|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Системы автоматизированного проектирования (РК6)

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***«Планирование перемещений в среде с препятствиями при помощи метода Искусственных Потенциальных Полей»***

Студент РК6-82Б **Мудриченко М.Н.**

(Подпись, дата)Фамилия, И.О.

Руководитель курсового проекта **Божко А.Н.**

(Подпись, дата)Фамилия, И.О.

*Москва, 2023 г.*

Оглавление

[Цель работы 3](#_Toc137734331)

[Задачи исследования в курсовом проекте 3](#_Toc137734332)

[1. Постановка задачи планирования движения 4](#_Toc137734333)

[2. Актуальность 6](#_Toc137734334)

[3. Конфигурационное пространство 10](#_Toc137734335)

[4. Классификация методов решения задач планирования движения 11](#_Toc137734336)

[5. Метод Искусственных Потенциальных Полей 12](#_Toc137734337)

[5.1. Описание метода. 12](#_Toc137734338)

[5.2. Потенциалы притяжения и отталкивания. 14](#_Toc137734339)

[5.4. Алгоритм Brushfire. 15](#_Toc137734340)

[5.4. Градиентный спуск. 18](#_Toc137734341)

[5.5. Алгоритм поиска наилучшего. 20](#_Toc137734342)

[6. Программная реализация. 21](#_Toc137734343)

[6.1. Описание среды программирования PYTHON. 21](#_Toc137734344)

[6.2. Описание работы программы 23](#_Toc137734345)

[6.3. Результаты работы программы. 24](#_Toc137734346)

[Заключение 30](#_Toc137734347)

[Список литературы 31](#_Toc137734348)

[Листинг 35](#_Toc137734349)

# Цель работы

Реализовать алгоритм планирования движения при помощи метода Искусственных Потенциальных Полей на языке PYTHON.

# Задачи исследования в курсовом проекте

1. Изучить задачу планирования движения

2. Классифицировать методы подходов к решению задачи планирования движения

3. Реализовать алгоритм метода Искусственны Потенциальных Полей на языке python.

-

# Постановка задачи планирования движения

Задача планирования движения задается описанием кинематики моделируемого устройства (манипулятора, робота), его окружения (препятствий, создающих помехи при движении устройства), а также начальной и целевой конфигураций устройства.

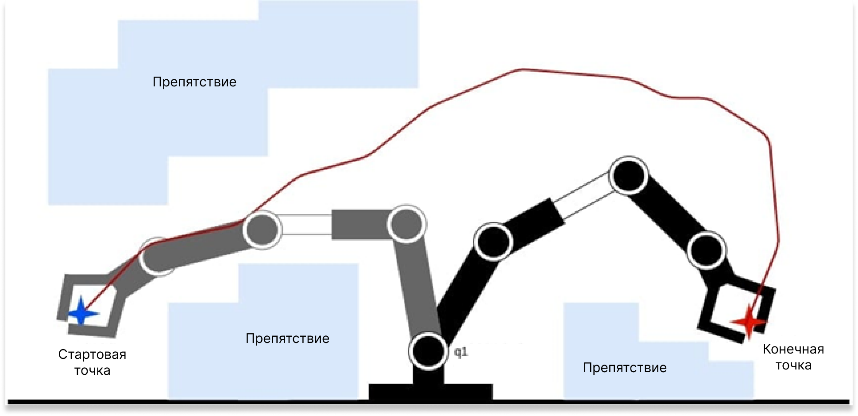
Решением задачи планирования движения является свободный от столкновений допустимый путь от начальной к целевой конфигурации. Окружение описывается геометрической моделью, задающей форму и положение всех препятствий. Устройство также описывается геометрической моделью — множеством перемещаемых тел, связанных кинематическими парами. Современный подход к решению задачи планирования движения состоит в использовании алгоритмов случайного поиска на дорожной карте (англ. roadmap). Дорожная карта — это граф, каждая вершина которого представляет собой точку в конфигурационном пространстве, описывающую положение устройства, не находящегося в контакте с препятствиями, а ребро представляет собой свободный от столкновений путь. 

Рисунок Автономно запрограммированный робот-манипулятор

Поскольку планирование маршрута, как правило, допускает бесконечное множество решений (хотя может не существовать ни одного решения), иногда данную задачу формулируют в постановке оптимизационной задачи с целевой функцией, соответствующей минимальной длине маршрута или максимальной удаленности перемещаемого объекта от препятствий. На практике поиск пути даже в простых сценах с относительно небольшим количеством препятствий становится трудноразрешимой задачей, если перемещаемый объект имеет сложную геометрию или высокое число степеней свободы. В современных индустриальных приложениях часто требуется моделировать поведение сложных кинематических систем с шестью и более степенями свободы в статическом или динамическом окружении, насчитывающим тысячи препятствий.

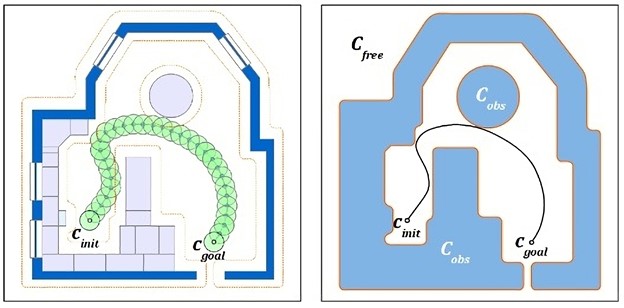


Рисунок Конфигурационные пространства двумерного твердого тела

# 2. Актуальность

Автоматизация любой роботизированной системы невозможна без решения задачи планирования движения. Планирование движения манипулятора заключается в нахождении кратчайших и свободных от столкновений путей от начального до конечного положения манипулятора (рис.1). В настоящее время манипуляторы либо управляются людьми [14], либо они ограничены в траекториях, которые были предварительно вычислены.

Автоматизация планирования движения дает ряд преимуществ по сравнению с существующими альтернативами. Это позволяет работникам не отвлекаться на детальное проектирование траекторий движения и предотвращение столкновений, а позволяет им сосредоточиться на задачах более высокого уровня. Роботы с технологией автоматического планирования движений могут выполнять задачи с меньшим количеством оперативных команд высокого уровня, в следствии чего выполнении задач роботами Автономно запрограммированный робот-манипулятор получается гораздо эффективнее, поскольку команды роботам можно будет отдавать с минимальным интервалом времени.

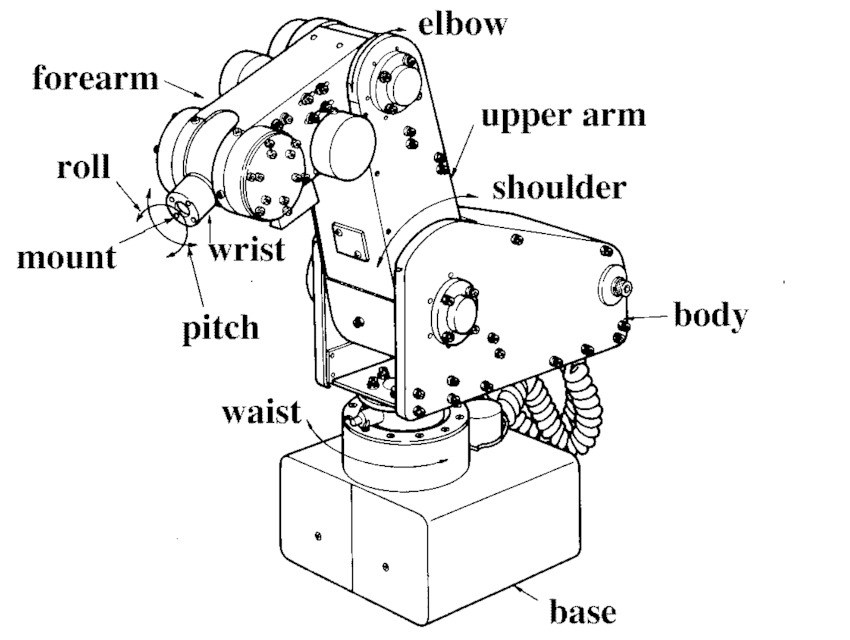


Рисунок . Изображение актуального в производстве манипулятора

С технологией планирования движения и соответствующей системой датчиков, роботы могут быстро адаптироваться к неожиданным изменениям в окружающей среде, обрабатывать и устранять ошибки совершенные в моделировании.

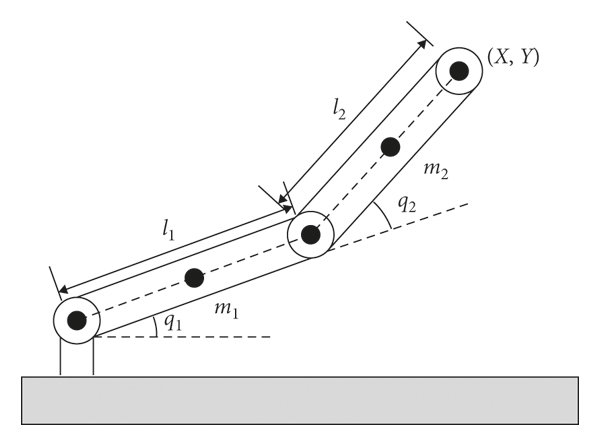


Рисунок . Схематичное изображения манипулятора с 2-мя степенями свободы

Технология планирования движения в “сыром виде” способна довольно быстро вычислять траектории в системах с двумя-тремя степенями свободы (рис.2-3). Однако, в большинстве современных роботах, количество степеней свободы начинается от 5, и может даже исчислять 10 и более (рис.4-6).

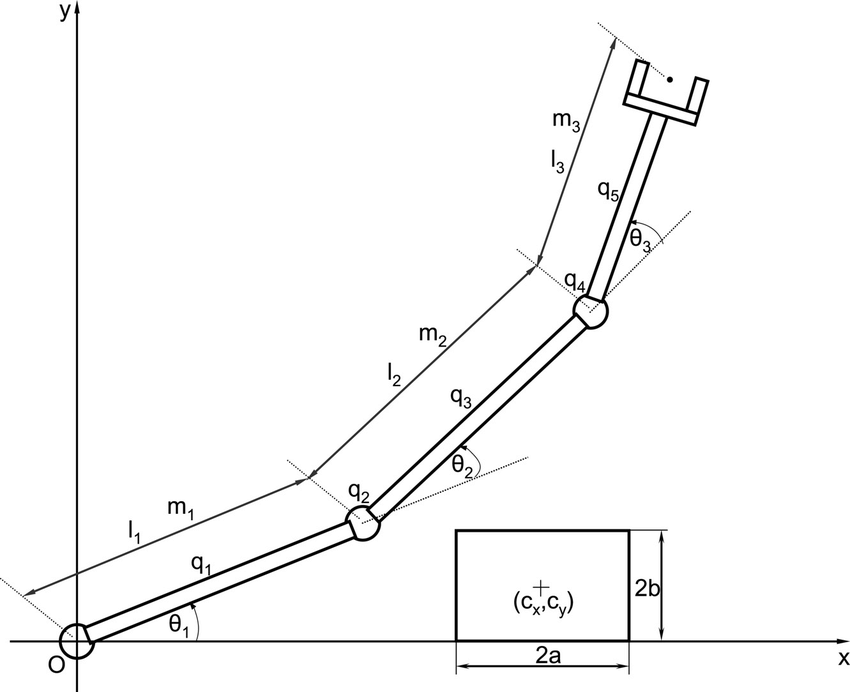


Рисунок . Схематичное изображения манипулятора с 3-мя степенями свободы

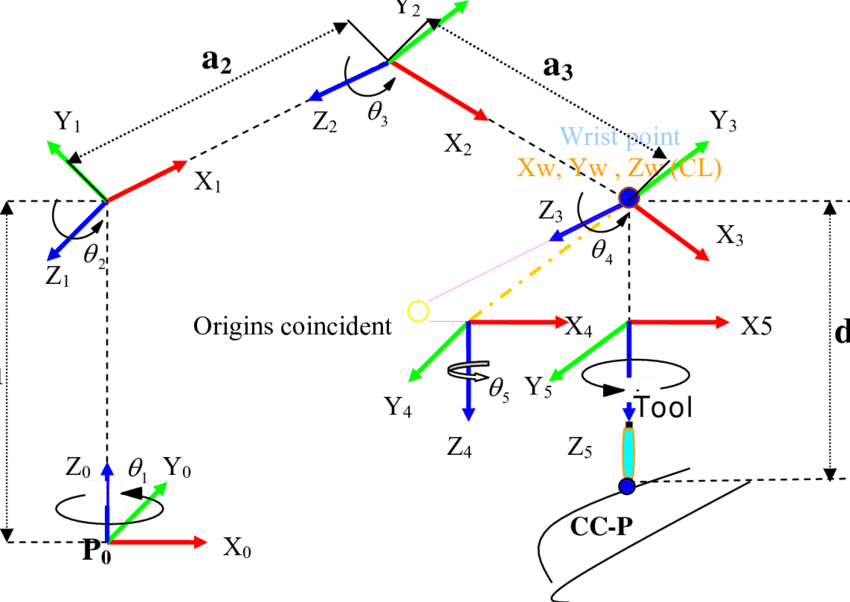
Необходимо использовать алгоритм с эффективной сложностью вычислений. Такая эффективная сложность представляется “планировщиком движения” алгоритмом, где производительность пропорциональна сложности задачи. 

Рисунок . Кинематическое представление системы с 5-ти степенями свободы

Технология совместима с любыми видами манипуляторов и может очень быстро решать “легкие” задачи, а “сложные” - систематически с постепенным увеличением времени вычислений. Для типичных задач и роботов с 5 или 6 степенями время вычисления варьируется от 20 секунд до 10 минут. В ближайшем будущем общая производительность должна повыситься, такие задачи станут выполняться в режиме реального времени благодаря более быстрым компьютерам.

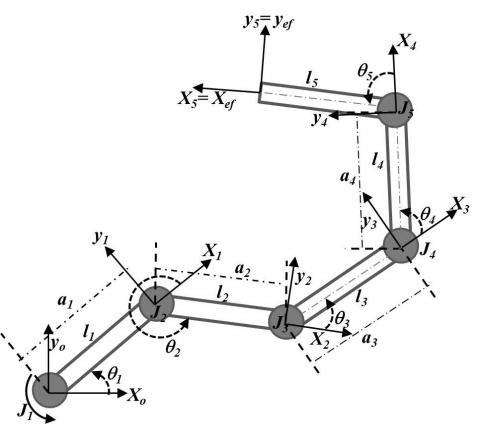


Рисунок . Схематичное изображения манипулятора с 5-ти степенями свободы

# 3. Конфигурационное пространство

Конфигурационное пространство, n-мерное пространство с числом измерений, равным числу n степеней свободы системы, вводимое для условного представления движения всей системы как движения некоторой точки в этом пространстве. При движении механической системы по отношению к некоторой системе отсчёта её конфигурацию, т. е. положение самой системы и взаимное расположение её частей, можно в любой момент времени определять обобщёнными координатами q1, q2,..., qn. Если эти координаты рассматривать как n декартовых координат в n-мерном пространстве, то каждой конфигурации системы будет соответствовать определённая точка в этом пространстве, называемая изображающей точкой. Такое пространство и называется К. п. У систем с 1, 2 и 3 степенями свободы (например, у плоского математического маятника, у сферического маятника и у свободной материальной точки) К. п. будут соответственно прямая, плоскость и 3-мерное пространство; у свободного твёрдого тела, имеющего 6 степеней свободы, К. п. будет 6- мерным и т. д. Конфигурационное пространство двумерного твёрдого тела. При движении системы её конфигурация будет непрерывно изменяться и изображающая точка будет тоже непрерывно менять своё положение в К. п., описывая кривую, называемую условно «траекторией системы». Следовательно, движение системы можно представить как движение в К. п. изображающей точки. Такое представление используют при рассмотрении некоторых свойств движущейся системы, в частности свойств, устанавливаемых рядом вариационных принципов механики. Искомый путь представляет собой непрерывную кривую в конфигурационном пространстве объекта, которая соединяет его начальное и конечное положения, исключает столкновения с препятствиями сцены и удовлетворяет всем установленным кинематическим и динамическим ограничениям. Конфигурационное пространство в роботизированных системах - это представление всевозможных движений робота, учитывая положения и углы его шарниров.

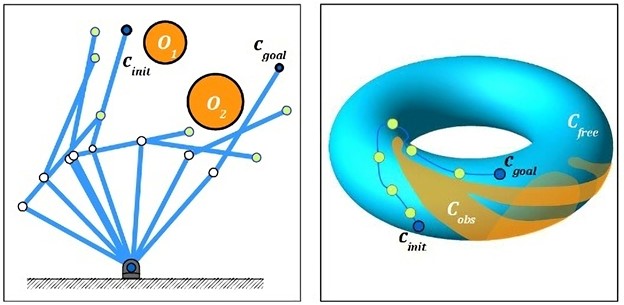


Рисунок -9. Конфигурационное пространство двузвенного манипуляционного робота. Углы поворота верхнего и нижнего звена соответствуют углу поворота образующей окружности тора и угловой координате точки на этой окружности соответственно

# 4. Классификация методов решения задач планирования движения

Методы планирования движения классифицируются по разным признакам. В контексте использования интеллектуальных технологий их можно разделить на точные и эвристические. По характеру окружающей обстановки можно разделить методы планирования на методы планирования в статической окружающей среде и в динамической среде. Методы также можно разделить по полноте информации об окружающей среде: методы с полной информацией (в таком случае говорят о глобальном планировании пути) и методы с неполной информацией (обычно речь идет о знании обстановки в непосредственной близости от робота, в этом случае речь идет о локальном планировании пути). Можно отметить методы с использованием карты окружающей среды или описания ее с помощью графа или дерева; методы на основе клеточной декомпозиции; методы потенциальных полей; оптимизационные методы; методы, основанные на интеллектуальных технологий, методы Иерархии подцели.

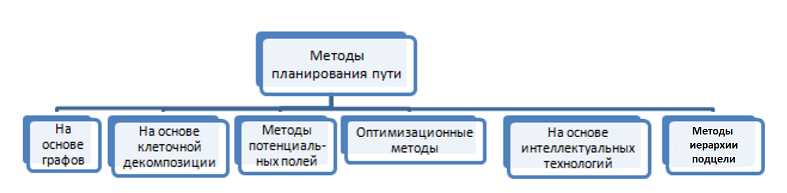


Рисунок 10. Классификация методов планирования движения

# 5. Метод Искусственных Потенциальных Полей

## 5.1. Описание метода.

Основная идея метода Искусственных Потенциальных Полей(ИПП) заключается в исследовании пространства в процессе поиска и оценка областей пространства по потенциалу, а не по контакту с препятствием. Потенциал – показатель перспективности и доступности области. В отличие от векторных алгоритмов и др., данный метод может использоваться для пространств большой размерности.

Потенциалом называют функцию , в которой каждой точке многомерного пространства соответствует число, которое является так называемым “потенциалом” или энергии точки.

Градиент в каждой точке рассматриваемого пространства является силой, которая направлена в сторону максимального возрастания . То есть каждой точке сопоставляется вектор, что превращает конфигурационное пространство в векторное поле.

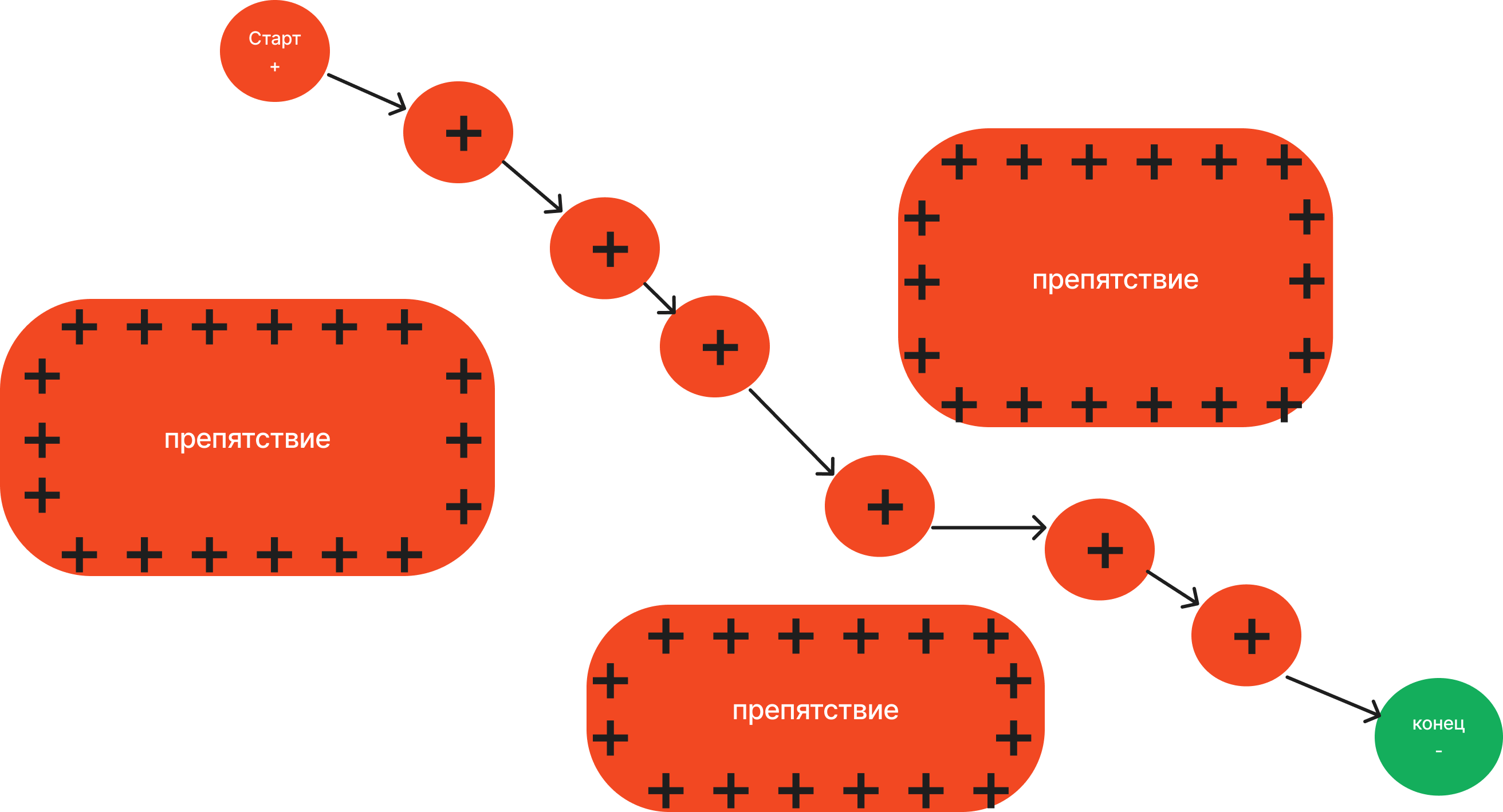


Рисунок 11. Движение точки в потенциальном пространстве

Робот рассматривается как положительно заряженная точка, которая отталкивается от препятствия и притягивается к отрицательно заряженной цели, которая является обычно концом маршрута движения робота. В каждой точке потенциального поля на робота действует комбинация сил притягивания и отталкивания, которая в итоге должна направить робота к цели, избегая препятствия. При расчете маршрута считается, что навигация робота обладает точностью, при котором погрешность движения считается близкой к нулю, в любом случае робот представлен вектором. Однако в реальных условиях роботы могут обладать меньшей точности, поэтому в различных ситуациях возможно рассчитать степень “риска”, под которым понимается близость прохождения робота с препятствиями.

## 5.2. Потенциалы притяжения и отталкивания.

Потенциал в каждой точке считается по формуле

*,* где – потенциал притяжения,

– потенциал отталкивания от *i*-го препятствия.

Суммирование происходит по всем препятствиям в потенциальном поле. Потенциальная функция монотонно возрастает по мере отдалении от цели.

Потенциал отталкивания выполняет функцию отталкивания от препятствий. Чем дальше робот от цели, тем быстрее он приближается к ней и наоборот. Функция должна возрастать по мере приближения робота к непроходимой зоне.

Функция отталкивания в каждой точке пространства может быть найдена как сумма частных потенциалов , взятая по всем препятствиям. Соответственно силу отталкивания можно найти по формуле

Для расчета оптимальной комбинированной функции отталкивания и притяжении требуется грамотно определять коэффиценты в градиентном спуске. Сила притяжения, например Манхэтеннское расстояние, должна соотноситься с силой отталкивания так, что силы отталкивания будет достаточно, чтобы исключить риск при прохождении препятствий и одновременно обеспечить проходимость между препятствий, расстояние между которыми допускается при выбранном шаге алгоритма.

## 5.4. Алгоритм Brushfire.

Для расчёта потенциала отталкивания необходимо знать расстояние от препятствий. Для точного решения данной задачи требуются огромные вычислительные мощности. Для упрощения используются следующие приёмы:

1. Учитывать расстояние до ближайшего препятствия.

2. Аппроксимировать препятствия простыми геометрическими формами.

3. Вычислять расстояния до характерных точек препятствий: барицентров, ближайших точек, критических точек и пр.

Алгоритм Brushfire, что означает пожар, предназначен для поиска расстояний до препятствий на дискретизированном пространстве. Пространство представляет из себя сетку.

На первом шаге все занятые грани и частично занятые грани получают значение 1. Также такое значение получают границы рассматриваемого поля, так как условно проходить сквозь них алгоритм не должен.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 1 | 1 |  |  |  |  |
|  |  |  |  |  | 1 | 1 |  |  |  |  |
|  |  | 1 | 1 |  |  |  |  |  |  |  |
|  |  | 1 | 1 |  |  |  |  | 1 | 1 |  |
|  |  |  |  |  |  |  | 1 | 1 | 1 |  |
|  |  |  |  |  |  | 1 | 1 | 1 | 1 |  |
|  |  |  |  |  |  |  | 1 | 1 | 1 |  |
|  |  |  |  |  |  |  | 1 | 1 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

Шаг 1

На втором шаге все пустые ячейки, находящиеся рядом с номером 1, получают номер 2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 |  |  |  |  | 2 | 2 |  |  |  | 2 |
| 2 |  |  |  | 2 | 1 | 1 | 2 |  |  | 2 |
| 2 |  | 2 | 2 | 2 | 1 | 1 | 2 |  |  | 2 |
| 2 | 2 | 1 | 1 | 2 | 2 | 2 |  | 2 | 2 | 2 |
| 2 | 2 | 1 | 1 | 2 |  |  | 2 | 1 | 1 | 2 |
| 2 |  | 2 | 2 |  |  | 2 | 1 | 1 | 1 | 2 |
| 2 |  |  |  |  | 2 | 1 | 1 | 1 | 1 | 2 |
| 2 |  |  |  |  |  | 2 | 1 | 1 | 1 | 2 |
| 2 |  |  |  |  |  | 2 | 1 | 1 | 2 | 2 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

Шаг 2

На третьем шаге и т.д. ячейки получают значение 3 и т.д.

Стоит заметить, что при малой размерности сетки, значение соседей определяется по принципу: боковые стороны без диагоналей.

В стол малом пространстве достаточно важно точно определять расстояние, из-за высокого риска столкновения. В более обширном пространстве, шаг сетки гораздо меньше, что позволяет с большей точностью определять положении препятствий. Можно руководствоваться принципом: боковые стороны и диагонали.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 3 | 3 | 3 | 3 | 2 | 2 | 3 | 3 | 3 | 2 |
| 2 | 3 | 4 | 3 | 2 | 1 | 1 | 2 | 3 | 3 | 2 |
| 2 | 3 | 2 | 2 | 2 | 1 | 1 | 2 | 3 | 3 | 2 |
| 2 | 2 | 1 | 1 | 2 | 2 | 2 | 3 | 2 | 2 | 2 |
| 2 | 2 | 1 | 1 | 2 | 3 | 3 | 2 | 1 | 1 | 2 |
| 2 | 3 | 2 | 2 | 3 | 3 | 2 | 1 | 1 | 1 | 2 |
| 2 | 3 | 3 | 3 | 3 | 2 | 1 | 1 | 1 | 1 | 2 |
| 2 | 3 | 4 | 4 | 4 | 3 | 2 | 1 | 1 | 1 | 2 |
| 2 | 3 | 3 | 3 | 3 | 3 | 2 | 1 | 1 | 2 | 2 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

Шаг 3,4

В результате заполненной сетки по номеру ячейки можно легко определить манхетеннское и евклидовое расстояние до ближайшего препятствия.

## 5.4. Градиентный спуск.

Чтобы избежать хаотического метания в процессе поиска, в результате работы некоторых функций отталкивания необходимо воспользоваться известным методом локальной оптимизации.

В каждой точке конфигурационного пространства определяется градиент потенциального поля и делается маленький шаг в противоположную сторону. Операция продолжается до тех пор, пока не будет достигнута точка с нулевым градиентом.

**Пример**

Потенциалы притягивания и отталкивания будем подсчитывать по следующим формулам:

– манхеттенское расстояние между финальной и текущей точкой.

*,* где *D* – расстояние до ближайшего препятствия.

выбирается в данном случае из соображения НОК, = 1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  | t |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 19 |  |  |  |  |  |
|  |  |  |  | 21 | 20 |  |  |  |  |  |
|  |  | 27 | 22 | 27 |  |  |  |  |  |  |
|  | 29 | 24 | 23 | 26 |  |  |  |  |  |  |
| 43 | 30 | 29 | 24 | 25 |  |  |  |  |  |  |
| S | 43 | 42 |  |  |  |  |  |  |  |  |

Шаги алгоритма градиентного спуска

## 5.5. Алгоритм поиска наилучшего.

Алгоритм поиска наилучшего является одним из самых простых способов поиска локального минимума. Предполагается, что свободная область конфигурационного пространства дискретизирована и для каждой ячейки получен потенциал.

1. В процессе решения создаётся дерево поиска T, которое на первом шаге состоит из одной стартовой вершины.
2. Среди листьев выбирается значение с минимальным значениям потенциала U.
3. Ищутся все соседи S(p)
4. Эти ячейки становятся потомками вершины p в дереве поиска T
5. Поиск продолжается, пока вершина t не попадёт в окрестностей какого-нибудь из листьев.

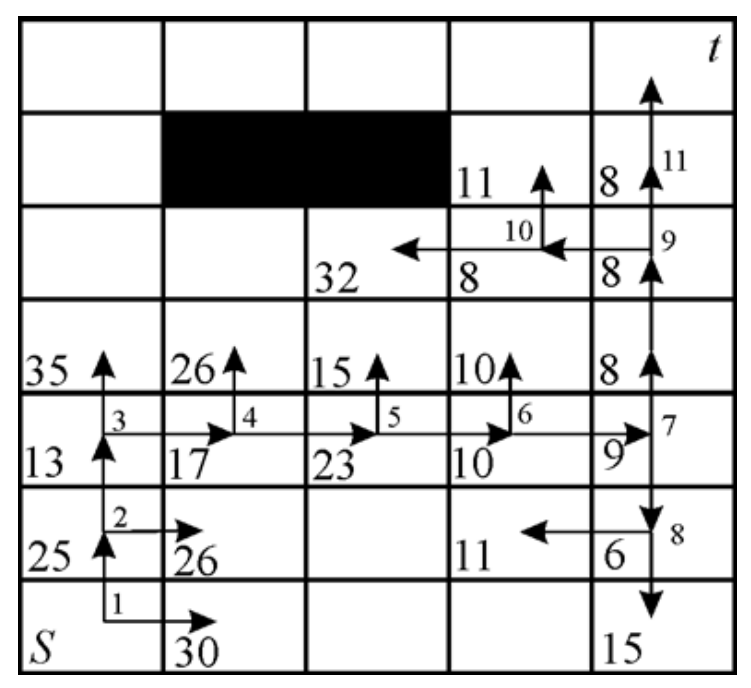


Рисунок 12. Пример работы алгоритма BFS.

# 6. Программная реализация.

## 6.1. Описание среды программирования PYTHON.

Программа реализации метода Искусственного потенциального поля выполнена на языке PYTHON.   
Преимущество данного языка заключается в возможности использования библиотек, содержащие математические алгоритмы, что значительно упрощает выполнение поставленной задачи.

Библиотеки, использованные в коде: NumPy и Matplotlib.

NumPy можно рассматривать как свободную альтернативу MATLAB. Язык программирования MATLAB внешне напоминает NumPy: оба они интерпретируемые, оба позволяют выполнять операции над массивами (матрицами), а не над скалярами.

Преимущество MATLAB в наличии большого количества пакетов («тулбоксов»), например, Simulink[en]. Для NumPy тоже существуют подобные «пакеты», например, библиотека SciPy предоставляет больше MATLAB-подобной функциональности, библиотека Matplotlib позволяет создавать графики в стиле MATLAB. И MATLAB, и NumPy для решения основных задач линейной алгебры используют код, основанный на коде библиотеки LAPACK.

Matplotlib является гибким, легко конфигурируемым пакетом, который вместе с [NumPy](https://ru.wikipedia.org/wiki/NumPy" \o "NumPy), [SciPy](https://ru.wikipedia.org/wiki/SciPy" \o "SciPy) и [IPython](https://ru.wikipedia.org/wiki/IPython" \o "IPython) предоставляет возможности, подобные MATLAB. В настоящее время пакет работает с несколькими графическими библиотеками, включая [wxWindows](https://ru.wikipedia.org/wiki/WxWindows" \o "WxWindows) и [PyGTK](https://ru.wikipedia.org/wiki/PyGTK" \o "PyGTK).

Пакет поддерживает многие виды графиков и [диаграмм](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0):

* Графики ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *line plot*)
* [Диаграммы рассеяния](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D1%80%D0%B0%D1%81%D1%81%D0%B5%D1%8F%D0%BD%D0%B8%D1%8F) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *scatter plot*)
* [Столбчатые диаграммы](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D0%BB%D0%B1%D1%87%D0%B0%D1%82%D0%B0%D1%8F_%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *bar chart*) и гистограммы ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *histogram*)
* [Круговые диаграммы](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D1%83%D0%B3%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *pie chart*)
* [Диаграммы стебель-листья](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D1%81%D1%82%D0%B5%D0%B1%D0%B5%D0%BB%D1%8C-%D0%BB%D0%B8%D1%81%D1%82%D1%8C%D1%8F) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *stem plot*)
* Контурные графики ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *contour plot*)
* Поля градиентов ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *quiver*)
* [Спектральные диаграммы](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *spectrogram*)

Пользователь может указать оси координат, решетку, добавить надписи и пояснения, использовать [логарифмическую шкалу](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%88%D0%BA%D0%B0%D0%BB%D0%B0) или [полярные координаты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D1%8B).



Рисунок 13

## 6.2. Описание работы программы

Данный код представляет собой реализацию алгоритма планирования движения для мобильного робота, основанного на методе искусственных потенциальны полей.

Для начала определяется файл формата json, построенный в результате работы программы generator.exe

Структура программы определяется в файле ***main.py***

В нём объявляются функции и задаются константы. Предусмотрена параметризация дискретизации сетки потенциального поля. Возможно задать шаг.

Инициализируется поле ***field*** класса ***Field***.

Вызывается функция ***field.brush\_fire(field.get\_barriers())***(см. Листинг), где функция ***field.get\_barriers())***, расставляет границы и препятствия, реализуя 1 шаг алгоритм Brushfire. Функция ***field.brush\_fire*** принимает на вход массив “единиц”, реализует алгоритм Brushfire до полного заполнения поля.

Затем вызывается функция ***field.gradient\_descent()***(см. Листинг), реализующая метод градиентного спуска.

Далее вызывается функция ***field.best\_first\_search()***(см. Листинг), реализующая алгоритм быстрого поиска по дереву. Воспользовавшись возможностями библиотеки ***treelib*** возможно вернуть все точки дерева в функцию ***self.\_find\_min\_dist()***(см. Листинг).

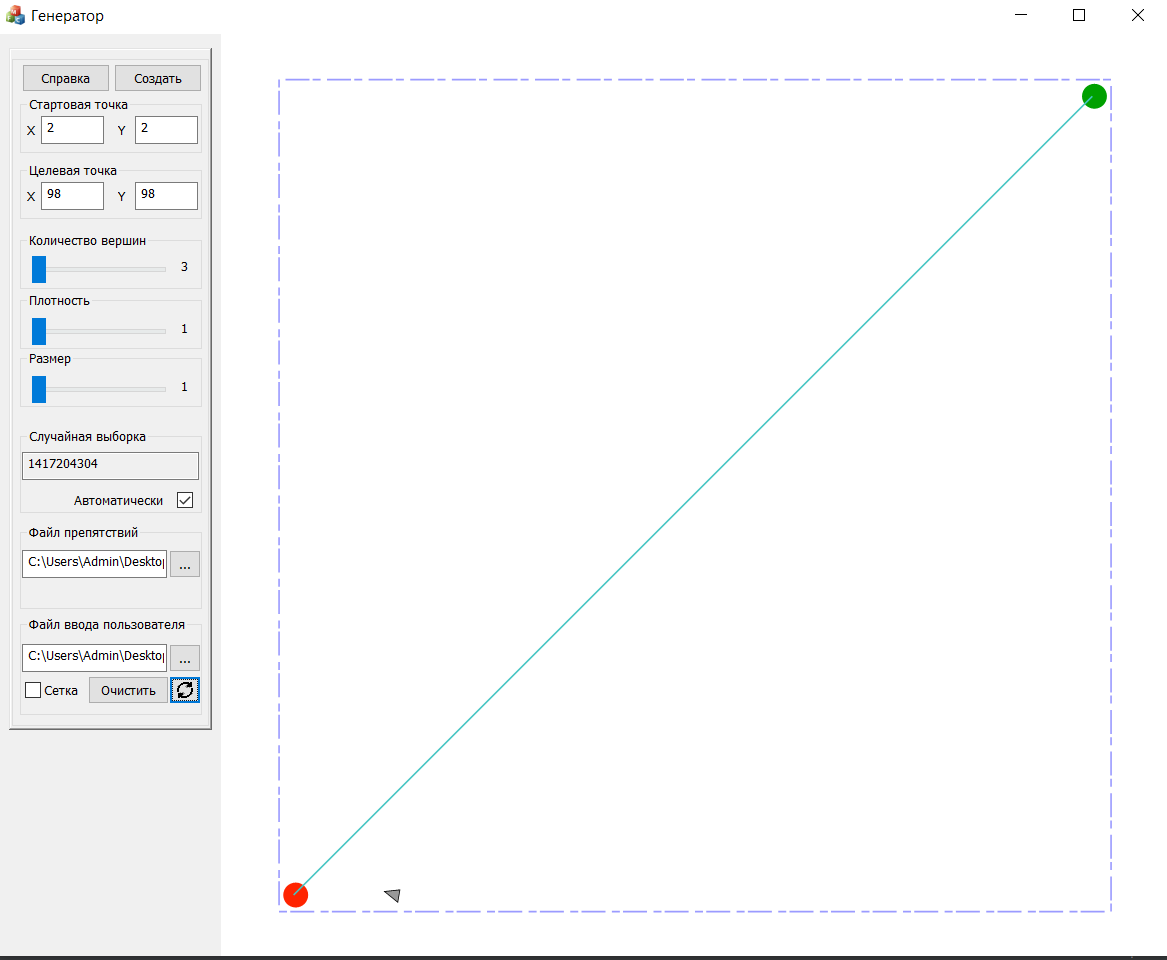
***self.\_find\_min\_dist()*** (см. Листинг), реализует алгоритм поиска минимального пути по дереву. Исходный путь передаётся в формат json и запускается в программе генератора.

## 6.3. Результаты работы программы.

Для тестирования эффективности и работоспособности алгоритма, необходимо провести тесты с различной плотностью, величиной и размером препятствий.

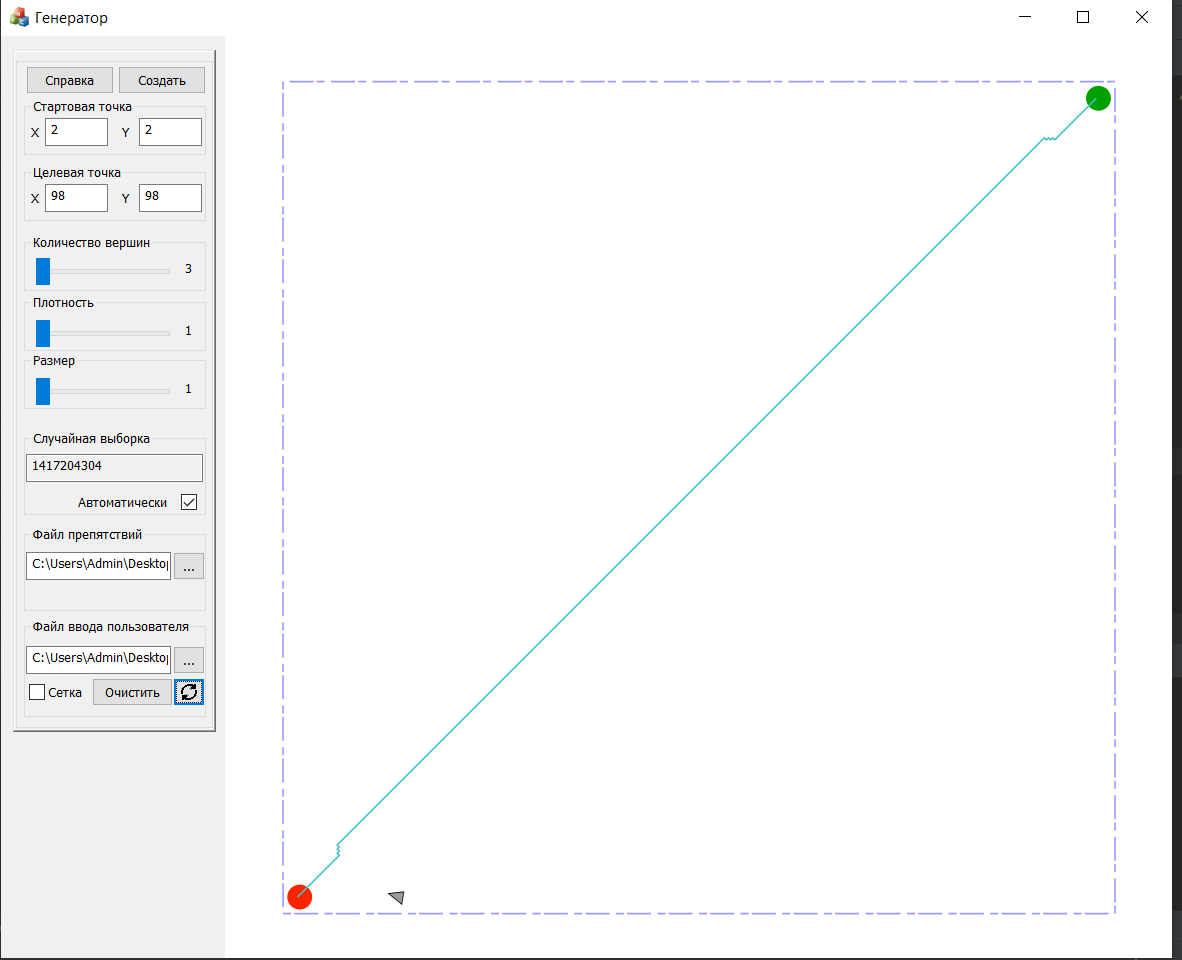
Для начала проверим работу алгоритма на параметрах:

количество вершин – 3, плотность – 1, размер – 1.



Время работы 4.91 сек

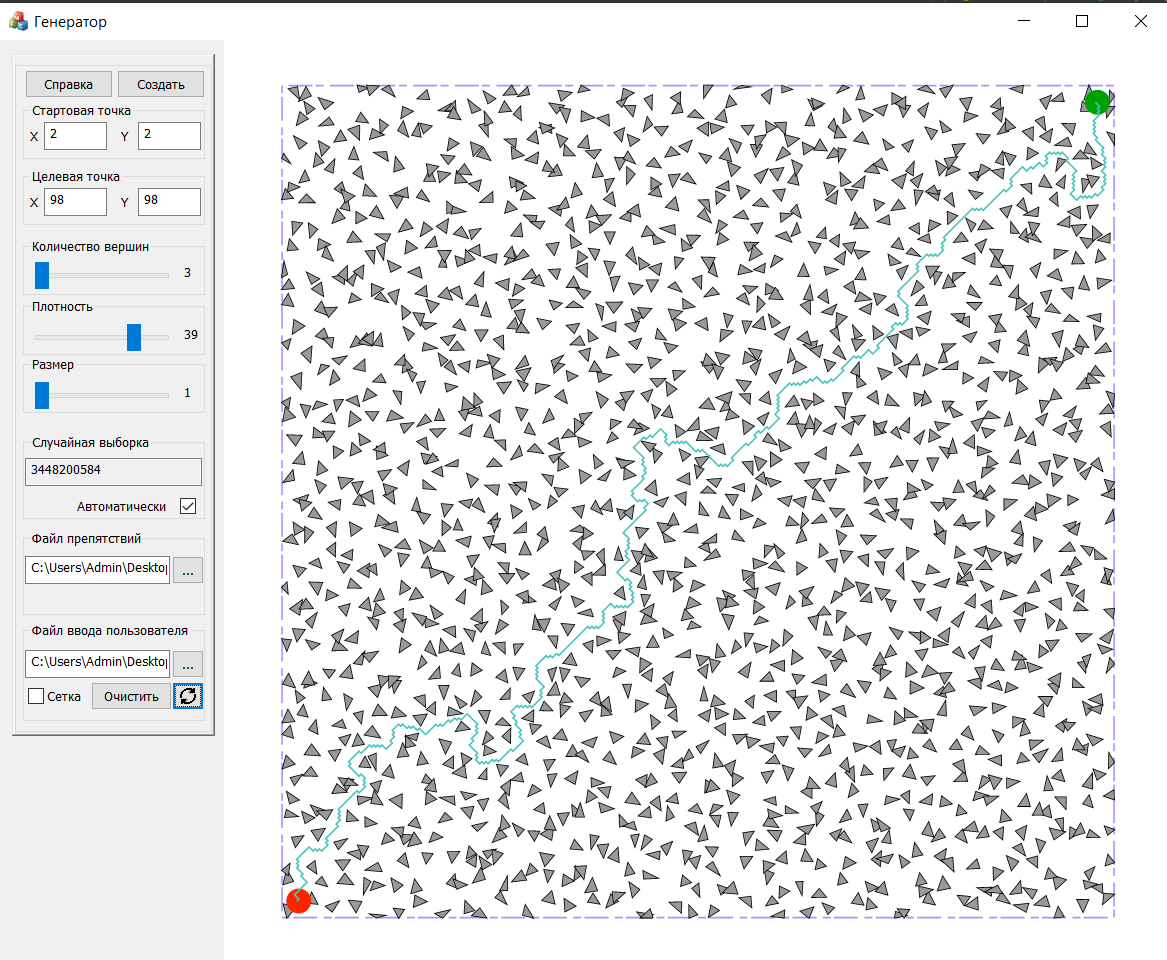
Заметим, что сила отталкивания препятствия не сильно влияет на выбор пути. Это связано со слабым влиянием коэффицента β в комбинированной формуле потенциала точек поля. Попробуем увеличить коэффицент в 100 раз



Время работы 59.99 сек

Как мы видим, при увеличении коэффицента β, зависимость траектории от силы отталкивания препятствия стало более заметно, однако время работы увеличилось почти в 12 раз, что является доказательством важности подбора коэффицентов уравнений потенциалов.

Рассмотрим пример с повышенной плотностью препятствий



Время работы 20.91 сек

Как мы видим, при определенном коэффиценте β, алгоритм позволяет построить маршрут сквозь большое количество препятствий, находящихся на маленьком расстоянии друг от друга.

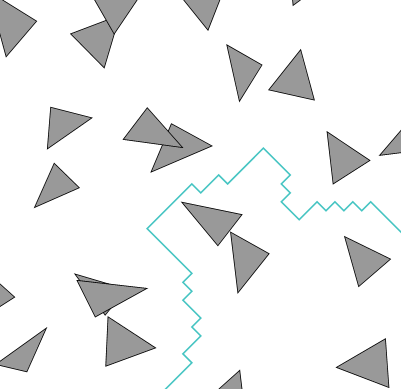
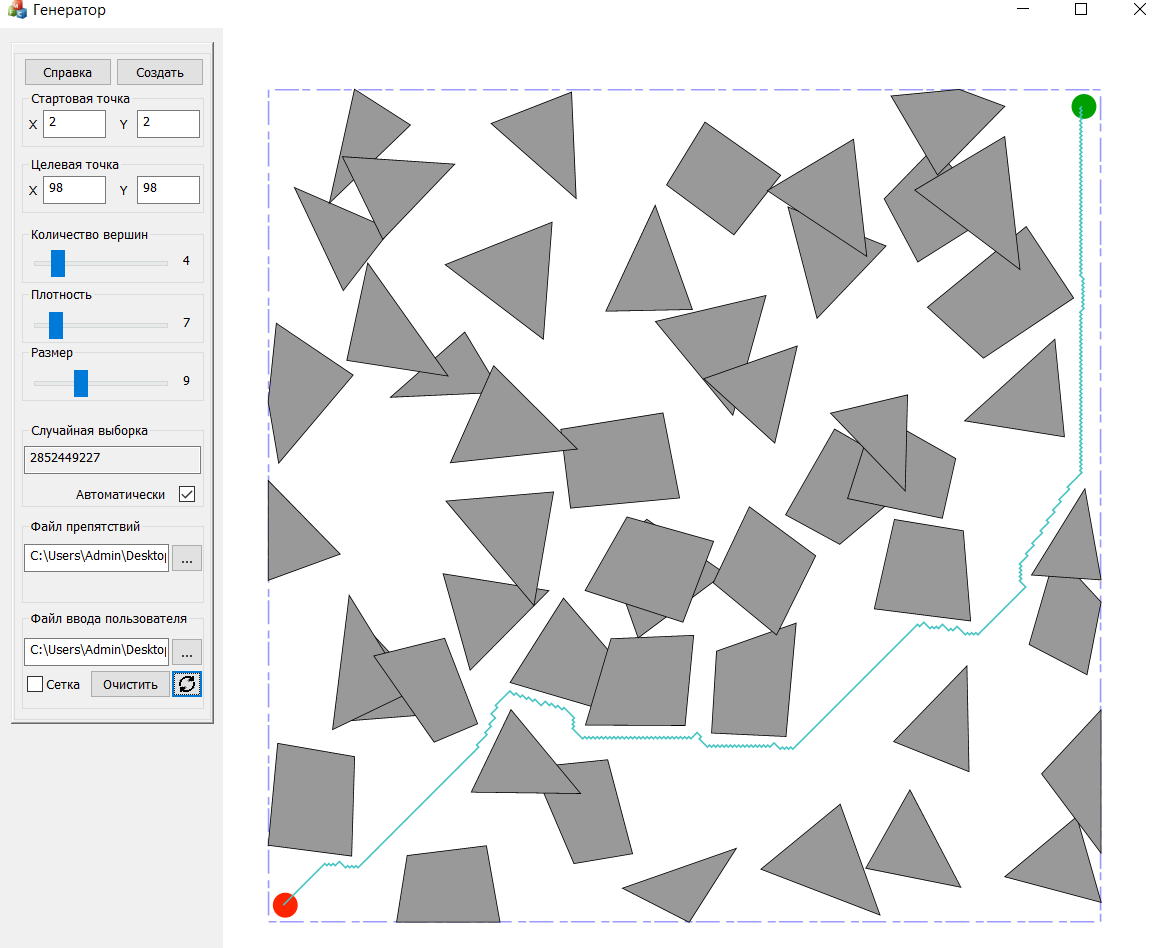


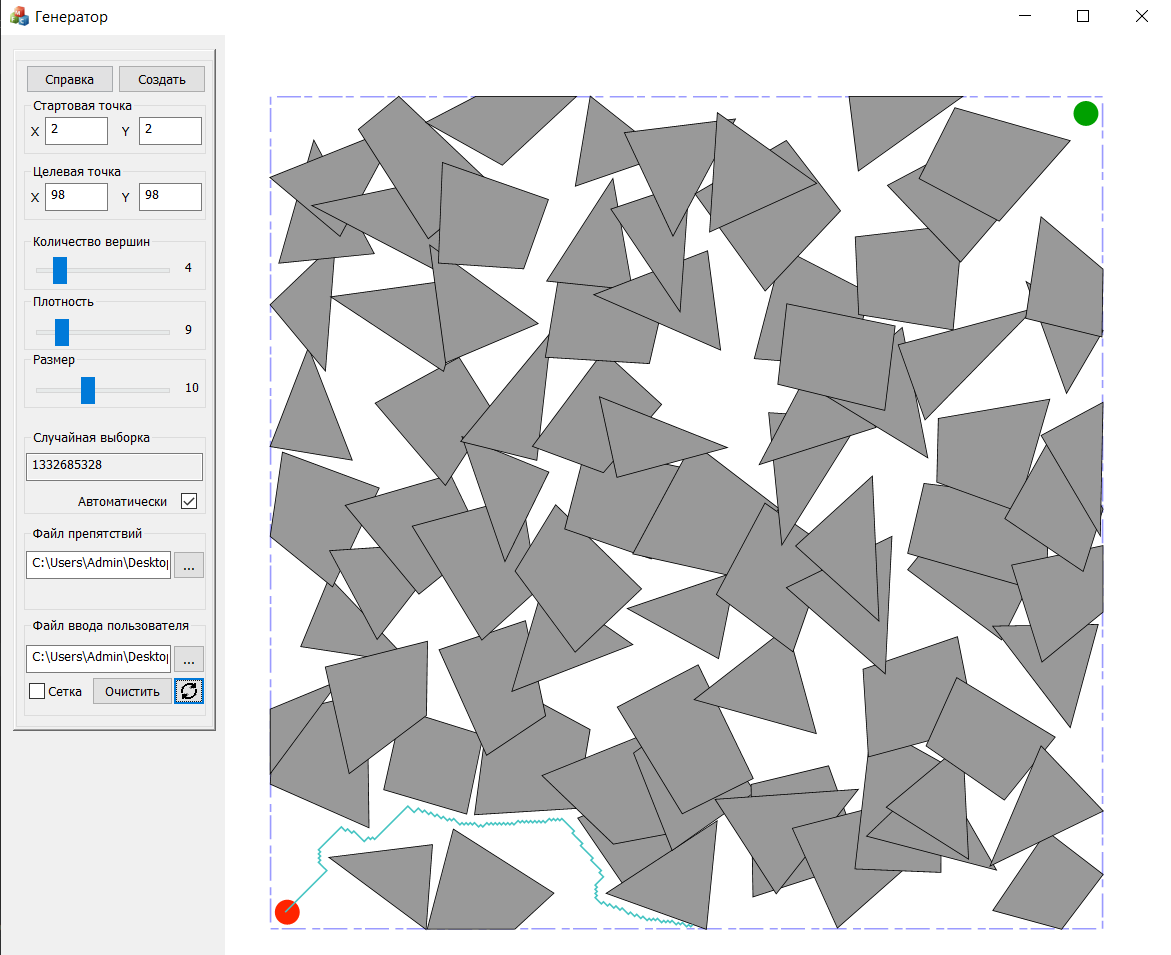
Рисунок 14

Проверим на другом примере работу алгоритма:



Время работы 20.08 сек

Проведём проверку аварийной остановки алгоритма



Время работы алгоритма 51.54 сек

Для случаев, когда проход через препятствия невозможен, предусмотрено ограничение по количеству итерации в дереве поиска.

Ограничение по количеству итерации подобрано опытном путём и предполагает собой решений временной эффективности работы алгоритма в случае невозможности прохождения пути.

# Заключение

Метод Искусственных Потенциальных Полей является эффективным алгоритмом поиска пути. В результате реализации алгоритма на языке Python удалось оптимизировать работу алгоритма в различных средах препятствий, была изучена зависимость соотношений потенциальных уравнений притяжения и отталкивания.

# Список литературы

[l] Aho, A., Hopcroft, J., and Ullman, J. The Design and Analysis of Computer

Algorithms, AddisonWesley, 1974.

[2] Barraquand, J. and Latombe, J., “A Monte-Carlo

algorithm for path planning with many degrees of

freedom,” Proc. of IEEE Int. Con$ on Robotics

and Automation, pp. 1712-1717, 1990.

[3] Branicky, M. S. and Newman, W. S., “Rapid

computation of configuration obstacles,” Proc. of

IEEE Int. Con$ on Robotics and Automation,

pp. 304310, 1990.

[4] Canny, J. F. and Lin, M. C, “An Opportunistic

Global Path Planner,” Proc. of IEEE Int. Conf. on Robotics and Automation, pp.

15541559, 1990.

[5] Canny, J. F., The Complexity of Robot Motion

Planning, MIT Press, 1988.

[6] Faverjon, B. and Tournassoud,, P., “A local approach for path planning of

manipulators with a

high number of degrees of freedom,” Proc. of IEEE

Int. Conf. on Robotics and Automation, pp. 1152-

1159,1987.

[7] Gilbert, E. G., Johnson, D. W. and Keerthi, S. S.,

A Fast Procedure for Computing the Distance

Between Complex Objects in Three-Dimensional

Space,” IEEE Journal of Robotics and Automation, vol. 4, no. 2, pp. 193-203, April

1988.

[8] Hasegawa, T. and Terasaki, H., “Collision avoid- ance: divide-and-conquer

approach by space characterization and intermediate goals,” IEEE Transactions on

Systems, Man, and Cybernetics, vol. 18, no. 3, pp. 337-347, May/June 1988.

[9] Kondo, K., “Motion Planning with Six Degrees of

Freedom by Multistrategic Bidirectional Heuristic

Free-%ace Enumeration”. IEEE Dansactions on Robot& and Automation; vol. 7, no.

3, pp. 267-

277, June 1991.

[10] Lozano-PBrez, T., “A Simple Motion-Planning

Algorithm for General Robot Manipulators,”

IEEE Journal of Robotics and Automation,

vol. RA-3, no. 3, pp. 224-238, June 1987.

[11] Paden, B., Mees, A. and Fisher, M., “Path Planning Using a Jacobian-Based

Freespace Generation

Algorithm,” Proc. of IEEE Int. Conf. on Robotics

and Automation, pp. 1732-1737, 1989.

[12] Reif, J. H., “Complexity of the Mover’s Problem

and Generalizations,” Proc. of 20th IEEE Symposium on Foundations of Computer

Science, pp.

[13] Schwartz, J. T. and Sharir, M., “On the

Piano Movers’ Problem: I. The Case of a Two-Dimensional Rigid Polygonal Body

Moving

Amidst Polygonal Barriers,” Communications on Pure and Applied Mathematics,

vol. 34, pp. 345-

398, 1983.

[14] Tendick, F., Voichick, F., Tharp, G. and Stark,

L., “A Supervisory Telerobotic Control System Using Model-Based Vision

Feedback.” Proc. of IEEE

421-427, 1979.

Int. Conf. on Robotics and Automation, p;. 2280-

[15] Антонов В.О., Гурчинский М.М., Петренко В.И., Тебуева Ф.Б. МЕТОД

ПЛАНИРОВАНИЯ ОПТИМАЛЬНОЙ ТРАЕКТОРИИ ДВИЖЕНИЯ

ТРЕХЗВЕННОГО МАНИПУЛЯТОРА В ОБЪЕМНОМ ПРОСТРАНСТВЕ С

ПРЕПЯТСТВИЕМ. Вестник Дагестанского государственного технического

университета. Технические науки. 2018;45(1):98-112.

https://doi.org/10.21822/2073-6185-2018-45-1-98-112

[16] Б.С.Э. С. М. Тарг.

[17] Математика и математическое моделирование.

2018. № 01. С. 15–58 Лю В. “Методы планирования пути в среде с

препятствиями”

# Листинг

Алгоритм заполнения

def get\_barriers(self):  
 *"""Расстановка препятствий на поле."""* with open('ATP/sample\_output\_file1.json') as f:  
 barriers = json.load(f)  
  
 return\_list = []  
 # Словарь списков областей каждого препятствия  
 list\_of\_ones = {str(i): [] for i in range(len(barriers)-3)}  
 for i\_pol, polygon in enumerate(barriers[3:]):  
 points\_list = []  
  
 # Создание списка координат вершин фигуры  
 for point in polygon['points']:  
 x = point['x']  
 y = point['y']  
  
 if x == 100.0:  
 x = 100.0 - 0.001  
 #  
 if y == 100.0:  
 y = 100.0 - 0.001  
  
 points\_list.append([x, y])  
 points\_list = np.array(points\_list)  
  
 # Расстановка вершин фигуры в поле  
 for point in points\_list:  
 x = self.\_stepping(point[0])  
 y = self.\_stepping(point[1])  
  
 list\_of\_ones[str(i\_pol)].append((y, x))  
  
 # Прохождение по всем парам вершин фигуры (граням)  
 # для обозначения вертикальных и наклонных граней  
 for i in range(points\_list.shape[0]):  
 j = i + 1 - (i // (points\_list.shape[0] - 1)) \* points\_list.shape[0]  
  
 y1, y2 = points\_list[i][1], points\_list[j][1]  
 x1, x2 = points\_list[i][0], points\_list[j][0]  
  
 # Прохождение по каждой точке на прямой с шагом  
 for y in np.arange(self.\_stepping(min((y1, y2))) + self.step, self.\_stepping(max((y1, y2))), self.step):  
 x = (y - y1) \* (x2 - x1) / (y2 - y1) + x1  
 x = self.\_stepping(x)  
  
 list\_of\_ones[str(i\_pol)].append((y, x))  
  
 # Заполнение горизонтальных линий фигур  
 for y in np.arange(self.\_stepping(np.min(points\_list[:, 1])),  
 self.\_stepping(np.max(points\_list[:, 1])), self.step):  
 counter = 0  
 cur\_line = []  
  
 # Читаем кол-во границ на горизонтальной  
 for yx in list\_of\_ones[str(i\_pol)]:  
 if yx[0] == y:  
 #берём нулевой это игрик как раз  
 counter += 1  
 #счётчик добавочных ячеек на текущем игрике  
 cur\_line.append(yx[1])  
 #добавляем в текущую строку x уже найденное на границе  
  
 # Если есть границы, заполняем пространство между ними  
 if counter >= 2:  
 for x in np.arange(min(cur\_line), max(cur\_line), self.step):  
 if (y, x) not in list\_of\_ones[str(i\_pol)]:  
 list\_of\_ones[str(i\_pol)].append((y, x))  
  
 # Наносим все по-отдельности фигуры на поле  
 for point in list\_of\_ones.keys():  
 for one in list\_of\_ones[point]:  
 x = int(one[1] / self.step)  
 y = int(one[0] / self.step)  
  
 self.field[self.size - y - 1][x].distance = 1  
 return\_list.append((self.size - y - 1, x))  
  
 return return\_list

Алгоритм Brushfire

def brush\_fire(self, list\_of\_ones):  
 *"""Алгоритм Brushfire. Нахождение расстояния до ближайшей преграды дял каждой точки."""* borders = list\_of\_ones  
 # Расстановка расстояний для границ поля  
 for y in range(self.size):  
 for x in range(self.size):  
  
 if x in (0, self.size-1) or y in (0, self.size-1):  
 self.field[y][x].distance = 1  
 borders.append((x, y))  
  
 length = 1  
 while True:  
 length += 1  
  
 new\_borders = []  
  
 for xy in borders:  
 cell = self.field[xy[0]][xy[1]]  
  
 for neigh in self.\_get\_neighbours(cell):  
 if neigh.distance == 0:  
 neigh.distance = length  
 new\_borders.append((neigh.y, neigh.x))  
  
 if len(new\_borders) == 0:  
 break  
  
 borders = new\_borders

Градиентный спуск

def gradient\_descent(self):  
 *"""Градиентный спуск для нахождения 'потенциала' каждой точки."""* self.\_get\_beta()  
  
 # Расчет сил притяжения для каждой точки  
 for y in range(self.size):  
 for x in range(self.size):  
 cell = self.field[y][x]  
 # cell.capability = int(self.beta / abs(0.5 - cell.distance))  
 # cell.capability = 100000 \* pow((1 / cell.distance), 2)  
 cell.capability = 100 \* pow((1 / (0.25 - cell.distance)), 2)  
 # cell.capability = abs((1 / (0.99 - cell .distance)))  
 # cell.capability = exp(25 \* (1 / cell.distance))  
 cell.capability += abs(self.end[0] - x) + abs(self.end[1] - y)

Алгоритм BFS

def best\_first\_search(self):  
 *"""Алгоритм поиска от наилучшего. Поиск пути до конечной точки."""* start = self.field[self.start[1]][self.start[0]]  
  
 # Клетки принадлежащие фигурам  
 skipped = set()  
 # Дерево пути  
 all\_tops = Tree()  
 # Начальная точка  
 all\_tops.create\_node(identifier=str((start.x, start.y)))  
  
 neighs = self.\_get\_neighbours(start)  
 for neigh in neighs:  
 try:  
 all\_tops.create\_node(identifier=str((neigh.x, neigh.y)), parent=str((start.x, start.y)))  
 except:  
 pass  
  
 quite = False  
 for i in range(self.iters):  
 # while True:  
 # if i == 10:  
 # all\_tops.show()  
 tops = set([self.field[eval(nod.tag)[1]][eval(nod.tag)[0]] for nod in all\_tops.leaves()])  
  
 #из идентефикатора "листа" возвращает кортеж x y  
  
 top = self.\_find\_min(tops - skipped)  
 #исключаем препятствия и находим минимум в текущей итерации ветки  
 # Проверка на отсутствие пути  
 if isinstance(top, float):  
 print("No path!")  
 break  
  
 cnt = 0  
 neighbours = self.\_get\_neighbours(top)  
 for neigh in neighbours:  
 try:  
 all\_tops.create\_node(identifier=str((neigh.x, neigh.y)), parent=str((top.x, top.y)))  
 except:  
 cnt += 1  
  
 # Условие достижения финальной точки  
 if neigh.x == self.end[0] and neigh.y == self.end[1]:  
 quite = True  
 if quite:  
 break  
 # Добавление точки принадлежащей фигуре  
 if cnt == len(neighbours):  
 skipped.add(top)  
  
 # all\_tops.show()  
 xy = self.\_find\_min\_dist(all\_tops.all\_nodes())  
 #возвращаем все точки дерева и передаем функции для нахождения min dist  
 print("End point:", xy)  
  
 self.\_create\_way(list(all\_tops.rsearch(str(xy))))

Нахождение минимальной дистанции

def \_find\_min\_dist(self, leaves):  
 *"""Нахождение точки с минимальным расстоянием до конечной точки."""* dist = lambda x\_, y\_: abs(self.end[0] - x\_) + abs(self.end[1] - y\_)  
 min\_dist = self.size \*\* 2  
 xy = (1, 198)  
  
 for leaf in leaves:  
 x = eval(leaf.tag)[0]  
 y = eval(leaf.tag)[1]  
 if dist(x, y) <= min\_dist:  
 min\_dist = dist(x, y)  
 xy = (x, y)  
  
 return xy