Алгоритм построения графа видимости для быстрого планирования траектории мобильного робота в среде с препятствиями

Касмынин К.А.

Московский физико-технический институт
Магистерская программа «Методы и технологии искусственного интеллекта»
Центр когнитивного моделирования МФТИ

2025 – Москва

Научный руководитель:
Миронов Константин Валерьевич, PhD

Мотивация

Планирование траектории в среде с препятствиями — ключевая задача автономной робототехники.

- Мобильные роботы применяются для доставки, бытовых задач и исследований в труднодоступных местах.
- Имеет решающее значение в автономной навигации, промышленной автоматизации и беспилотных транспортных средствах.



Граф видимости, несмотря на вычислительную сложность, гарантирует нахождение кратчайшего пути. Различные методы его ускорения делают этот подход более предпочтительным по сравнению с другими методами планирования траекторий.

Проблема и цель работы

Проблема

Планирование траекторий роботов требует быстрого и точного построения маршрутов. Граф видимости является эффективным методом поиска кратчайшего пути, но его построение требует значительных вычислительных ресурсов.

Цель работы

Исследование методов генерации пути мобильных роботов. Разработка алгоритма построения графа видимости для эффективного планирования траектории мобильного робота в среде с препятствиями. Применение различных методов извлечения и аппроксимации полигонов совместно с ускоренным построением графа видимости, включая векторизацию вычислений.

Формальная постановка задачи

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.

Задача состоит из трех основных подзадач:

1. Извлечение полигонов препятствий
2. Построение графа видимости
3. Поиск кратчайшего пути

Формальная постановка задачи. Извлечение препятствий

Дано: аэрофотоснимок, спутниковое изображение или карта занятости I, на котором необходимо идентифицировать препятствия.
Цель: определить множество препятствий $O = \{O_1, O_2, ..., O_n\}$, где каждое O_i представляет собой многоугольник, описывающий контур препятствия.
Выход: набор многоугольников $P = \{P_1, P_2, ..., P_m\}$, с информацией о препятствиях для последующего построения графа видимости $G(V, E)$.

Формальная постановка задачи
Построение графа видимости

Построение графа видимости

Задача заключается в построении графа $G(V, E)$, где V — вершины многоугольников препятствий, а E — ребра, соединяющие вершины, не пересекающие препятствия.

Рабочая область: Двумерное пространство U, ограниченное:

$$x_{\min} \leq x \leq x_{\max}, \; y_{\min} \leq y \leq y_{\max}$$

Препятствия: Задано множество препятствий P в виде многоугольников.

Цель: Построить граф видимости $G(V, E)$, выявляя многоугольники препятствий P и соединяя их вершины $p_1, p_2, ..., p_n$ ребрами E, гарантируя, что ребра не пересекают препятствия.

Формальная постановка задачи
Задача поиска кратчайшего пути

Нахождение кратчайшего пути

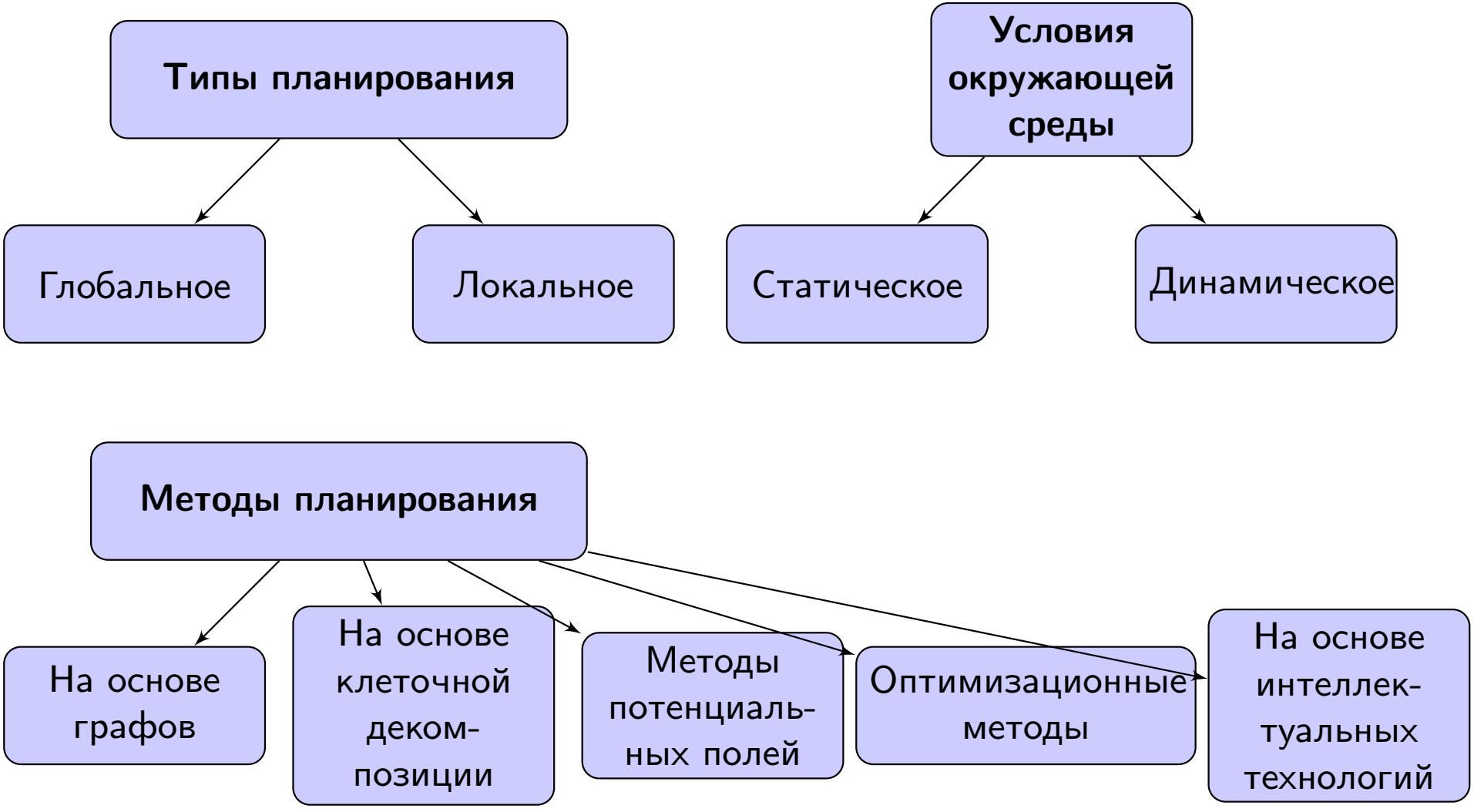
Цель: Определить кратчайший путь от начальной точки s к целевой точке g, учитывая препятствия в рабочей области U.

Шаги:

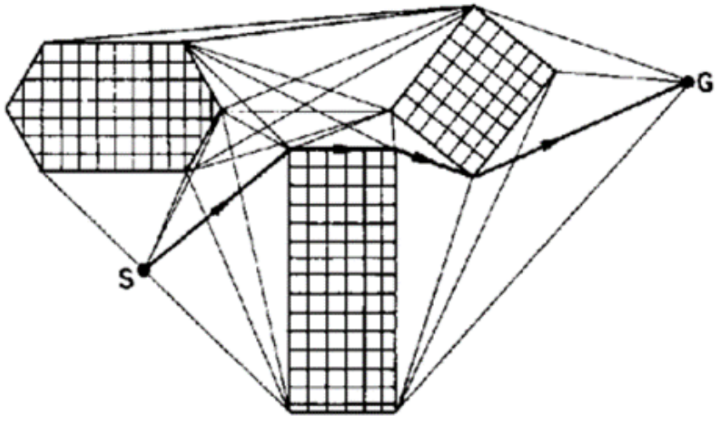
- Выделить многоугольники препятствий и их вершины.
- Построить граф видимости $G(V, E)$, где ребра представляют возможные пути, не пересекающие препятствия.
- Применить алгоритмы поиска пути (например, Дейкстры) для нахождения пути от s к g.

Результат: Найденный кратчайший путь проходит по ребрам графа видимости, соединяя s и g, при этом избегая препятствия.

Методы планирования пути

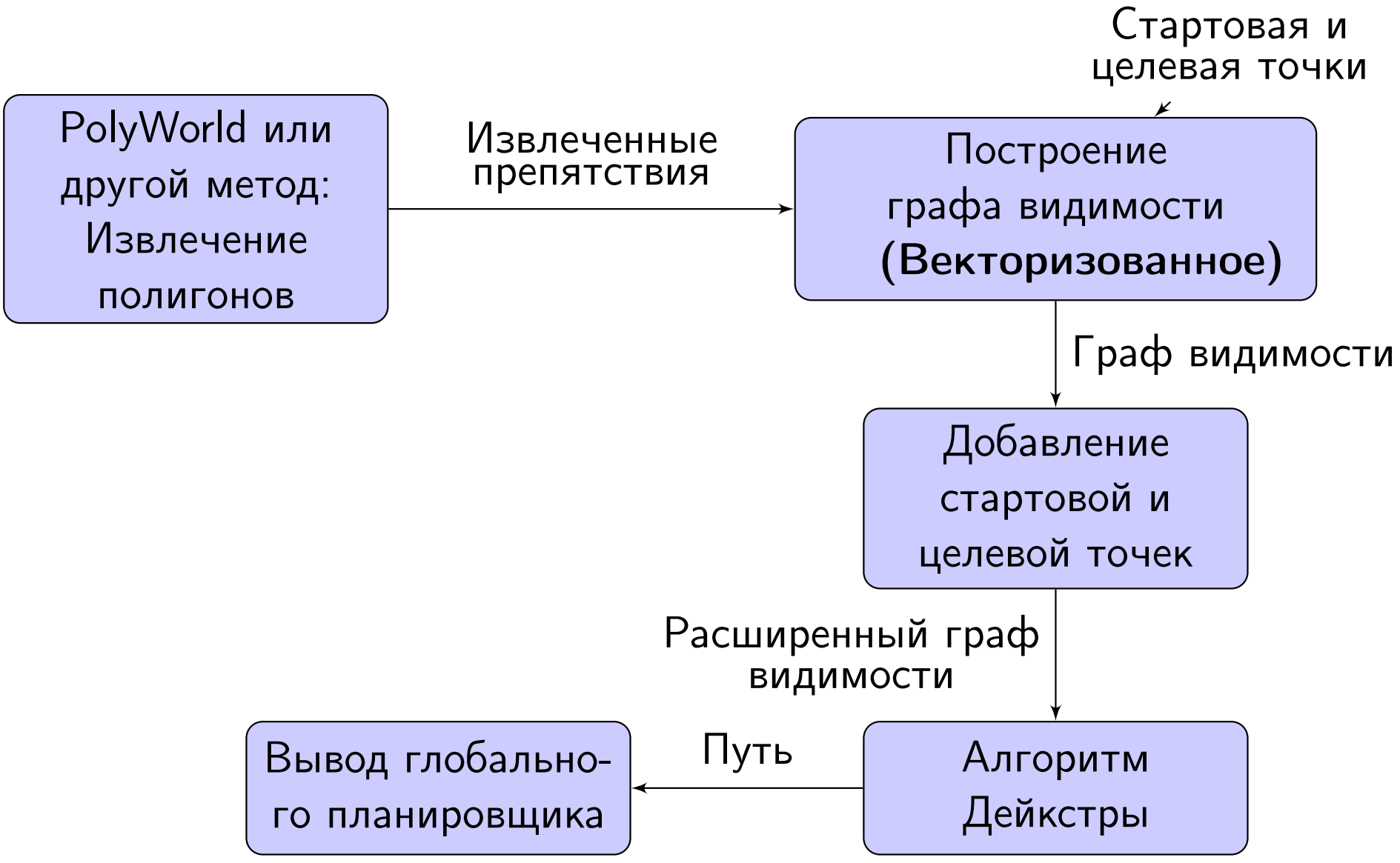


Используемый метод: Граф видимости



- **Описание метода:** Граф видимости представляет собой графовую модель, используемую для решения задачи планирования траектории в пространствах с многоугольными препятствиями.
- **Принцип работы:** Вершины графа соответствуют вершинам многоугольников и ключевым точкам (начальной и конечной), а ребра графа соединяют эти вершины, избегая пересечения с внутренними областями многоугольников.
- **Основные проблемы:** граф видимости требует много времени на построение и извлечение полигонов препятствий.

Архитектура

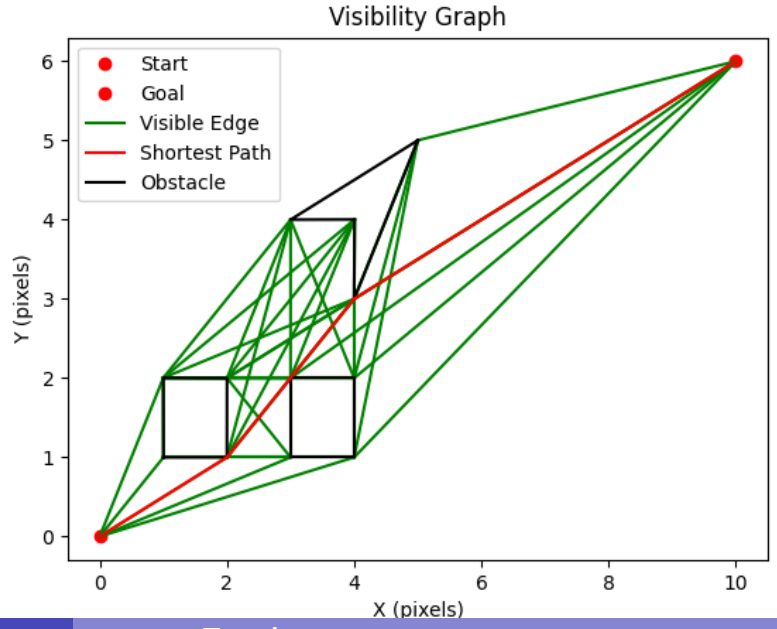


Реализация построение графа видимости

Граф видимости строится путем соединения видимых вершин препятствий и дополнительных точек (стартовой и целевой) с помощью ребер.

Этапы построения графа видимости:

1. Определить полигоны препятствий на карте.
2. Использовать вершины этих полигонов и соединить каждую вершину с другими вершинами прямыми линиями, если они не пересекаются с препятствиями и если не идут внутрь препятствий.
3. Если существует прямая видимость между двумя точками, ребро добавляется в граф видимости.



Алгоритм построения графа видимости

```
1: procedure ObstacleBuild(𝒪)
2:     ℰ_O ← ∅
3:     𝒱 ← ∅
4:     for all O_i ∈ 𝒪 do
5:         for all j = 1, 2, . . . , m_i − 1 do
6:             e ← (v_{i,j}, v_{i,j+1})
7:             ℰ_O ← ℰ_O ∪ {e}
8:             𝒱 ← 𝒱 ∪ {v_{i,j}, v_{i,j+1}}
9:     return ℰ_O, 𝒱
10: procedure Convex(𝒪)
11:     convex_obs ← ∅
12:     for all O_i ∈ 𝒪 do
13:
    convex_obs ← convex_obs ∪ GrahamScan(O_i)
14:     return convex_obs
15: procedure VisGraph(𝒱, ℰ_O, convex_obs)
16:     ℰ_V ← ∅
17:     for all v ∈ 𝒱 do
18:         for all v′ ∈ 𝒱 \ {v} do
19:             e ← (v, v′)
20:             is_valid_edge ← True
21:             for all O_i ∈ 𝒪 do
22:                 if v, v′ ∈ O_i then
23:                     p_prev ← PreviousVertex(O_i, v)
24:                     p_next ← NextVertex(O_i, v)
25:                     if
    ¬IsInsideAngle(e, p_prev, p_next, convex_obs) then
26:                         is_valid_edge ← False
27:                         break
28:             if is_valid_edge then
29:                 ℰ_V ← ℰ_V ∪ {e}
30:     ℰ_V ← IsEdgeIntersecting(ℰ_V, ℰ_O)
31:     return ℰ_V
32: procedure AddPoints(p_O, p_E, 𝒱, ℰ_O)
33:     ℰ′_V ← ∅
34:     for all v ∈ 𝒱 do
35:         e_O ← (p_O, v)
36:         e_E ← (p_E, v)
37:         if ¬IsEdgeIntersecting(e_O, ℰ_O) then
38:             ℰ′_V ← ℰ′_V ∪ {e_O}
39:         if ¬IsEdgeIntersecting(e_E, ℰ_O) then
40:             ℰ′_V ← ℰ′_V ∪ {e_E}
41:     ℰ′_V ← IsEdgeIntersecting(ℰ′_V)
42:     return ℰ′_V
```

Проверка пересечений

Для эффективной векторизации при проверке пересечений отрезков применяется метод Хелда, использующий определитель для определения положения точки относительно отрезка.

Формула:

Метод Хелда использует определитель для проверки положения точки p относительно отрезка ab. Определитель вычисляется по формуле:

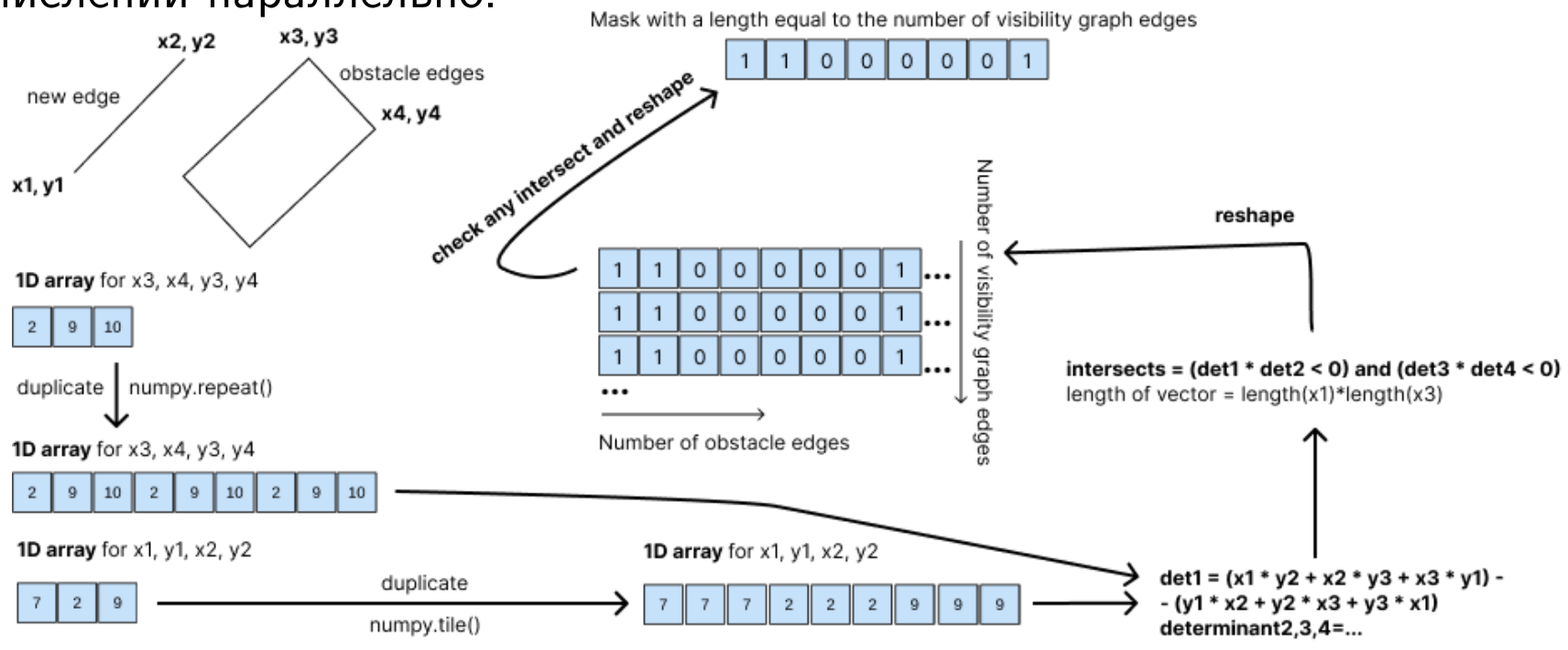
$$\det(a, b, p) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ p_x & p_y & 1 \end{vmatrix}$$

Этот определитель помогает определить относительное положение точки p по отношению к отрезку ab. Если $\det(a, b, p) > 0$, точка p находится слева от направленного отрезка от a к b; если $\det(a, b, p) < 0$, она находится справа; и если $\det(a, b, p) = 0$, точка лежит на одной прямой с отрезком.

Векторизация

Оптимизация с помощью векторизации: Построение графа видимости требует множества проверок пересечений между полигонами препятствий и потенциальными ребрами. Этот процесс может быть вычислительно затратным, особенно в сложных средах. Чтобы справиться с этой задачей, используем векторизацию.

Векторизация является ключевой особенностью в данной работе, используемой для ускорения операций. Представляя координаты препятствий в виде векторов, мы можем эффективно выполнять множество вычислений параллельно.



Алгоритм векторизованной проверки пересечений

procedure IsEdgeIntersecting(*edges*)
    $N \leftarrow$ number of *edges*
    $M \leftarrow$ number of *obstacle_edges*

obstacle_edges $\leftarrow$ array of obstacle edges
    all_edges $\leftarrow$ array of edges
    $x1 \leftarrow \text{Tile}(\textit{obstacle_edges}[:, 0], N)$
    $y1 \leftarrow \text{Tile}(\textit{obstacle_edges}[:, 1], N)$
    $x2 \leftarrow \text{Tile}(\textit{obstacle_edges}[:, 2], N)$
    $y2 \leftarrow \text{Tile}(\textit{obstacle_edges}[:, 3], N)$
    $x3 \leftarrow \text{Repeat}(\textit{all_edges}[:, 0], M)$
    $y3 \leftarrow \text{Repeat}(\textit{all_edges}[:, 1], M)$
    $x4 \leftarrow \text{Repeat}(\textit{all_edges}[:, 2], M)$
    $y4 \leftarrow \text{Repeat}(\textit{all_edges}[:, 3], M)$
    $D_1 \leftarrow (x1 \cdot y2 + x2 \cdot y3 + x3 \cdot y1) - (y1 \cdot x2 + y2 \cdot x3 + y3 \cdot x1)$
    $D_2 \leftarrow (x1 \cdot y2 + x2 \cdot y4 + x4 \cdot y1) - (y1 \cdot x2 + y2 \cdot x4 + y4 \cdot x1)$
    $D_3 \leftarrow (x3 \cdot y4 + x4 \cdot y1 + x1 \cdot y3) - (y3 \cdot x4 + y4 \cdot x1 + y1 \cdot x3)$
    $D_4 \leftarrow (x3 \cdot y4 + x4 \cdot y2 + x2 \cdot y3) - (y3 \cdot x4 + y4 \cdot x2 + y2 \cdot x3)$
    $\textit{intersects} \leftarrow (\textit{sign}(D_1) \cdot \textit{sign}(D_2) < 0) \wedge (\textit{sign}(D_3) \cdot \textit{sign}(D_4) < 0)$

$\textit{intersects} \leftarrow \text{Reshape}(\textit{intersects}, (N, M))$
    $\textit{visible_edges_mask} \leftarrow \neg\text{any}(\textit{intersects}, \text{axis} = 1)$
    visible_edges $\leftarrow$ Filter *edges* using *visible_edges_mask*
    return *visible_edges*

Методы удаления лишних ребер

Проблема:

В первых реализациях для предотвращения построения внутрь проверялось только вхождение ребра в одно препятствие, что могло приводить к потере некоторых ребер в рамках одного препятствия.

Поэтому были реализованы следующие подходы:

Метод на основе углов:

- Построение минимальных выпуклых многоугольников с использованием алгоритма Грэхема.
- Проверка выпуклых и вогнутых углов на основе построенного многоугольника.
- Применение правил, которые отсекают ребра, заходящие внутрь препятствия, но сохраняют соединения между вершинами одного препятствия.

Метод на основе дополнительных ребер:

- Генерация вспомогательных ребер, параллельных ребрам препятствий.
- Использование доп. ребер для проверки пересечений без их добавления в граф видимости.

Реализация

Поиск кратчайшего пути

Широко используемые алгоритмы: A*, Theta* и другие методы.

Граф видимости: Предоставляет альтернативный метод, соединяя вершины препятствий. Для поиска пути применяется Дейкстра.

Преимущества графа видимости:

- Граф видимости позволяет использовать уже построенный граф при перепланировании маршрута, что делает этот процесс значительно быстрее по сравнению с A* и Theta*.
- Разреженная структура графа обеспечивает меньший объем вычислений и эффективный поиск пути с помощью Дейкстры, поскольку число вершин на пути меньше, чем на карте занятости, и ребра имеют разную длину.
- Построение графа занимает основное время, но дальнейший поиск пути требует миллисекунды.

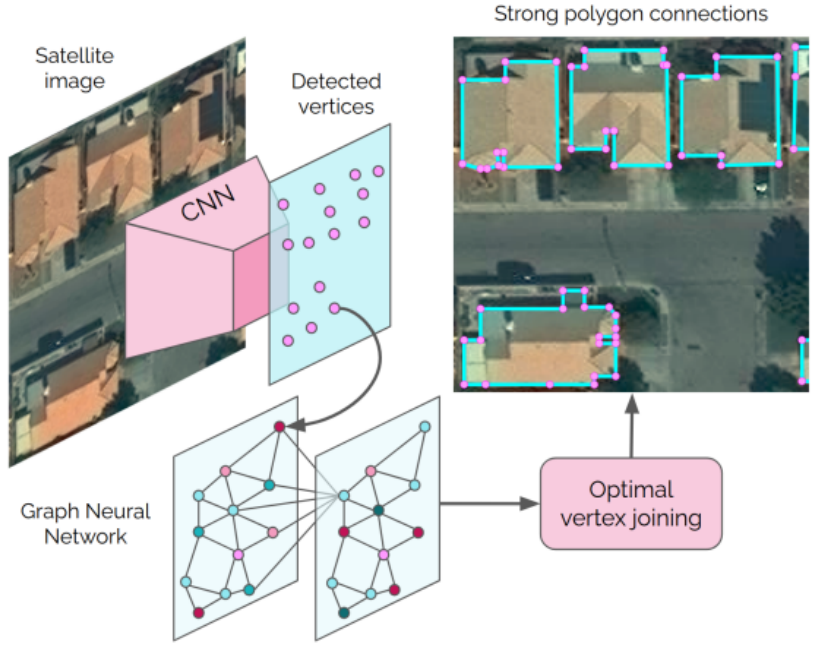
После выполнения алгоритма модуль визуализации отображает путь на графе вместе с ребрами видимости и препятствиями.

Методы извлечения полигонов

Существует два подхода к выделению полигонов препятствий: **классические** и **нейросетевые**.

Методы извлечения полигонов:

- **Классические методы:** Используют бинарные изображения для выделения контуров и аппроксимации полигонов.
- **Нейросетевые методы:** Позволяют обучать модели для выделения границ объектов. Например, **PolyWorld**.



Извлечение полигонов объектов с помощью PolyWorld

Zorzi S., Bazrafkan S., Habenschuss S., Fraundorfer F.: PolyWorld: polygonal building extraction with graph neural networks in satellite images (2022)

Реализация метода в виде ROS-узла



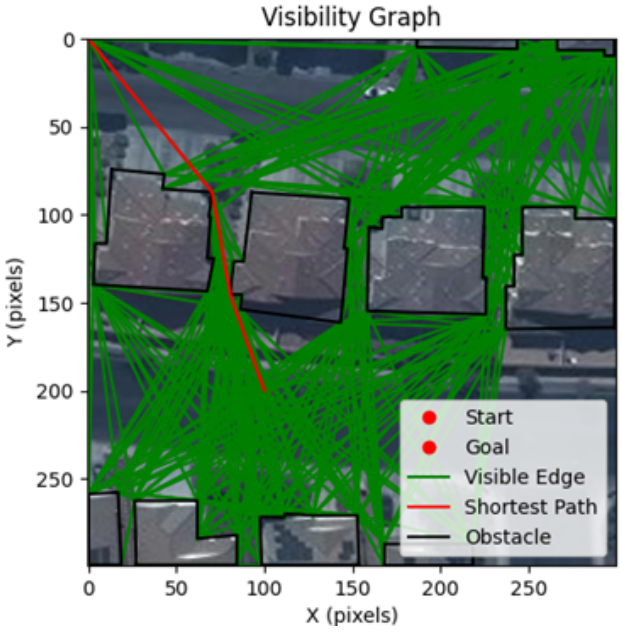
Реализован ROS-узел для планирования пути с использованием графа видимости. Входные данные:

- **Карта занятости:** из топика `/occupancy_grid_map/grid_map`.
- **Начальная позиция робота:** Получаем из `TF`.
- **Целевая точка:** Поступает из `/move_base_simple/goal`, задается в Rviz.

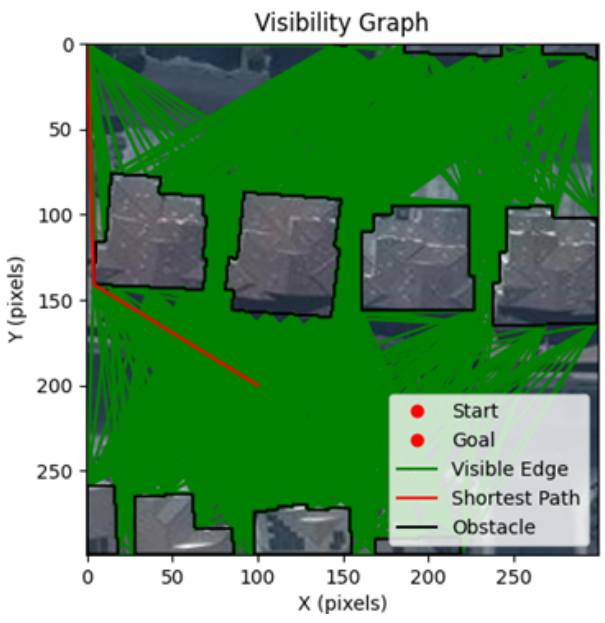
После обработки входных данных выполняется:

- Извлечение полигонов препятствий.
- Построение графа видимости и поиск пути.
- Публикация пути в топик `/path`, визуализация в Rviz.

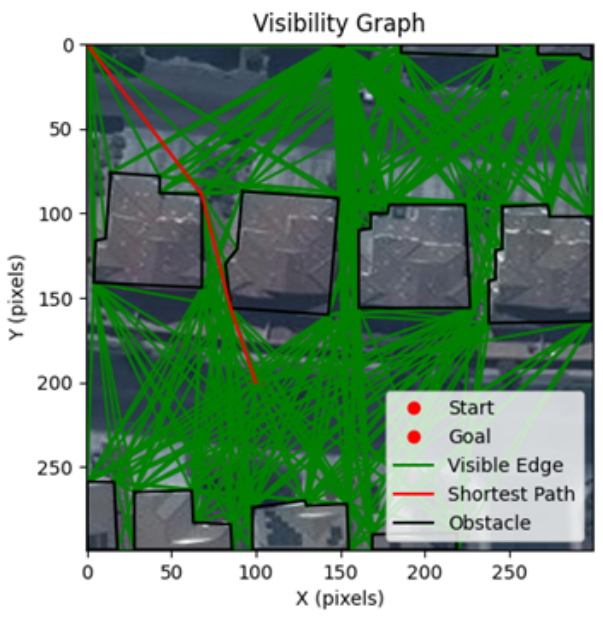
Эксперименты



Граф видимости и PolyWorld



Граф видимости (без аппроксимации)



Граф видимости (с аппроксимацией)

- Показаны изображения с наложенными ребрами и вершинами.

Эксперименты и результаты

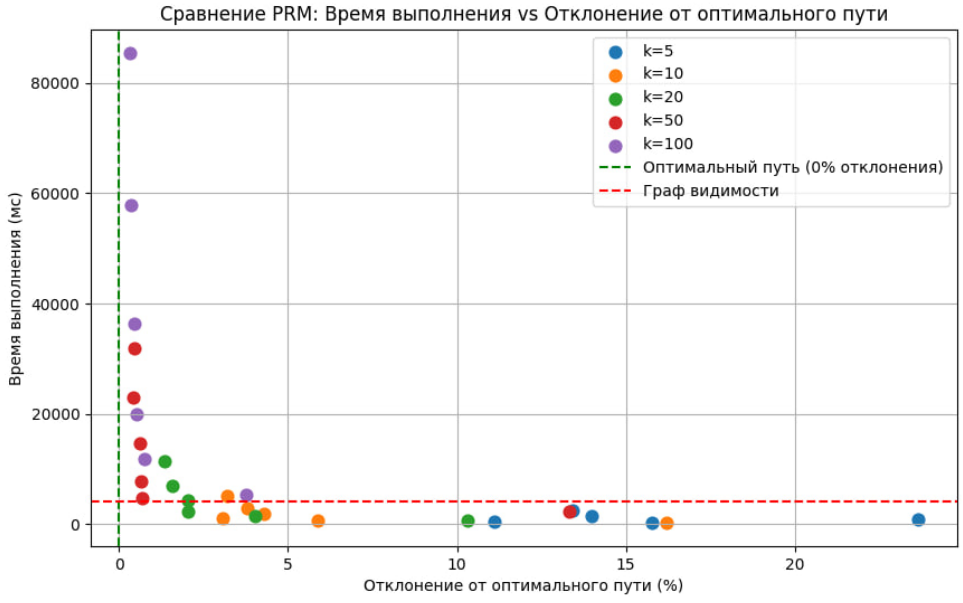
Проведены эксперименты, в которых сравнили различные алгоритмы построения путей, включая граф видимости, Theta* и A*, в одинаковых условиях. Среда была представлена как список вершин препятствий, полученных из изображения 300x300, и в виде сетки занятости для алгоритмов Theta* и A*. Результаты включают время построения среды (с извлечением полигонов) и поиска пути.

Эксперимент	Метод	Время выполнения
1	PolyWorld + Visibility Graph + Dijkstra	12.7 секунд
2	Non-Neural Network (No Approximation) + Visibility Graph + Dijkstra	170-200 секунд
3	Non-Neural Network (With Approximation) + Visibility Graph + Dijkstra	6.4 секунд
4	Partial Vectorization + PolyWorld + Visibility Graph + Dijkstra	**4.8** секунд
5	Partial Vectorization + Non-Neural Network (With Approximation) + Visibility Graph + Dijkstra	1.53 секунд
6	Partial Vectorization + Non-Neural Network (No Approximation) + Visibility Graph + Dijkstra	36.05 секунд
New Full Vectorization		
7	Full Vectorization + PolyWorld + Visibility Graph + Dijkstra	2.1 секунды
8	Full Vectorization + Non-Neural Network (With Approximation) + Visibility Graph + Dijkstra	**0.05** секунд
9	Full Vectorization + Non-Neural Network (No Approximation) + Visibility Graph + Dijkstra	2 секунды
10	Theta*	6.286 секунд
11	A*	50.1 секунд

Полная векторизация значительно сокращает время выполнения по сравнению с предыдущими подходами.

Сравнение графа видимости и PRM

- **Граф видимости** строится на основе вершин препятствий и гарантирует кратчайший путь.
- **PRM** использует случайные точки.
- **Векторизация и аппроксимация** ускоряют граф видимости, что позволяет использовать его вместо PRM.

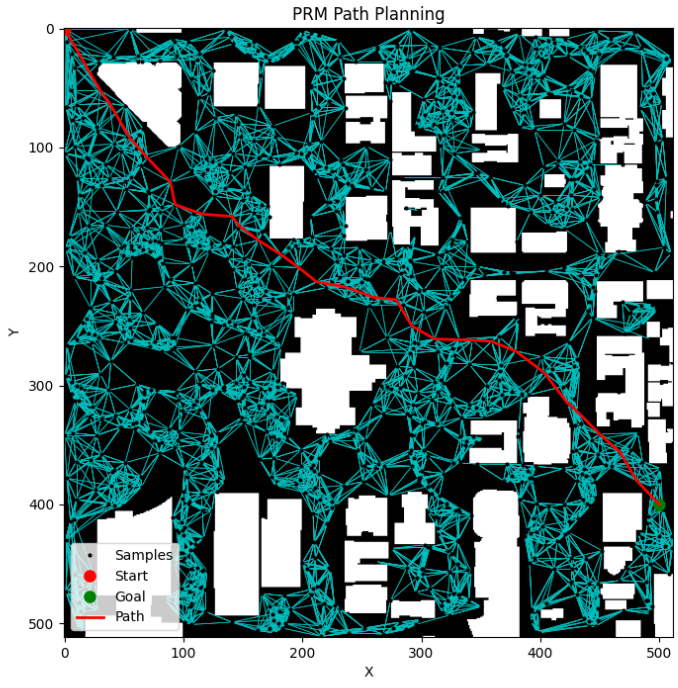


Время выполнения и отклонение

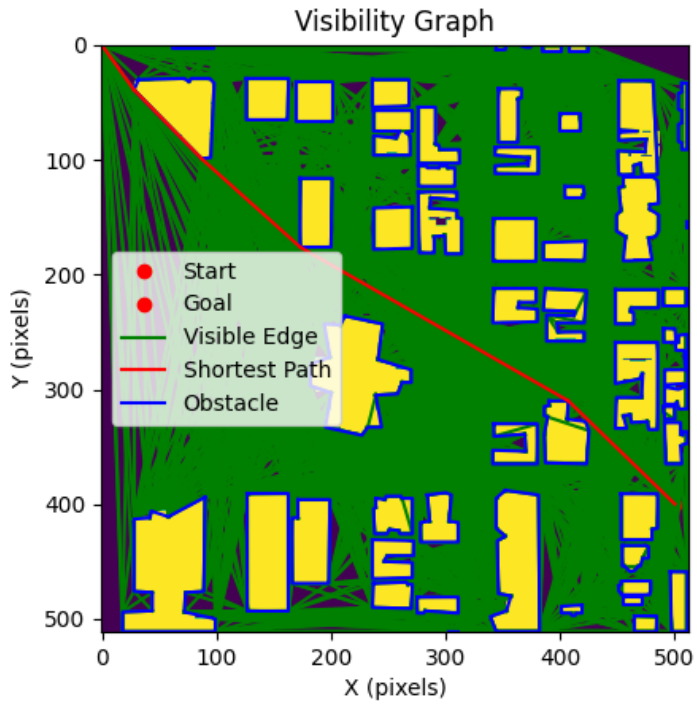
Оба метода строят граф перед поиском пути и позволяют перепланировать маршрут на его основе.

Метод	Время (мс)	Длина пути	Соседей (k)
Граф видимости	4091	**100%**	—
PRM (500 точек)	359	104–111%	10
PRM (1000 точек)	1289	102–107%	15
PRM (5000 точек)	3128	104–106%	10
PRM (5000 точек)	4574	102–103%	15

Сравнение PRM и графа видимости



PRM (1000 вершин)



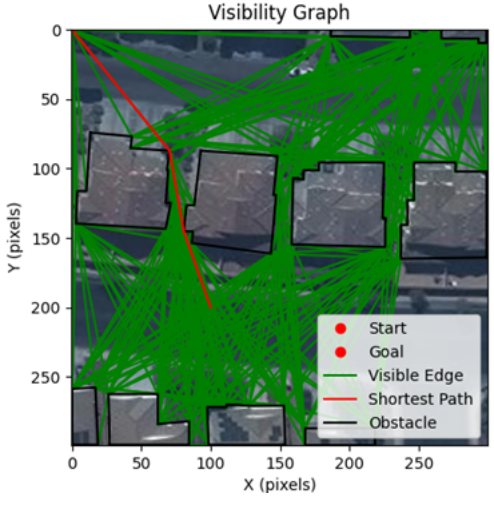
Граф видимости

Probabilistic Roadmap использует случайные точки, что может приводить к неоптимальному пути.

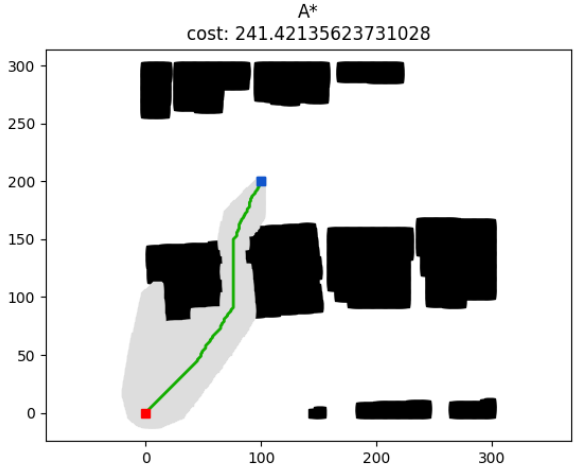
Граф видимости строится по вершинам препятствий, гарантируя кратчайший маршрут.

Эксперименты

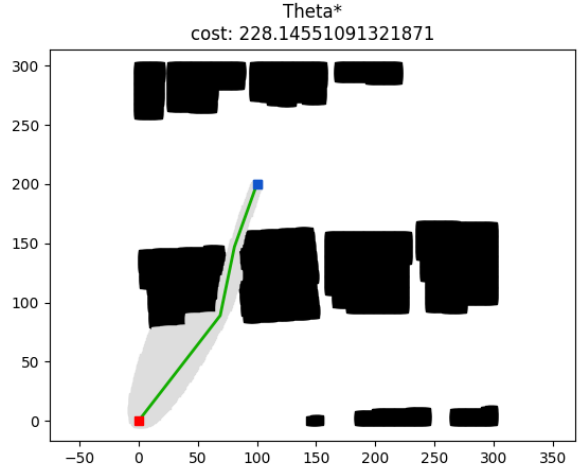
Сравнение различных алгоритмов планирования маршрута, включая Visibility Graph, A* и Theta*.



Visibility graph and PolyWorld



A*



Theta*

Сравнение Python (NumPy) и C++ (AVX)

Описание экспериментов

Проведено сравнение производительности векторизованных реализаций на Python с NumPy и C++ с AVX. Оценивалось время построения графа на малых и больших картах. В обоих случаях векторизация снижает количество итераций при проверке пересечений ребер.

Метод	**Маленькая карта** (мс)	**Большая карта** (мс)
Python + Numpy (угловой метод)	52	**4091**
Python + Numpy (доп. ребра)	**32**	10100
C++ + AVX (доп. ребра)	48-50	7515
C++ + AVX (угловой метод)	104	4119

Таблица: Сравнение времени работы различных методов

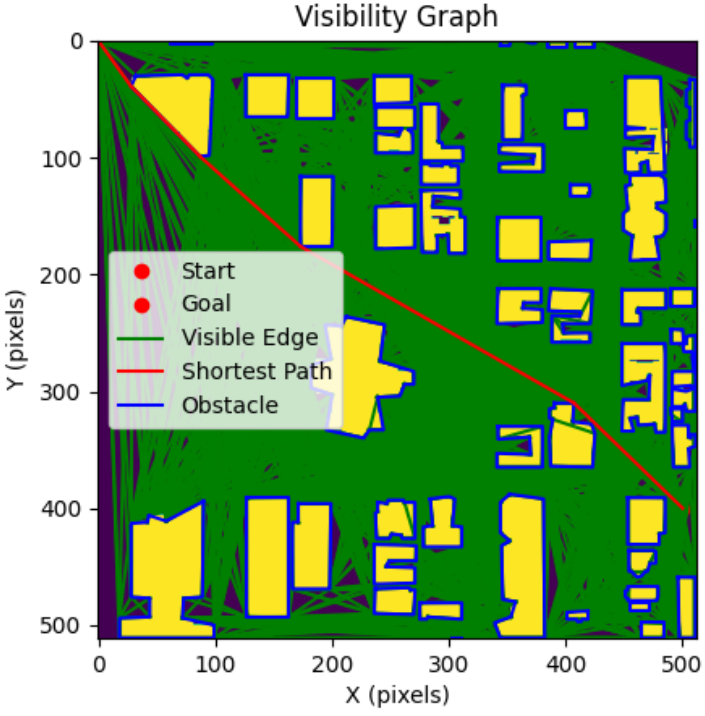
Выводы

- На малых картах Python (NumPy) превосходит C++ (AVX).
- На больших картах при угловом методе быстрее работает Python (NumPy), а при дополнительных ребрах быстрее реализация с AVX инструкциями.

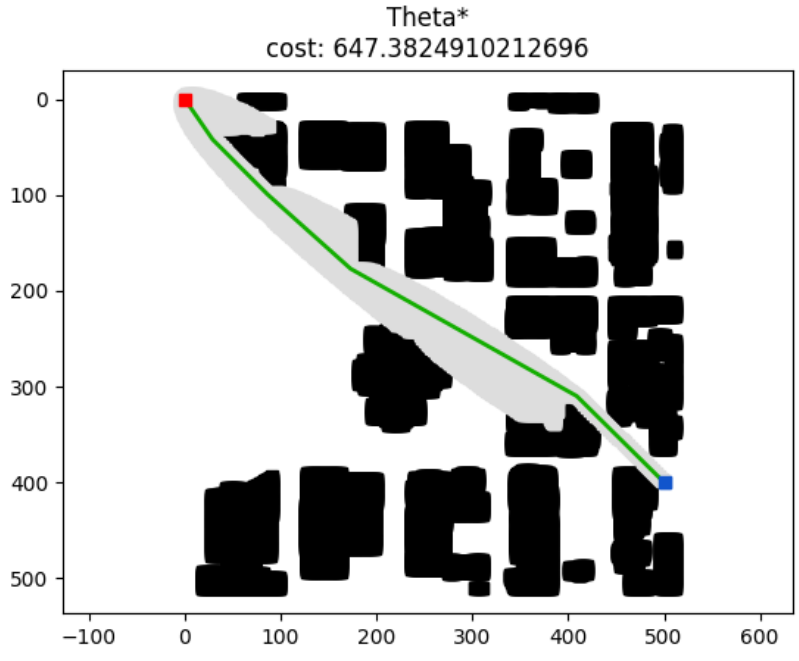
Эксперименты

Эксперименты на большой, загруженной препятствиями карте.

- Theta* занял около 180 секунд для построения пути.
- Visibility graph занял 4 секунды, включая построение графа.
- Поиск пути на графе видимости занял всего 1 миллисекунду.



Visibility graph



Theta*

Заключение

- Разработан метод построения графа видимости с полной векторизацией операций проверки пересечений, что значительно сократило время выполнения алгоритма.
- Эксперименты показали превосходство предложенного подхода над другими популярными методами (Theta*, A*) по времени построения пути при одинаковых условиях.
- Интеграция нейросетевых методов (PolyWorld) или аппроксимации полигонов ускоряет построение графа видимости.
- Опубликована статья: *Vectorized Visibility Graph Planning with Neural Polygon Extraction (ICR 2024)*
- **Планы на будущее:**
  - Реализовать механизм перепланирования маршрута
  - Зафиксировать полученные результаты в статье и дипломной работе

Контакты

- Касмынин Кирилл Электронная почта: kirill.kas@mail.ru