Исследование методов генерации траекторий мобильного робота в среде с препятствиями

Касмынин К.А.

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

2024 – Москва

Научный руководитель:
Старший научный сотрудник, PhD
Миронов Константин Валерьевич

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.



Методы генерации траекторий необходимы для обеспечения безопасности, эффективности и точности движения роботов.

Проблема и цель работы

Проблема

Проблема планирования маршрута в средах с препятствиями представляет собой актуальное и сложное исследовательское направление. Алгоритмы должны генерировать безопасные пути без столкновений с препятствиями. Они также должны снижать вычислительные затраты при построении траекторий.

Цель работы

Исследование и разработка методов генерации безопасных маршрутов мобильных роботов. Обеспечение эффективного и точного движения роботов при автономной навигации и выполнении различных задач. Применение различных методов извлечения полигонов совместно с ускоренным построением графа видимости, включая векторизацию вычислений и использование других эффективных подходов для сокращения вычислительных затрат его построения.

Формальная постановка задачи

Общее описание задачи

Задача состоит из трех основных подзадач:

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

Формальная постановка задачи

Дано: аэрофотоснимок, спутниковое изображение или occupancy grid 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 на изображении.
Выход: набор многоугольников $P = \{P_1, P_2, ..., P_m\}$ и граф видимости $G(V, E)$, где V - вершины, соответствующие вершинам многоугольников P, а E - ребра, соединяющие вершины, между которыми возможно движение.

МФТИ

Построение графа видимости

Задача заключается в поиске кратчайшего пути на плоскости с использованием графа видимости.

Рабочая область: Область U определена в двумерном пространстве L, ограниченном:

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

Препятствия: Задано множество препятствий, каждое из которых представлено в виде многоугольника.

Цель: Построить граф видимости, выявляя многоугольники препятствий и соединяя их вершины $p_1, p_2, ..., p_n$ ребрами, при этом гарантируя, что ребра не пересекают препятствия.

Формальная постановка задачи
Задача поиска кратчайшего пути

Нахождение кратчайшего пути

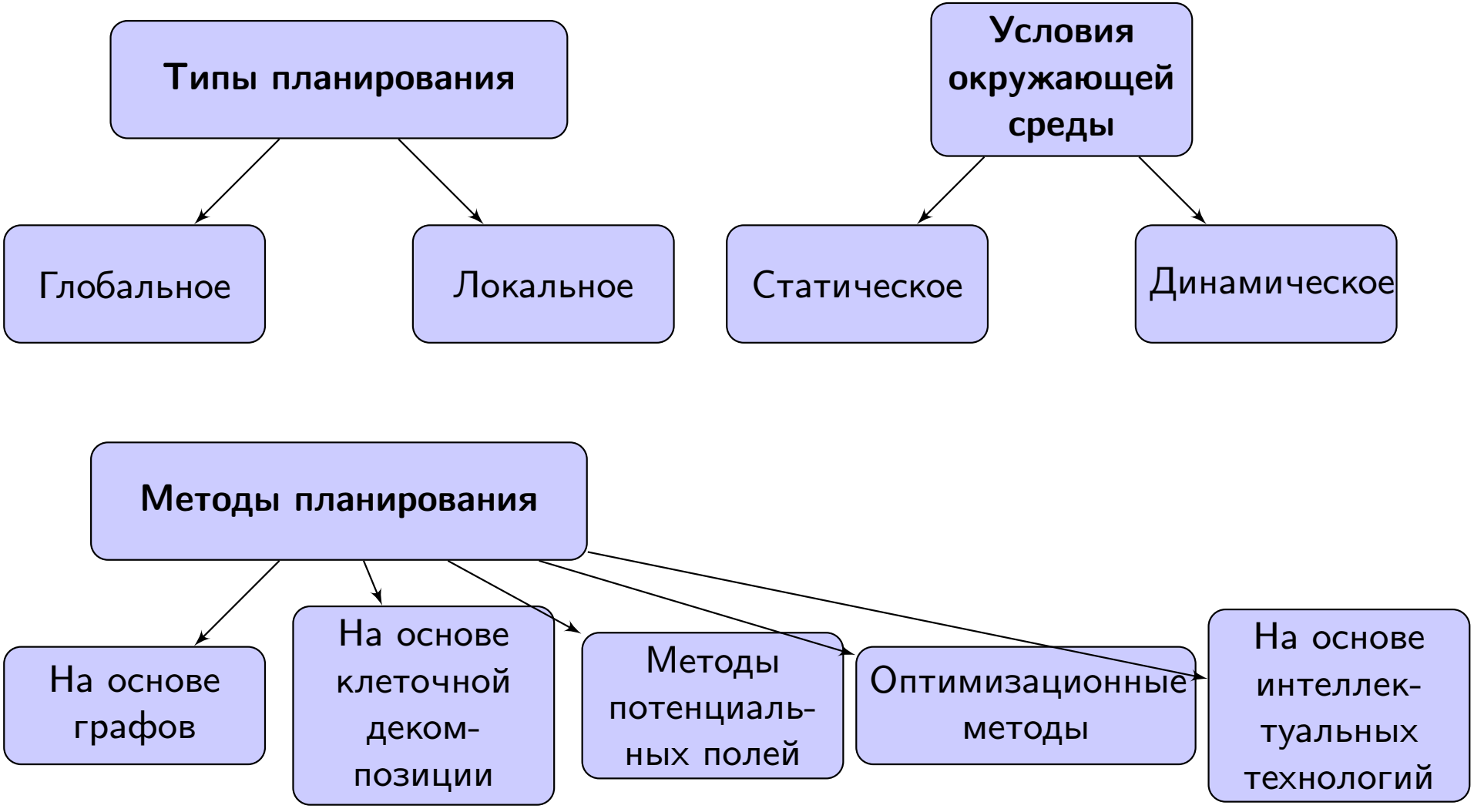
Цель: Определить кратчайший путь от начальной точки s к целевой точке g, учитывая препятствия в рабочей области U.

Шаги:

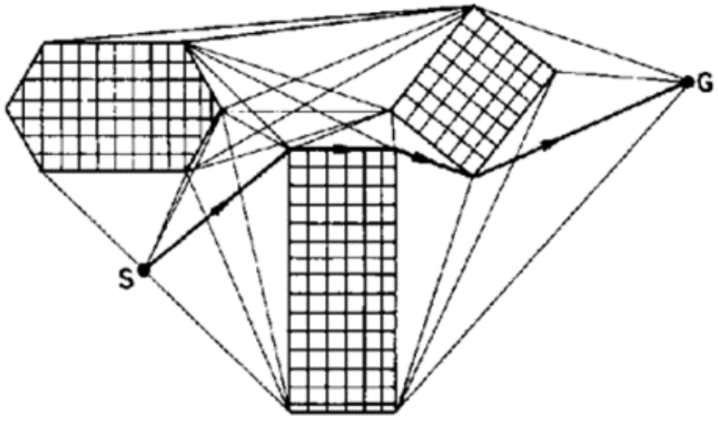
- Выделить многоугольники препятствий и их вершины.
- Построить граф видимости $G(V, E)$, где ребра представляют возможные пути, не пересекающие препятствия.
- Применить алгоритмы поиска кратчайшего пути (например, алгоритм Дейкстры) для нахождения пути от s к g по ребрам графа видимости.

Результат: Найденный путь проходит по ребрам графа видимости, соединяя s и g, и обеспечивает эффективный маршрут, избегая препятствия.

Методы планирования пути



Используемый метод: Граф видимости



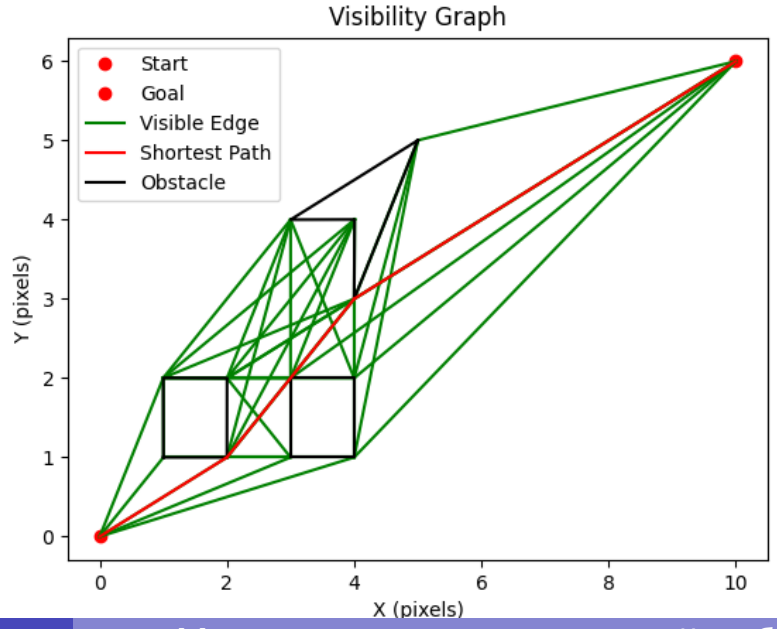
- **Описание метода:** Граф видимости представляет собой графовую модель, используемую для решения задачи планирования траектории в пространствах с многоугольными препятствиями.
- **Принцип работы:** Вершины графа соответствуют вершинам многоугольников и ключевым точкам (начальной и конечной), а ребра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

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.



Visibility Graph Algorithm

```
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:

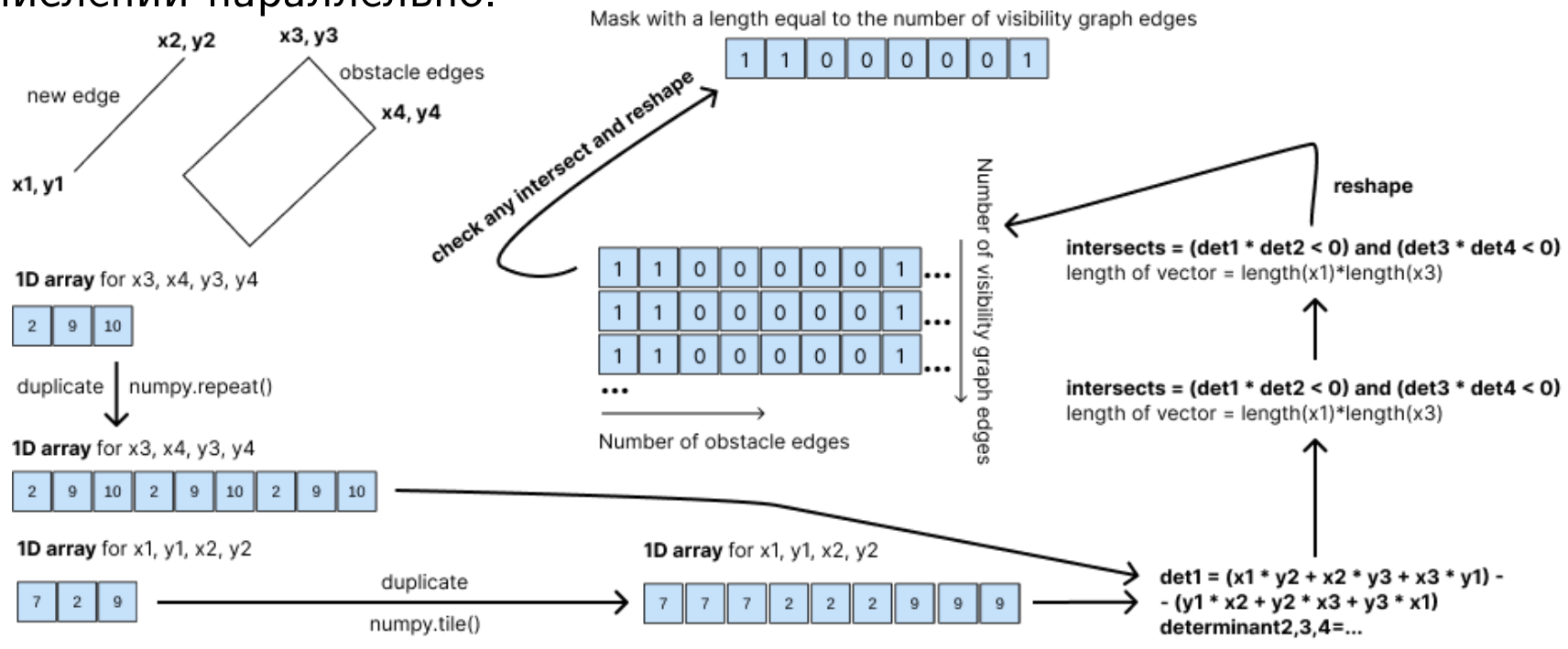
$$\det(a, b, p) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ p_x & p_y & 1 \end{vmatrix}$$

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.



Vectorized Edge Intersection Check Algorithm

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

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

Работа с выпуклыми и вогнутыми углами

Проблема:

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

Сейчас построение полного графа производится на основе углов.

Реализовано для решения:

- Построение минимальных выпуклых многоугольников с использованием алгоритма Грэхэма.
- Проверка выпуклых и вогнутых углов на основе построенного многоугольника.
- Применение правил, которые отсекают ребра, заходящие внутрь препятствия.
- Возможность построения ребер между вершинами одного препятствия, не заходя внутрь него.

Примечание: Хотя метод работает эффективно для внешних препятствий, возможны сложности при работе в "indoor"условиях

Реализация

Поиск кратчайшего пути

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

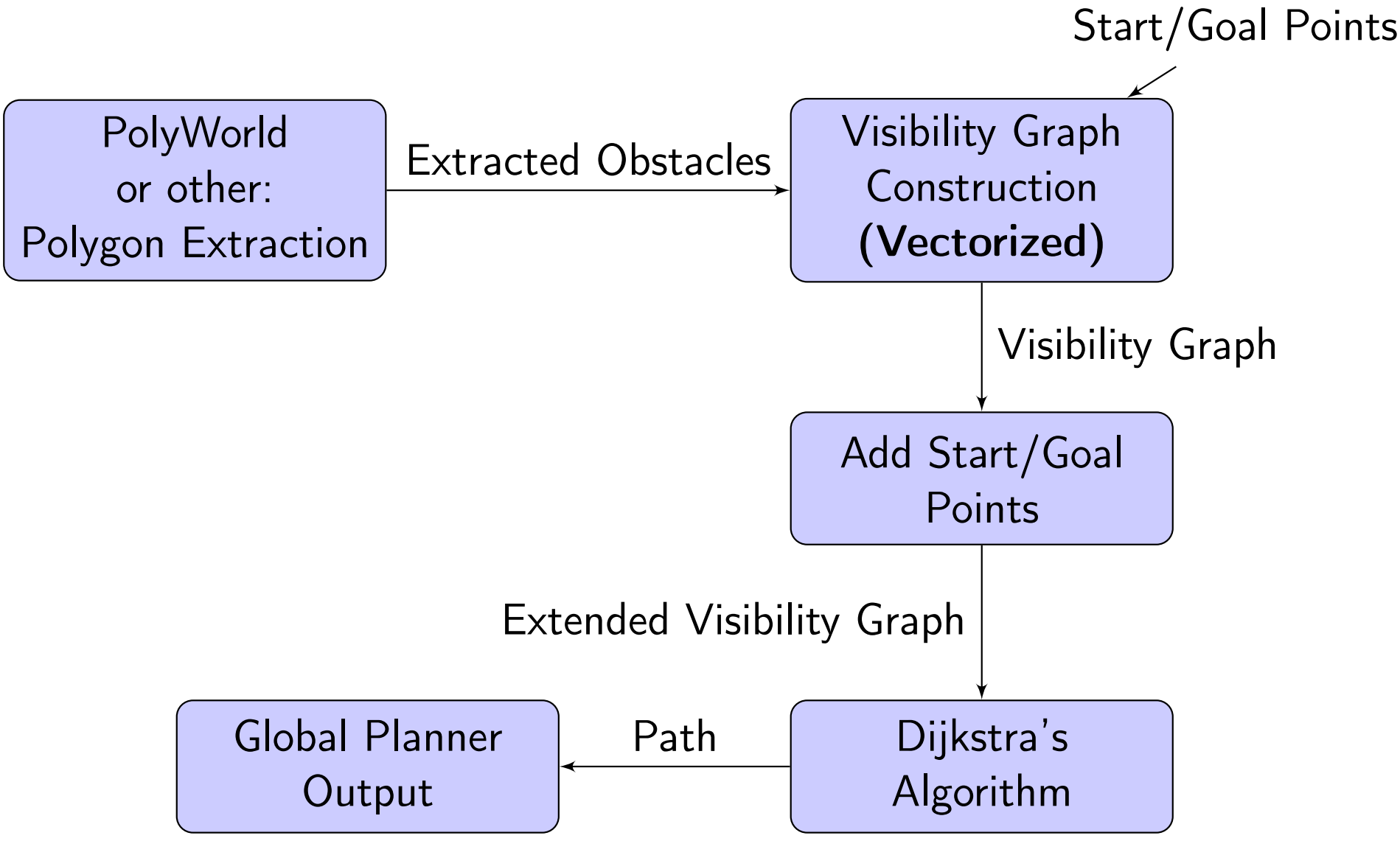
Граф видимости: Предоставляет альтернативный метод, соединяя вершины препятствий. Для поиска пути применяется Дейкстра.

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.

Архитектура



Методы извлечения полигонов и PolyWorld

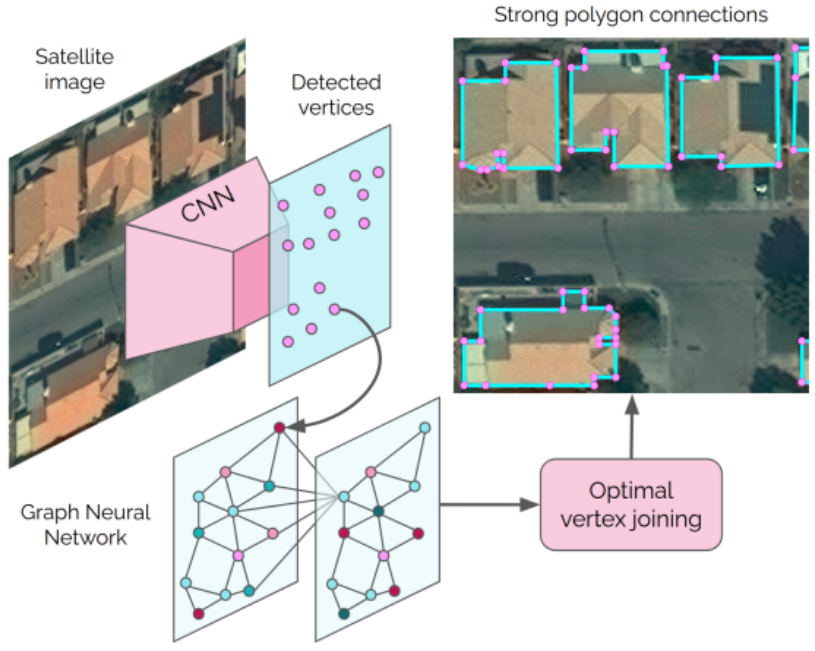
PolyWorld обеспечивает эффективное и точное обнаружение препятствий.

Методы извлечения полигонов:

- **Классические методы:** Используют бинарные изображения для выделения контуров и аппроксимации полигонов.
- **Нейросетевые методы:** Например, **PolyWorld**.

Архитектура PolyWorld:

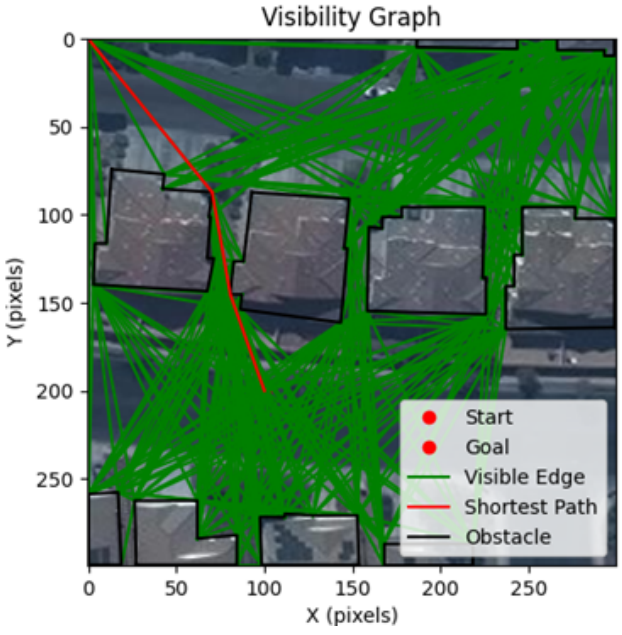
- **Vertex Detection Network**
- **Attentional Graph Neural Network**
- **Optimal Connection Network**



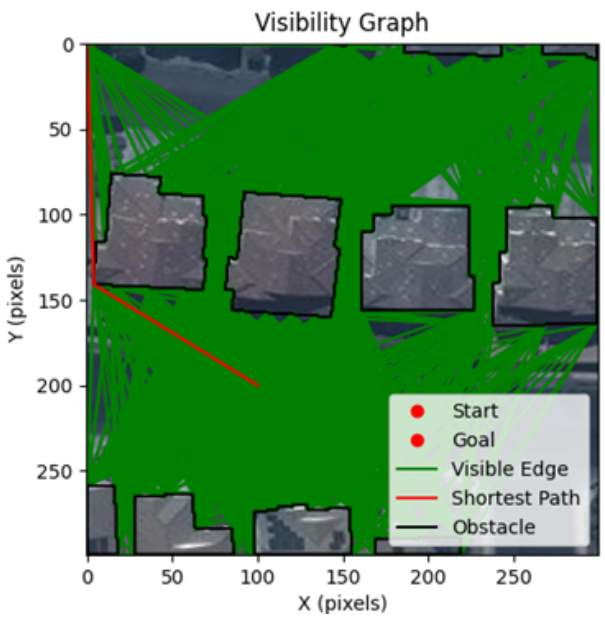
Извлечение полигонов объектов с помощью PolyWorld

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

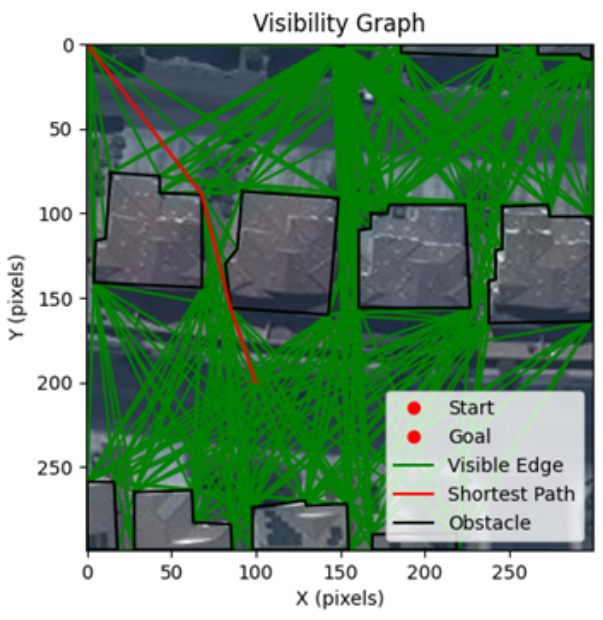
Эксперименты



Граф видимости и PolyWorld



Граф видимости (без аппроксимации)



Граф видимости (с аппроксимацией)

- Показаны изображения с наложенными ребрами и вершинами.

Эксперименты и результаты

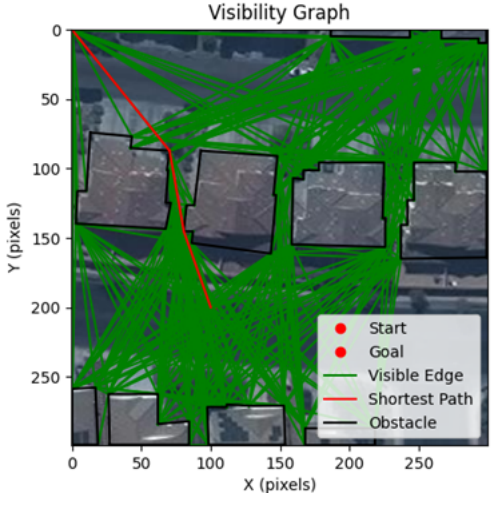
Проведены эксперименты, в которых сравнили различные алгоритмы построения путей, включая граф видимости, Theta* и A*, в одинаковых условиях. Среда была представлена как список вершин препятствий, полученных из изображения 300x300, и в виде сетки занятости для алгоритмов Theta* и A*. Результаты включают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

Experiment	Method	Execution Time
1	PolyWorld + Visibility Graph + Dijkstra	12.7 seconds
2	Non-Neural Network (No Approximation) + Visibility Graph + Dijkstra	170-200 seconds
3	Non-Neural Network (With Approximation) + Visibility Graph + Dijkstra	6.4 seconds
4	Partial Vectorization + PolyWorld + Visibility Graph + Dijkstra	**4.8** seconds
5	Partial Vectorization + Non-Neural Network (With Approximation) + Visibility Graph + Dijkstra	1.53 seconds
6	Partial Vectorization + Non-Neural Network (No Approximation) + Visibility Graph + Dijkstra	36.05 seconds
New Full Vectorization		
7	Full Vectorization + PolyWorld + Visibility Graph + Dijkstra	2.1 seconds
8	Full Vectorization + Non-Neural Network (With Approximation) + Visibility Graph + Dijkstra	**0.05** seconds
9	Full Vectorization + Non-Neural Network (No Approximation) + Visibility Graph + Dijkstra	2 seconds
10	Theta*	6.286 seconds
11	A*	50.1 seconds

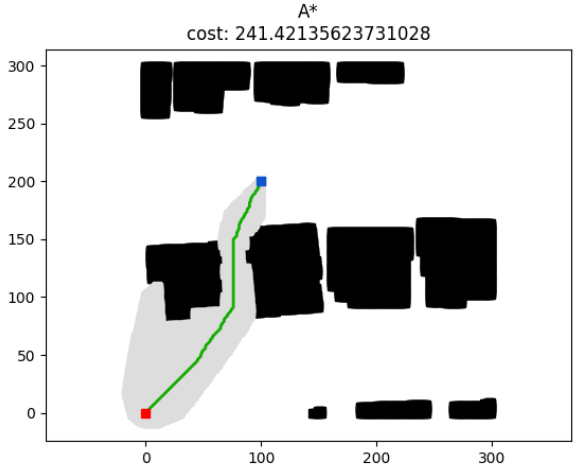
Полная векторизация значительно сокращает время выполнения по сравнению с предыдущими подходами.

Эксперименты

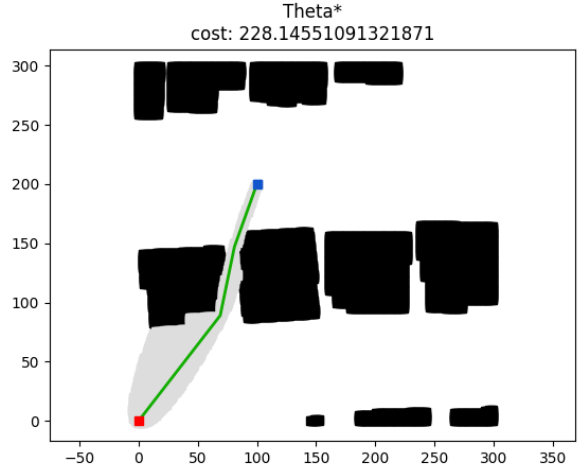
Сравнение различных алгоритмов планирования маршрута, включая Visibility Graph, A* и Theta*.



Visibility graph and PolyWorld



A*

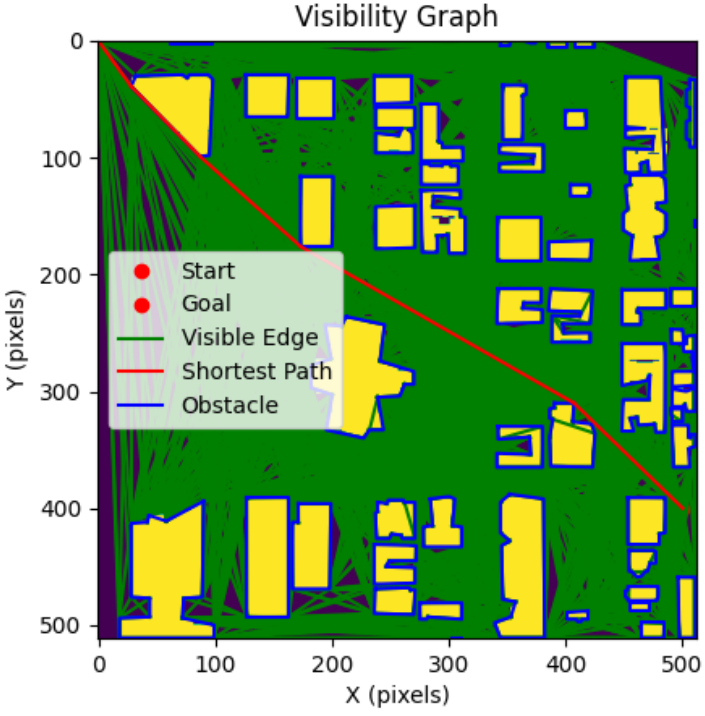


Theta*

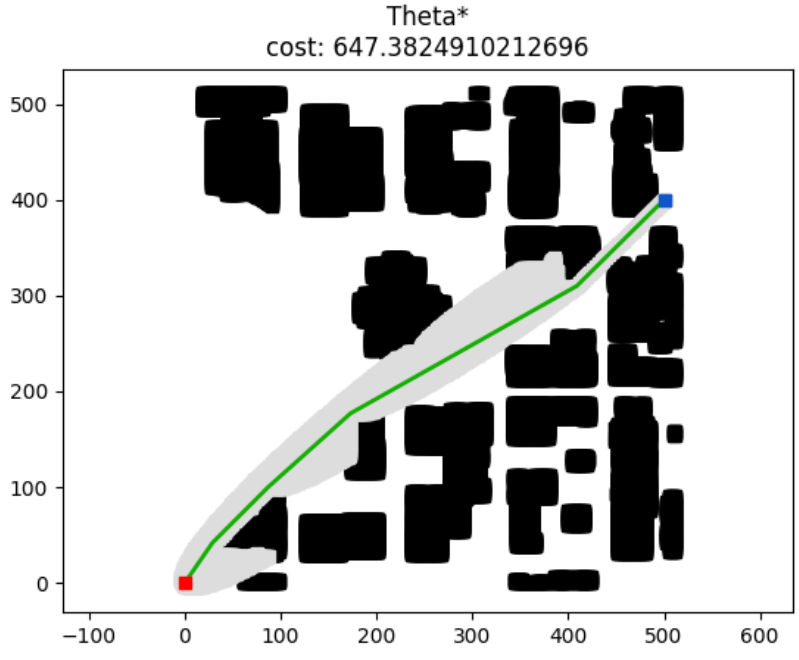
Эксперименты

Эксперименты на большой, загруженной препятствиями карте.

- Theta* занял около 180 секунд для построения пути.
- Visibility graph занял 4.9 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:**
  - Реализация метода в виде ROS-узла для планирования пути.
  - Интеграция разработанного метода в робота и его внедрение в ROS-пайплайн.

Контакты

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