Т. 10, № 16. — С. 5613.

36. A survey of path planning algorithms for mobile robots / K. Karur, N. Sharma, C. Dharmatti, J. E. Siegel // Vehicles. — 2021. — Т. 3, № 3. — С. 448—468.

37. An Intelligent Hybrid Control to Enhance Applicability of Mobile Robots in Cluttered Environments / T. Zahid, Z. Kausar, M. F. Shah, M. T. Saeed, J. Pan // IEEE Access. — 2021. — Т. 9. — С. 50151—50162. — DOI: 10.1109/ACCESS.2021.3068988.

38. FAR Planner: Fast, Attemptable Route Planner using Dynamic Visibility Update / F. Yang, C. Cao, H. Zhu, J. Oh, J. Zhang // CoRR. — 2021. — Т. abs/2110.09460. — arXiv: 2110.09460. — URL: https://arxiv.org/abs/2110.09460.

39. *Lee W.*, *Choi G.-H.*, *Kim T.-w.* Visibility graph-based path-planning algorithm with quadtree representation // Applied Ocean Research. — 2021. — Т. 117. — С. 102887.

40. *Pradhan S.*, *Mandava R. K.*, *Vundavilli P. R.* Development of path planning algorithm for biped robot using combined multi-point RRT and visibility graph // International Journal of Information Technology. — 2021. — Т. 13, № 4. — С. 1513—1519.

41. *Sanchez-Ibanez J. R.*, *Pérez-del-Pulgar C. J.*, *García-Cerezo A.* Path planning for autonomous mobile robots: A review // Sensors. — 2021. — Т. 21, № 23. — С. 7898.

42. Dynamic free-space roadmap for safe quadrotor motion planning / J. Guo, Z. Xun, S. Geng, Y. Lin, C. Xu, F. Gao // 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — IEEE. 2022. — С. 10523—10528.

43. PolyWorld: Polygonal Building Extraction with Graph Neural Networks in Satellite Images / S. Zorzi, S. Bazrafkan, S. Habenschuss, F. Fraundorfer // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. — 2022. — С. 1848—1857.

44. Research on Motion Planning Methods in Two-Dimensional Mapped Environments / V. K. Pshikhopov, M. Y. Medvedev, D. O. Brosalin, M. A. Vasilieva, B. V. Gurenko, N. Hamdan // Bulletin of the Southern Federal University. Technical Sciences. — 2022. — 3 (227). — C. 170—192.

45. *Kasmynin K.*, *Mironov K.* Vectorized Visibility Graph Planning with Neural Polygon Extraction // International Conference on Interactive Collaborative Robotics. — Springer. 2024. — C. 265—280.