

**ФИЛИАЛ МОСКОВСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА ИМЕНИ М.В. ЛОМОНОСОВА В Г. ЕРЕВАНЕ**

**НАПРАВЛЕНИЕ ПОДГОТОВКИ**

«**Прикладная Математика и Информатика**»

**Курсовая работа:**

«Применение Искусственного Интеллекта в Робототехнике»

**Выполнил:**  
**студент 3 курса направления подготовки «Прикладная математика и информатика»**

**Хроян Георгий Овикович**

**научный руководитель:**

**Смирнов Илья Николаевич**

**к.ф.-м.н., преподаватель Филиала**

**МГУ имени М.В.Ломоносова в г. Ереване**

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_

**Ереван 2024**

## [Содержание](#_Содержание)

[Введение…………………………………………………………………..3](#_Введение)

[Глава 1. Анализ существующих решений………………..5](#_Глава_1._Анализ)

[1.1. Обзор Литературы……………………………………5](#_1.1._Обзор_Литературы)[1.2. Анализ существующих подходов…………….6](#_1.2._Анализ_существующих) **1.3. Выбор подхода**

Глава 2. Разработка и реализация алгоритма

2.1. Описание подхода  
2.2. Программная реализация  
 2.2.1. Описание среды Webots  
 2.2.2. Реализация алгоритма  
 2.2.3. Тестирование и отладка

Глава 3. Демонстрация результатов

Заключение

Список литературы

## **Введение**

**Актуальность темы:**

Робототехника – динамично развивающаяся область, играющая все более важную роль в различных сферах жизни.  
Мобильные роботы, способные самостоятельно ориентироваться и перемещаться в пространстве, востребованы в медицине, логистике, промышленности, сельском хозяйстве и других областях.   
  
В современном мире робототехника играет ключевую роль в решении множества задач, от промышленного производства до исследований в сфере науки и технологий.  
Одним из важных аспектов в этой области является разработка и программирование роботов для автономного перемещения в различных средах.

**Цель работы:**

Разработка и реализация на симуляторе Webots модели робота, способной проходить по оптимальному маршруту к заданной цели в условиях помещения с препятствиями (статичными).

**Задачи работы:**

1. Анализ существующих алгоритмов построения оптимальных маршрутов для мобильных роботов.
2. Выбор и обоснование алгоритма для решения поставленной задачи.
3. Выбор модели робота в среде Webots.
4. Реализация алгоритма построения оптимального маршрута в программном коде.
5. Моделирование различных сценариев движения робота и анализ полученных результатов.
6. Сравнение разработанного алгоритма с другими подходами.

Предмет исследования:

Алгоритмы построения оптимальных маршрутов для мобильных роботов.

**Объект исследования:**

Модель робота, реализованная на симуляторе Webots.

**Методы исследования:**

* Анализ литературных источников;
* Программирование;
* Сравнительный анализ.

**Используемые программные решения:**

* Webots – среда для моделирования роботов;
* Python – язык программирования.

## **Глава 1. Анализ существующих решений**

### **1.1. Обзор Литературы**

**Аннотация**: “Алгоритмы планирования пути используются мобильными роботами, беспилотными летательными аппаратами и автономными автомобилями для определения безопасных, эффективных, без столкновений и наименее затратных маршрутов движения из пункта отправления в пункт назначения”.

Планирование пути является одной из ключевых задач робототехники. Для автономного перемещения роботов в различных средах необходимо разрабатывать эффективные алгоритмы, способные прокладывать оптимальные или близкие к оптимальным траектории с учетом препятствий, ограничений робота и динамических изменений в окружающей среде.  
Планирование пути - это недетерминированная сложная задача с полиномиальным временем (“NP”).

В настоящее время существует множество алгоритмов для нахождения оптимального пути. В ([[1]](#footnote-1)) представлены такие алгоритмы поиска оптимального пути, как алгоритм Дейкстры ([[2]](#footnote-2)), и несколько его вариантов, которые обычно используются в таких приложениях, как Google Maps и других системах маршрутизации трафика. Также представлены алгоритм A\* ([[3]](#footnote-3)) и его варианты, как современные алгоритмы для поиска пути в статических средах, чтобы преодолеть трудоёмкость вычислений алгоритмы Дейкстры при слепом поиске. В ([[4]](#footnote-4)) рассматриваются алгоритм D\* и его вариации, в качестве эффективного инструмента для планирования маршрута в динамически изменяющихся средах. Из-за того, что не гарантируется качество решения в больших динамических средах, при использовании алгоритма D\* или его вариаций, можно использовать алгоритм “Быстрого исследования случайного дерева” (RRT), описанный в ([[5]](#footnote-5)) и гибридный подход, который сочетает алгоритм релаксированного A\* (RA\*) ([[6]](#footnote-6)) и один метаэвристический алгоритм.

Три метаэвристических алгоритма – генетический алгоритм (Generic algorithm), алгоритм колонии муравьев (The ant colony optimization , ACO) и алгоритм светлячка (Firefly algorithm) рассмотрены в ([[7]](#footnote-7)) ([[8]](#footnote-8)) ([[9]](#footnote-9)) соответственно. В ([[10]](#footnote-10)) авторами рассматривается планирование маршрута с использованием процесса имитации отжига (Simulated Annealing (SA)).   
В данной курсовой работе будут рассмотрены выше указанные алгоритмы, алгоритмы A\* и RRT также будут реализованы в программном коде.

### ****1.2. Анализ существующих подходов****

#### **Алгоритм Дейкстры**

Для понимания алгоритма Дейсктры познакомимся с основами графовой теории.

**Основные понятия теории графов**

**Граф** – математическая структура, используемая для представления множества объектов и отношений между ними, состоящая из двух компонентов: вершин и рёбер.

**Вершины** – это отдельные объекты в графах, в задачи поиска пути могут представлять физические места.

**Рёбра** – это связи между вершинами, они бывают взвешенными или невзвешенными, а также направленными или ненаправленными. В взвешенном графе каждому ребру присваивается значение, представляющее стоимость или расстояние между двумя вершинами. В направленном графе рёбра имеют направление, т.е. они ведут от одной вершины к другому.

**Путь в графе** – это последовательность вершин, в которой каждая следующая вершина, связана с предыдущей ребром. Длина пути – сумма весов всех рёбер, если граф взвешенный, или просто количество рёбер в ином случаи.

Алгоритмы поиска пути используют структуру графа и веса его рёбер, чтобы найти оптимальный путь между двумя вершинами. Оптимальным может быть самый дешёвый путь, самый короткий, и т.д.. То есть, оптимальность пути определяется исходя из задачи.

Алгоритм Дейкстры – это классический алгоритм поиска пути, который был разработан *Эдсгером Дейкстрой* в 1959 году. Он используется для поиска кратчайшего пути во взвешенном графе от одной вершины до всех остальных вершин.

Принцип работы алгоритма.

Для начала устанавливается начальная вершина. Алгоритм Дейкстры работает по принципу “жадного ” алгоритма, т.е. на каждом шаге стримиться минимизировать текущую общую стоимость пути.

Сначала инициализируются два множества:

* Множество, которое содержит обработанные вершины ( в самом начале оно пустое) – обозначим A
* Множество, которое содержит все остальные вершины ( в самом начале содержит все вершины графа ) – обозначим NA

Потом каждой вершине присваивается вес, который является минимально известной стоимостью пути от начальной вершины до данной. Для начальной вершины этот вес равен 0, а для всех остальных бесконечности.

На каждом шаге алгоритм выбирает вершину с наименьшим весом из множества NA, перемещает эту вершину в множество A и обновляет веса всех соседей выбранной вершины. Вес соседа обновляется, если через выбранную вершину можно добраться до этого соседа с меньшей стоимостью. Процесс будет продолжаться, пока не будут посещены все вершины, либо не найдётся путь до конечной вершины, при условии, что она задана.

1 Функция Dijkstra**(**Graph**,** source**):**

2

3 Для каждой вершины v в Graph**.**Вершины**:**

4 dist**[**v**]** ← БЕСКОНЕЧНОСТЬ

5 prev**[**v**]** ← НЕОПРЕДЕЛЕНО

6 Добавить v в Q

7 dist**[**source**]** ← 0

8

9 Пока Q не пусто**:**

10 u ← вершина в Q с **min** dist**[**u**]**

11 Удалить u из Q

12

13 Для каждого соседа v для u всё ещё находявшегося в Q**:**

14 alt ← dist**[**u**]** **+** Graph**.**Рёбра**(**u**,** v**)**

15 Если alt **<** dist**[**v**]:**

16 dist**[**v**]** ← alt

17 prev**[**v**]** ← u

18

19 Вернуть dist**[],** prev**[]**

<- Псевдокод алгоритма Дейкстры:

dist **-** массив**,** который содержит текущее растояние от source до других вершин**.**

т**.**е**.** dist**[**u**]** **-** текущее растояние с вершины source до вершины u**.**

prev содержит указатели на узлы предыдущего перехода на кратчайшем пути от источника к данной вершине

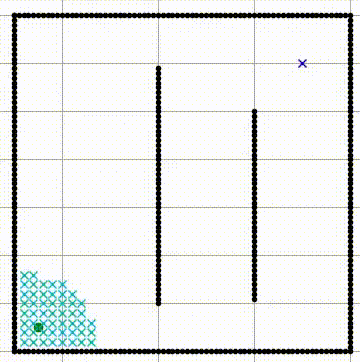
u ← вершина в Q с **min** dist**[**u**]**

выполняет поиск вершины u в наборе вершин Q**,** которая имеет наименьшее значение dist**[**u**].**

Graph**.**Рёбра**(**u**,** v**)** **-** возвращает возвращает длину ребра**,** соединяющего **(**т**.**е**.** расстояние между**)**

два соседних узла u и v**.**

Переменная alt в строке 14 **-** это длина пути от корневого узла до соседнего узла v**,** если бы он проходил через u**.** Если этот путь короче текущего кратчайшего пути**,**

**записанного для v**,** этот текущий путь заменяется этим путем alt**.**

*Пример работы алгоритма Дейкстры с представлением Графа в виде сетки, где каждая вершина может быть либо проходимой, либо препятствием.*

Недостатками алгоритма Дейкстры являются то, что он требует большое количество памяти, поскольку ему приходится вычислять все возможные исходы, чтобы определить кратчайший путь, и он не может обрабатывать отрицательные ребра. Его вычислительная сложность равна O(n2), где n - количество узлов.   
Модификация алгоритма Дейкстры представленная в ([[11]](#footnote-11)) вводит схему хранения на основе многоуровневого словаря, состоящего из двух словарей и списка структур данных, организованных иерархически. Первый словарь отображает каждый узел на его соседние узлы. Второй словарь хранит информацию о пути для каждого соседнего маршрута. Многоуровневый словарь предоставляет комплексную структуру данных для алгоритма Дейкстры в приложениях внутренней навигации, где координаты Глобальной Навигационной Спутниковой Системы (ГНСС) и направление компаса не надежны.   
Алгоритм Флойда — популярный графовый алгоритм для поиска кратчайшего пути в графе с положительными или отрицательными весами, в то время как Дейкстра лучше всего подходит для поиска кратчайшего пути от одного источника (поиск кратчайших путей от исходной вершины ко всем остальным вершинам в графе) в графе с положительными весами.  
В ([[12]](#footnote-12)) представлена также ещё одна модификация алгоритма Дейкстры.   
Алгоритм Дейкстры лучше всего подходит для статической среды и/или глобального планирования маршрута, поскольку большинство требуемых данных заранее определены для вычисления кратчайшего пути; однако есть приложения, в которых алгоритм Дейкстры использовался для динамических сред. В этом случае окружающая среда частично известна или полностью неизвестна, и, таким образом, информация об узлах в отношении препятствий вычисляется "на лету"; это называется локальным планированием, и алгоритм Дейкстры запускается для оценки вычисления кратчайшего пути. Однако мы не можем использовать только алгоритм Дейкстры в динамических средах.

**Таблица 1:** Пригодность алгоритма Дейкстры и его вариантов, классифицированных на основе статических и динамических ограничений окружающей среды.

| **Алгоритм** | **Статические ограничения** | **Динамические ограничения** |
| --- | --- | --- |
| Дейкстра | x |  |
| Улучшенный Дейкстра | x |  |
| Многоуровневый словарь | x | x |
| Флойд и Дейкстра | x |  |

#### **Алгоритм А\***

**Алгоритм A\*** — это популярный алгоритм поиска пути, часто используемый в областях, таких как искусственный интеллект и компьютерные игры. Алгоритм A\* был предложен *Питером Хартом, Нильсом Нильсоном и Бертраном Рафаэлем* в **1968 году**.

Принцип работы алгоритма

Алгоритм А\* работает на основе оценки стоимости пути до цели, которая вычисляется как сумма двух компонент:  
 - известная стоимость пути от начальной вершины до текущей.  
 - эвристическая оценка стоимости пути от текущей вершины до цели.  
Эта сумма даёт оценку общей стоимости через данную вершину. Эвристика играет ключевую роль в алгоритме A\*. Она должна быть адекватной, чтобы алгоритм был эффективным.   
На каждом шаге алгоритм выбирает вершину с наименьшей оценкой из списка открытых вершин (вершин, которые уже были обнаружены, но еще не обработаны), затем смотрит на соседей этой вершины и обновляет их стоимости , если через текущую вершину можно добраться до них быстрее. Если вершина еще не была открыта, она добавляется в список открытых вершин. Процесс продолжается, пока не будет найден путь до целевой вершины или пока не кончатся вершины в списке открытых вершин (что означает, что путь не существует).

Псевдокод алгоритма A\*

1 Функция A**\*(**start**,** goal**,** f**)**

2 # множество уже пройденных вершин

3 var closed **:=** пустое множество

4 # множество частных решений

5 var **open** **:=** make\_queue**(**f**)**

6 enqueue**(open,** path**(**start**))**

7 Пока **open** не пусто

8 var p **:=** remove\_first**(open)** # извлекаем первое значение из open

9 var x **:=** последняя вершина в p # p - path\путь

10 если x в closed

11 **продолжить** # если уже в closed игнорируем вершину и

12 #переходим к след. итерации

13 если x равен goal

14 **вернуть** p

15 добавить**(**closed**,** x**)** # отмечаем, что х уже пройдена,

16 #добавляя в closed

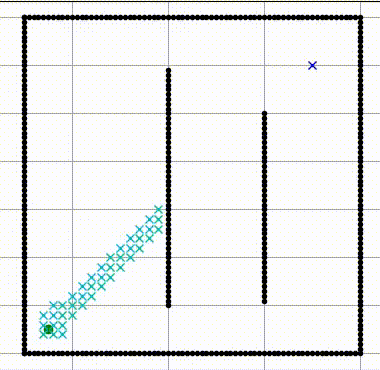
17 # добавляем смежные вершины

18 для каждого y в преемниках**(**x**)** # для каждой смежной вершины

19 enqueue**(open,** add\_to\_path**(**p**,** y**))** # добавляем вершину в путь

20 # и новый путь помещаем в open

21 **вернуть** None # если список пуст и p не найден – возвращаем None

*Пример работы алгоритма A\* с представлением Графа в виде сетки, где каждая вершина может быть либо проходимой, либо препятствием.*

Алгоритм A\* широко используется в статических средах, хотя есть случаи его использования и в динамический средах.

**Таблица 2.** Наиболее распространенные типы Эвристических Функций, используемых в алгоритмах планирования маршрута.

| **Функция** | **Уравнение** |
| --- | --- |
| Евклидово расстояние |  |
| Манхэттенское расстояние |  |
| октильное расстояние | max|𝑥1−𝑥2|+|𝑦1−𝑦2| |

#### **Алгоритм D\***

Для поиска пути в динамических средах используется алгоритм D\* ( или динамический алгоритм A\*), который позволяет создавать траекторию без столкновений среди движущихся препятствий.   
Алгоритм D\* - алгоритм поиска кратчайшего пути во взвешенном ориентированном графе, где структура графа неизвестна заранее или постоянно меняется. Был разработан *Свеном Кёнигом и Максимом Лихачевым* в 2002 году.

Принцип работы алгоритма

D\* аналогично алгоритмам Дейкстре и A\*, использует список узлов, называемый "открытым списком", для оценки. Узлы могут иметь различные состояния:

* **"Новый"**: узел никогда не был добавлен в открытый список.
* **"Открытый"**: узел в настоящее время находится в открытом списке.
* **"Закрытый":** узел больше не находится в открытом списке.
* **"Поднятый"**: стоимость узла выше, чем в последний раз, когда он был в открытом списке.
* **"Нижний"**: стоимость узла ниже, чем в последний раз, когда он был в открытом списке.

**Расширение**

Алгоритм **D\*** работает путём последовательного выбора узла из списка **открытых** узлов и его оценки. Затем он передаёт изменения этого узла на все его соседей и добавляет их в список открытых узлов. Этот процесс называется "*расширением*". В отличие от **A\***, который идёт от начала к концу пути, **D\*** начинает поиск с конечного узла. Каждый расширенный узел имеет указатель на предыдущий узел, который ведёт к конечной точке, и каждый узел знает точную стоимость достижения конечного узла. Когда начальный узел становится следующим узлом для расширения, алгоритм завершается, и путь к цели можно найти, следуя обратным указателям.

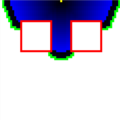
**Figure 1. Расширение в процессе. Финишный узел (жёлтый) находится в середине верхнего ряда точек, начальный узел находится в середине нижнего ряда. Красный указывает на препятствие; чёрный/синий обозначает расширенные узлы (яркость обозначает стоимость). Зелёным обозначены узлы, которые расширяются.**

Figure 1

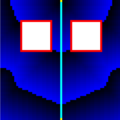
**Figure 2. Расширение завершено. Путь указан голубым цветом.**

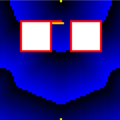
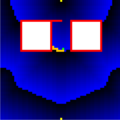
Figure 2

**Преодоление препятствий**

При обнаружении препятствия на пути все затронутые точки снова добавляются в открытый список, но уже с меткой "**Поднятый**". Однако прежде чем увеличить стоимость такого узла, алгоритм проверяет его соседей и рассматривает возможность снижения стоимости. Если это невозможно, то состояние "**Поднятый**" распространяется на всех потомков узлов, то есть на узлы, на которые указывают обратные ссылки. Затем эти узлы оцениваются, и состояние "**Поднятый**" передаётся, создавая волну. Когда узел, помеченный как "**Поднятый**", может быть уменьшен, его обратная ссылка обновляется, и состояние "**Нижний**" передаётся его соседям. Эти волны состояний "**Поднятый**" и "**Нижний**" являются сутью D\*.

К этому моменту эти волны не затрагивают целый ряд других точек, поэтому алгоритм работает только с теми точками, которые подвержены изменению стоимости.

Начало формы

Figure 3. Препятствие добавлено (красный) и узлы отмечены как **ПОДНЯТЫЕ** (жёлтый).

Изображение 2

Figure 3

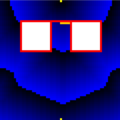
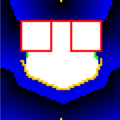
Изображение 1

Figure 4

Figure 4. Расширение в процессе. Жёлтым цветом обозначены узлы, отмеченные как **ПОДНЯТЫЕ**, зелёным — узлы, отмеченные как **НИЖНИЕ**.

**Ещё один тупик**

На этот раз выход из тупика не столь прост. Ни одна из точек не может найти новый маршрут через своих соседей к целевой точке. Поэтому они продолжают распространять повышение стоимости. Единственными точками, способными привести к пункту назначения по приемлемому маршруту, оказываются те, что находятся за пределами блокирующего препятствия. Таким образом, формируются две волны с состоянием "Нижний", которые распространяются в виде точек, отмеченных как недостижимые с учетом новой информации о маршруте.

Figure 5. Канал заблокирован дополнительными препятствиями (красный).

Изображение 3

Изображение 2

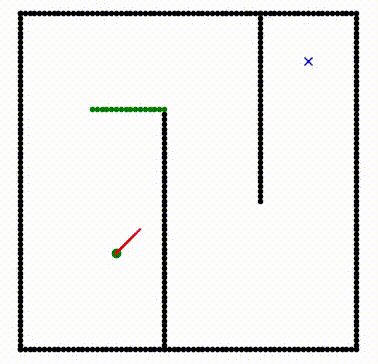
Изображение 1  
Псевдокод

Figure 6. Расширение в процессе. **ПОДНЯТАЯ** волна (жёлтый), **НИЖНЯЯ** волна (зелёный).

Figure 6

Figure 7

Figure 5  
Псведокод

****Figure 7. Найден **НОВЫЙ** путь (голубой).

1 **while** **(!**openList**.**isEmpty**())** **{**

2 point **=** openList**.**getFirst**();**

3 expand**(**point**);**

4 **}**

5 void expand**(**currentPoint**)** **{**

6 boolean isRaise **=** isRaise**(**currentPoint**);**

7 double cost**;**

8 **for** each **(**neighbor **in** currentPoint**.**getNeighbors**())** **{**

9 **if** **(**isRaise**)** **{**

10 **if** **(**neighbor**.**nextPoint **==** currentPoint**)** **{**

11 neighbor**.**setNextPointAndUpdateCost**(**currentPoint**);**

12 openList**.**add**(**neighbor**);**

13 **}** **else** **{**

14 cost **=** neighbor**.**calculateCostVia**(**currentPoint**);**

15 **if** **(**cost **<** neighbor**.**getCost**())** **{**

16 currentPoint**.**setMinimumCostToCurrentCost**();**

17 openList**.**add**(**currentPoint**);**

18 **}**

19 **}**

20 **}** **else** **{**

21 cost **=** neighbor**.**calculateCostVia**(**currentPoint**);**

22 **if** **(**cost **<** neighbor**.**getCost**())** **{**

23 neighbor**.**setNextPointAndUpdateCost**(**currentPoint**);**

24 openList**.**add**(**neighbor**);**

25 **}**

26 **}**

27 **}**

28 **}**

29 boolean isRaise**(**point**)** **{**

30 double cost**;**

31 **if** **(**point**.**getCurrentCost**()** **>** point**.**getMinimumCost**())** **{**

32 **for** each**(**neighbor **in** point**.**getNeighbors**())** **{**

33 cost **=** point**.**calculateCostVia**(**neighbor**);**

34 **if** **(**cost **<** point**.**getCurrentCost**())** **{**

35 point**.**setNextPointAndUpdateCost**(**neighbor**);**

36 **}**

37 **}**

38 **}**

39 **return** point**.**getCurrentCost**()** **>** point**.**getMinimumCost**();**

40 **}**

*Пример работы алгоритма D\* с представлением Графа в виде сетки, где каждая вершина может быть либо проходимой, либо препятствием*

Существуют также модификации алгоритма D\*, например D\* Lite представленный в ([[13]](#footnote-13)), Enhanced D\* Lite ([[14]](#footnote-14)), Field D\* ([[15]](#footnote-15)).

**Таблица 3:** Классификация алгоритма D\* и его вариаций.

| **Алгоритм** | **Статические ограничения** | **Динамические ограничения** |
| --- | --- | --- |
| D\* |  | x |
| D\* Lite |  | x |
| Enhanced D\* Lite |  | x |
| Field D\* |  | x |

#### **Алгоритм “Быстрого исследования случайного дерева” (RRT)**

Довольно часто, при планировании маршрута, нам нужно просто добраться из точки А в точку B, не зная ничего о предыдущих запросах планирования. Например, если среда между запросами планирования изменяется, то часто лучше перепланировать маршрут с нуля, чем воссоздать информацию, полученную из прошлых запросов.

Одним из методов, которые применяются при таких случаях – замена графа на дерево.  
**Дерево** –этопросто особый тип графа, в котором узлы упорядочены в “родительские” и “дочерние”. Каждый узел имеет один родительский узел и ноль или более дочерних узлов.

Если мы будем хранить состояния, в которых был агент, в качестве родителей, а все состояния, в которые агент может перейти из них в качестве дочерних узлов, то мы сможем сформировать древоподобную структуру, растущую из текущего состояния агента растущую во все места, в которых агент может находится. Рано или поздно дерево достигнет целевого состояния, нужного нам и у нас появится решение.

Задачу выращивания случайного дерева выполняют два класса алгоритмов:  
1. EST ( “Expansive Space Trees” или “расширяемые пространственные деревья” ), этот подход выбирает случайный узел в дереве и растёт в случайном направлении.   
2. RRT ( “Rapidly Exploring Randomized Trees”, “Быстрого исследования случайного дерева” )**,** этот подход начинается со случайной выборки узлов в пространстве, а затем выращивает ближайший узел по направлению к этой случайной выборке. Поговорим об RRT.

Принцип работы алгоритма

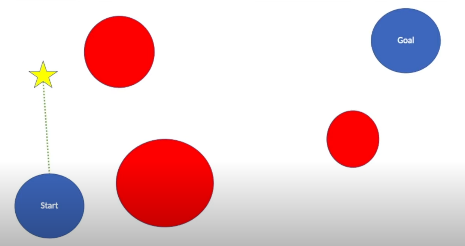
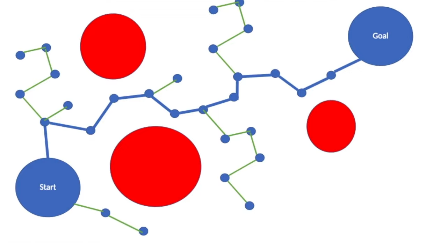
* Создаём пустое дерево T.
* Добавляем в Т начальное состояние агента.
* В течении N итераций, или пока не достигнута цель
  + Случайным образом делаем выборку для узла R.
  + Находим в Т ближайший к R узел. Называем его K.
  + Делаем шаги по лучу от К до R на малую величину эпсилон, пока не выполнится следующее условие:
    - При наличии коллизии возвращаемся к созданию случайной выборки.
    - В противном случае добавляем к Т новый узел в этой конфигурации.
    - Если мы достигли максимального расстояния d от К, то возвращаемся к созданию случайной выборки.
* Если узел цели теперь находится в пределах расстояния d от любого узла дерева, то у нас есть решение.

Figure 10

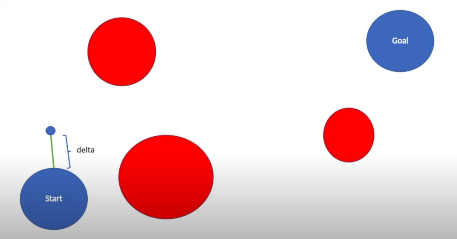
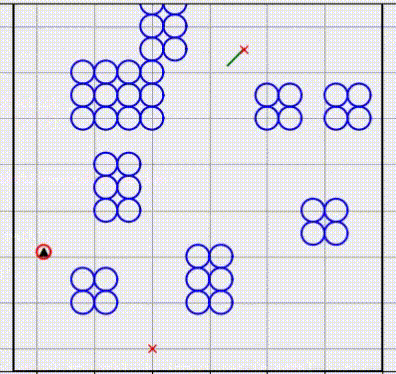
  
*Figure 8: выбираем случайную точку в окружающей среде (звезда), провод линию между начальным узлом и этой точкой.  
Figure 9: Помещаем новую точку на расстоянии* ***delta*** *вдоль этой линии, при условии, что это не приводит к столкновению.  
Figure 10: на каждом шаге повторяем это действие, до тех пор, пока не достигнем цели.*

Figure 9

Figure 8

*****Пример работы алгоритма RRT.*

У алгоритма RRT существует множество вариаций, два из них RRT-connect ([[16]](#footnote-16)), RRT\* ([[17]](#footnote-17)).  
RRT-connect - выращивает два дерева из начала и из цели, и пытается соединить их прямой линией через случайные интервалы.  
RRT\* - изобретён в прошлом десятилетии. Гарантирует оптимальность благодаря ребалансировке дерева. В тысячи раз медленнее, чем RRTConnect.

**Таблица 4**. Классификация алгоритма RRT и его вариантов на основе статических и динамических ограничений.

| **Алгоритм** | **Статические ограничения** | **Динамические ограничения** |
| --- | --- | --- |
| RRT |  | x |
| RRT Connect |  | x |
| RRT\* |  | x |

1. **1. Karur, K., Sharma, N., Dharmatti, C., & Siegel, J. (2021). A Survey of Path Planning Algorithms for Mobile Robots. Vehicles, 3(3), 448-468.** [**[https://doi.org/10.3390/vehicles3030027]**](https://doi.org/10.3390/vehicles3030027) [↑](#footnote-ref-1)
2. **2. Dijkstra, E. A note on two problems in connexion with graphs. *Numer. Math.* 1959, *1*, 269–271.** [**[https://doi.org/10.1007/BF01386390]**](https://doi.org/10.1007/BF01386390) [↑](#footnote-ref-2)
3. **. Hart, P.E.; Nilsson, N.J.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* 1968, *4*, 100–107.** [**[https://doi.org/10.1109/TSSC.1968.300136]**](https://doi.org/10.1109/TSSC.1968.300136) [↑](#footnote-ref-3)
4. **. Stentz, A. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*; Springer: Berlin/Heidelberg, Germany, 1997; pp. 203–220.** [**[Google Scholar]**](https://scholar.google.com/scholar_lookup?title=Optimal+and+efficient+path+planning+for+partially+known+environments&author=Stentz,+A.&publication_year=1997&pages=203%E2%80%93220) [↑](#footnote-ref-4)
5. **. LaValle,S.M. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998.** [**[http://lavalle.pl/papers/Lav98c.pdf]**](http://lavalle.pl/papers/Lav98c.pdf) [↑](#footnote-ref-5)
6. **Ammar, A.; Bennaceur, H.; Chaari, I.; Koubaa, A.; Alajlan, M. Relaxed Dijkstra and A\* with linear complexity for robot path planning problems in large-scale grid environments. *Soft Comput.* 2015, *20*, 1–23.** [**[https://doi.org/10.1007/s00500-015-1750-1]**](https://doi.org/10.1007/s00500-015-1750-1) [↑](#footnote-ref-6)
7. **. Burchardt, H.; Salomon, R. Implementation of Path Planning using Genetic Algorithms on Mobile Robots. In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 1831–1836.** [**[https://doi.org/10.1109/CEC.2006.1688529]**](https://doi.org/10.1109/CEC.2006.1688529) [↑](#footnote-ref-7)
8. **. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* 1996, *26*, 29–41.** [**[https://doi.org/10.1109/3477.484436]**](https://doi.org/10.1109/3477.484436) [↑](#footnote-ref-8)
9. **. Fister, I., Jr.; Perc, M.; Kamal, S.; Fister, I. A review of chaos-based Firefly algorithms: Perspectives and research challenges. *Appl. Math. Comput.* 2015, *252*.** [**[https://doi.org/10.1016/j.amc.2014.12.006]**](https://doi.org/10.1016/j.amc.2014.12.006) [↑](#footnote-ref-9)
10. **. Tsuzuki, M.S.G.; Martins, T.C.; Takase, F.K. Robot path planning using simulated annealing. INCOM'2006: 12th IFAC/IFIP/IFORS/IEEE/IMS Symposium Information Control Problems in Manufacturing, May 17-19, 2006, Saint-Etienne, France. 1-6.** [**[https://doi.org/10.3182/20060517-3-FR-2903.00105]**](https://doi.org/10.3182/20060517-3-FR-2903.00105) [↑](#footnote-ref-10)
11. **. Syed Abdullah, F.; Iyal, S.; Makhtar, M.; Jamal, A.A. Robotic Indoor Path Planning Using Dijkstra’s Algorithm with Multi-Layer Dictionaries. In Proceedings of the 2015 2nd International Conference on Information Science and Security (ICISS), Seoul, Korea, 14–16 December 2015.** [**[https://doi.org/10.1109/ICISSEC.2015.7371031]**](https://doi.org/10.1109/ICISSEC.2015.7371031) [↑](#footnote-ref-11)
12. **. Kang, H.; Lee, B.h.; Kim, K. Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm. In Proceedings of the 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, Wuhan, China, 19–20 December 2009; Volume 18, pp. 1002–1004.** [**[https://doi.org/10.1109/PACIIA.2008.376]**](https://doi.org/10.1109/PACIIA.2008.376) [↑](#footnote-ref-12)
13. **S. Koenig and M. Likhachev. D\* Lite. In*AAAI Conference of Artificial Intelligence (AAAI),* pages 476-483, 2002.** [**[http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf]**](http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf) [↑](#footnote-ref-13)
14. **Yun, S. C., Ganapathy, V., & Chien, T. W. (2010). Enhanced D∗ Lite Algorithm for mobile robot navigation. 2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA). doi:10.1109/isiea.2010.5679403** [↑](#footnote-ref-14)
15. **Ferguson, D.; Stentz, A. Field D\*: An Interpolation-Based Path Planner and Replanner. In *Robotics Research*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 28, pp. 239–253.** [**[https://doi.org/10.1007/978-3-540-48113-3\_22]**](https://doi.org/10.1007/978-3-540-48113-3_22) [↑](#footnote-ref-15)
16. **Kuffner, J. J., & LaValle, S. M. (n.d.). RRT-connect: An efficient approach to single-query path planning. Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065). doi:10.1109/robot.2000.844730**[**[https://doi.org/10.1109/ROBOT.2000.844730]**](https://doi.org/10.1109/ROBOT.2000.844730) [↑](#footnote-ref-16)
17. **Karaman, S., & Frazzoli, E. (2010). Optimal kinodynamic motion planning using incremental sampling-based methods. 49th IEEE Conference on Decision and Control (CDC). doi:10.1109/cdc.2010.5717430**[**[https://doi.org/10.1109/CDC.2010.5717430]**](https://doi.org/10.1109/CDC.2010.5717430) [↑](#footnote-ref-17)