Методы построения и анализа алгоритмов

Расчетно-графическое задание:

Построение пути для мобильного робота в двумерном пространстве

Задача

Мобильный робот движется по плоскости. Робот имеет форму круга радиусом r с координатой центра (C_x, C_y); робот может двигаться в любом направлении без ограничений, с некоторой заданной постоянной скоростью. Также на плоскости находятся объекты-препятствия. Заданы начальная и конечная координаты робота Init=($Init_x$, $Init_y$) и Goal=($Goal_x$, $Goal_y$), дополнительно может быть задано несколько промежуточных координат.

Задача: найти кратчайший (по возможности) путь для робота из начальной координаты в конечную, не приводящий к столкновению робота с препятствиями. При наличии дополнительных координат путь должен проходить через каждую дополнительную координату минимум по одному разу.

Основные определения

Препятствия в задании могут задаваться одним из следующих образов:

- параллельные осям прямоугольники (Рис. 1a)
- круги (Рис. 1b)
- треугольники (Рис. 1c)

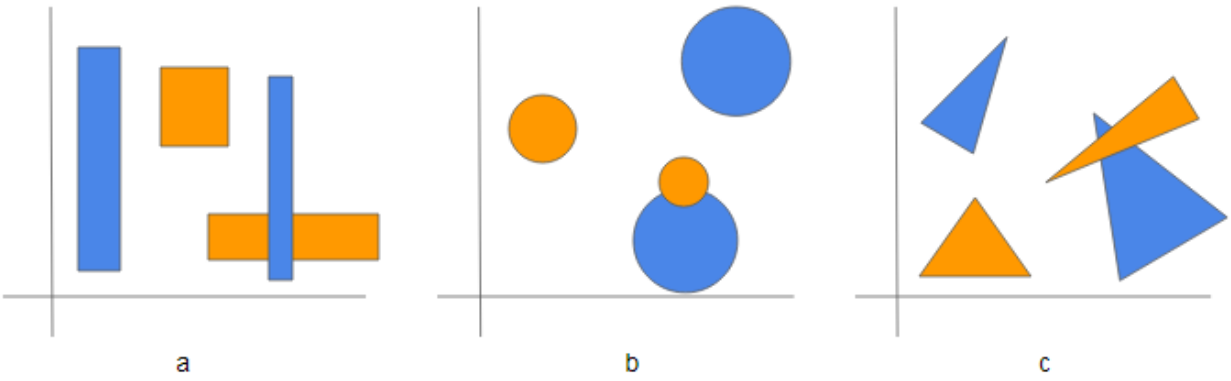

Рис. 1. Типы препятствий

Также считаем, что рабочая область ограничена прямоугольной границей, выход за которую не желателен.

Робот задается как круг радиусом $r \in R$, положение робота определяется его координатой в двумерном пространстве - положением центра круга $(C_x, C_y) \in R^2$.

Путь - последовательность координат $P_i=(P_{ix}, P_{iy}) \in R^2$, i=1..n, P_1=Init, P_n=Goal. Координаты в пути соединяются отрезками, т.е. между каждой парой координат робот движется по прямой.

Пространство конфигураций

Конфигурация робота - набор параметров, однозначно определяющий положение робота в пространстве. Пространство конфигураций (configuration space, C) - множество всех возможных конфигураций робота. В нашей задаче конфигурация робота - его координата, соответственно имеем двумерное пространство конфигураций.

Пространство конфигураций C можно разбить на два подпространства (Рис. 2): C_{free} - допустимые конфигурации, т.е. не приводящие к столкновениям с препятствиями, и C_{obs} - недопустимые конфигурации, т.е. приводящие к столкновениям с препятствиями.

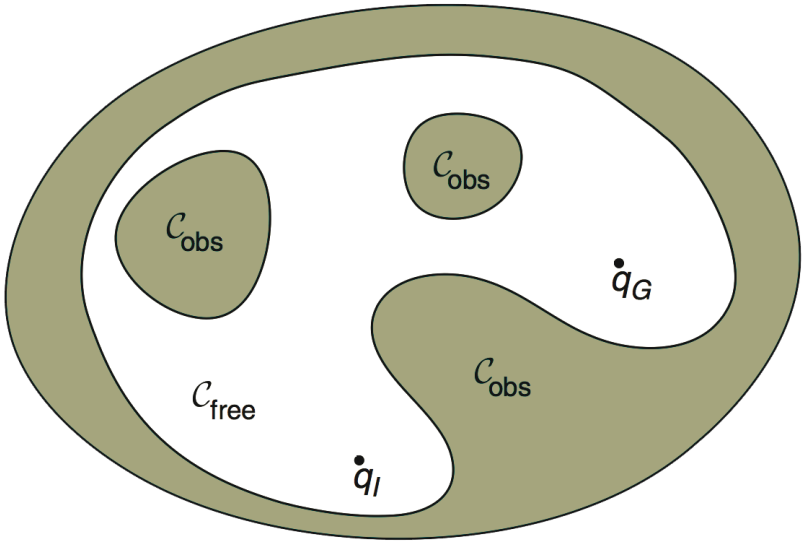

Рис. 2. Подпространства C_{free} и C_{obs}

В пространстве конфигураций конфигурация робота - **материальная точка**, соответственно задача поиска (кратчайшего) пути сводится к задаче поиска в C (кратчайшей) кривой/ломаной прямой, полностью лежащей в C_{free} и соединяющей начальную и конечную конфигурации q_{Init} и q_{Goal} (Рис. 3).

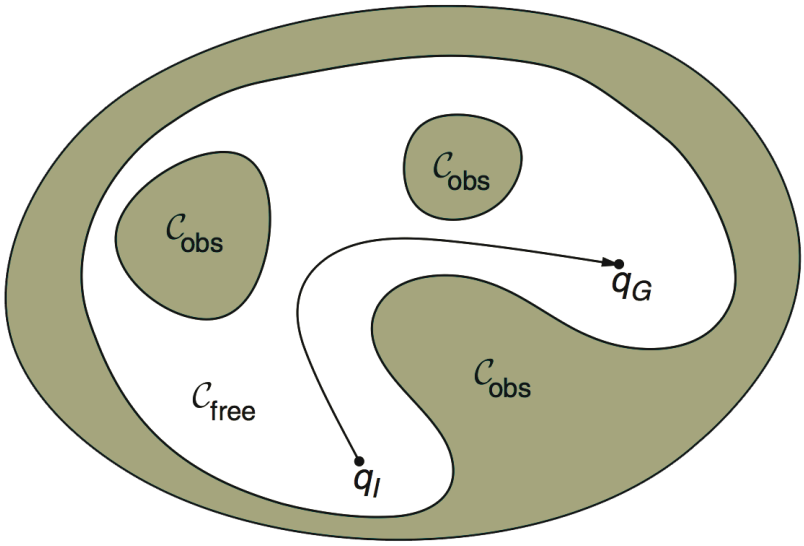

Рис. 3. Путь робота в C_{free}

Различные алгоритмы поиска пути используют явное или неявное представление C_{obs}. Для получения C_{obs} в явном виде фигуры-препятствия “увеличиваются” по краям на форму робота, т.е. препятствие в C получается путем прохождения формы робота по краю исходной фигуры и объединению полученной фигуры с исходной (Рис. 4).

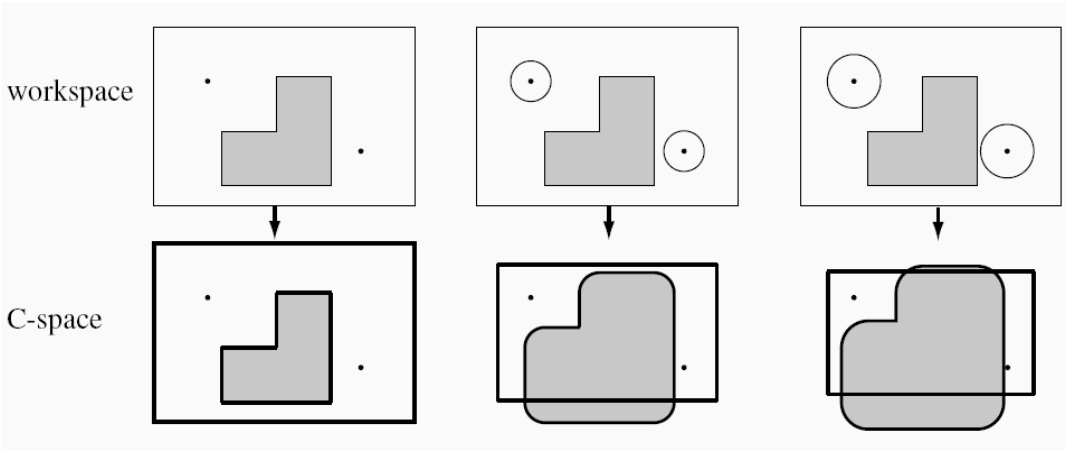

Рис. 4. Получение C_{obs} в явном виде

Для простоты введем следующие упрощенные правила получения фигур-препятствий в C_{obs}, не меняющие тип фигуры (здесь r - или радиус робота, или величина немного больше радиуса робота, если например нежелательно слишком близкое сближение с препятствиями):

- прямоугольники: каждый прямоугольник увеличивается на r в каждую сторону по каждой координате
- круги: радиус каждого круга увеличивается на r
- треугольники: треугольник в C_{obs} получается из исходного треугольника путем сдвига каждой его стороны в направлении вектора перпендикулярного к стороне (направление - “из” треугольника), на величину r, и нахождении точек пересечения для прямых, проходящих через сдвинутые стороны

Алгоритмы построения пути

Все из рассматриваемых алгоритмов построения пути так или иначе основываются на дискретизации пространства конфигураций и его дальнейшем исследовании, после чего задача поиска пути сводится к задаче поиска кратчайшего пути в взвешенном графе. Для поиска кратчайшего пути в графе могут использоваться такие алгоритмы, как алгоритм Дейкстры, A*, D*, Theta* и т.д..

При наличии в задаче промежуточных координат в явном или неявном виде добавляется задача коммивояжера. Ввиду большой вычислительной сложности точного решения задачи, ее можно решать эвристическими методами: например, жадными или генетическими алгоритмами.

Комбинаторные методы

Граф видимости (Visibility graph)

В C строится граф, вершинами графа являются все вершины полигонов-препятствий из C_{obs} плюс начальная и конечная конфигурации (Рис. 5). Пара вершин соединяется ребром (отрезком), если они “видны” друг из друга - т.е. отрезок не содержит пересечений с C_{obs} (при этом границы полигонов включаются в граф), вес ребра - длина пути. Для полученного графа решается задача поиска кратчайшего пути из начальной конфигурации в конечную.

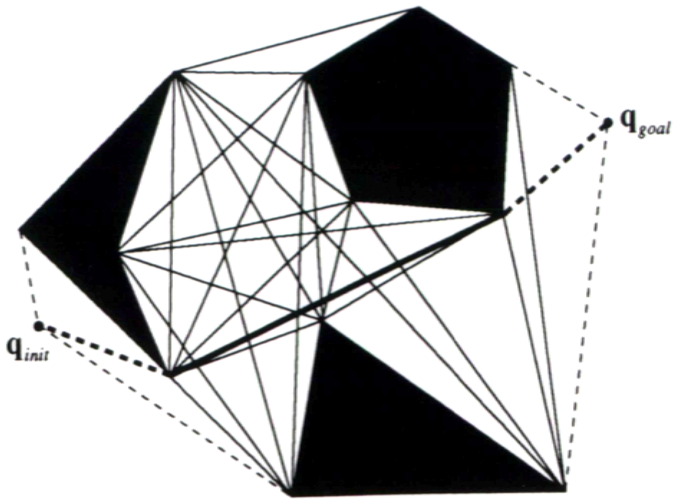

Рис. 5. Решение задачи с помощью графа видимости

Препятствия-круги для этого алгоритма можно аппроксимировать многоугольниками с некоторым заданным числом вершин, описанными вокруг кругов.

Разбиение пространства на регулярные клетки

Пространство конфигураций разбивается на регулярные клетки (Рис. 5). Разбиение может проводиться рекурсивным или иным способом. Клетки могут быть как одного фиксированного размера, так и разных, обычно есть предел на минимальный размер клетки.

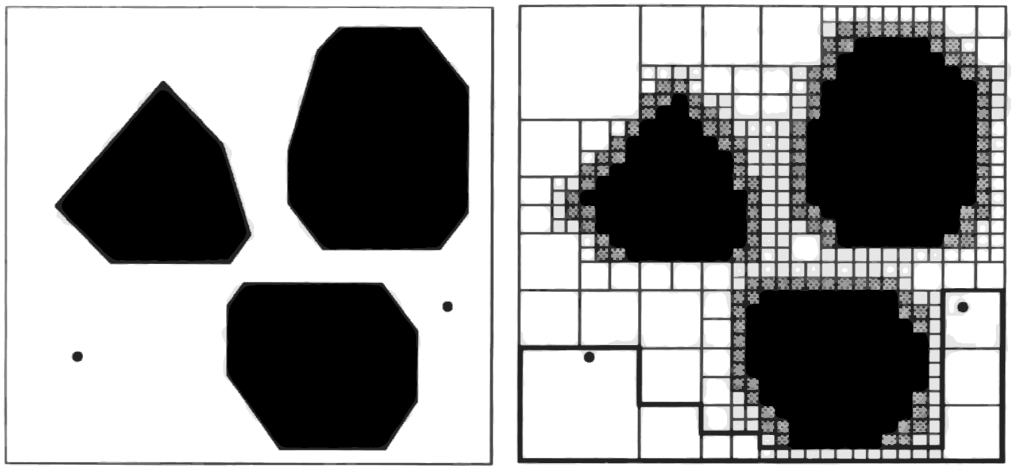

Рис. 6. Декомпозиция пространства конфигураций на регулярные клетки

Клетки делятся на две группы - свободные и занятые, свободные клетки полностью лежат в C_{free}, занятые - частично или полностью пересекаются с C_{obs}. Свободные клетки объединяются в граф, вершины графа - центры клеток, ребра - пути между центрами клеток по прямой (Рис. 7). Для полученного графа решается задача поиска кратчайшего пути (Рис. 8).

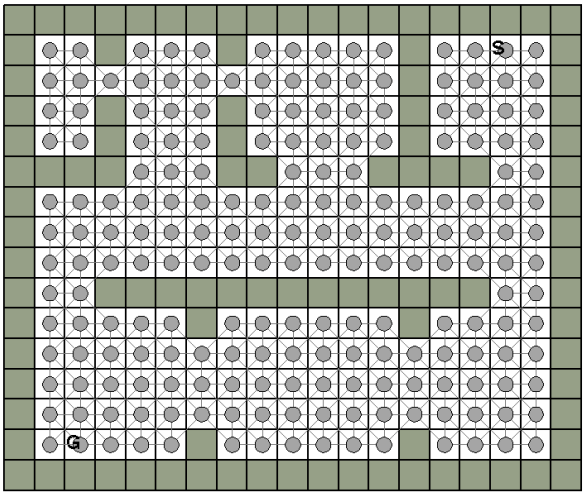

Рис. 7. Объединение свободных клеток в связный граф

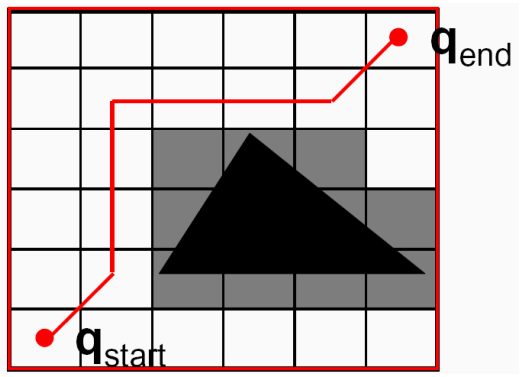

Рис. 8. Решение задачи с помощью разбиения пространства на регулярные клетки

Методы, основанные на случайных образцах

Данные методы исследуют пространство C с помощью случайных образцов - сэмплов. При этом представление C_{obs} в явном виде обычно не используется, а используется только проверка на столкновение робота с препятствиями в заданной конфигурации. Результатом исследования является граф или дерево, в котором затем ищется кратчайший путь.

Probabilistic Road Maps (PRM)

В данном методе в C_{free} сначала строится граф (карта дорог - road map) на основе случайных конфигураций из C_{free}. Затем к графу добавляются начальная и конечная конфигурации и задача решается методом поиска кратчайшего пути в графе (Рис. 9). Построенную карту дорог можно переиспользовать для поиска пути между разными конфигурациями.

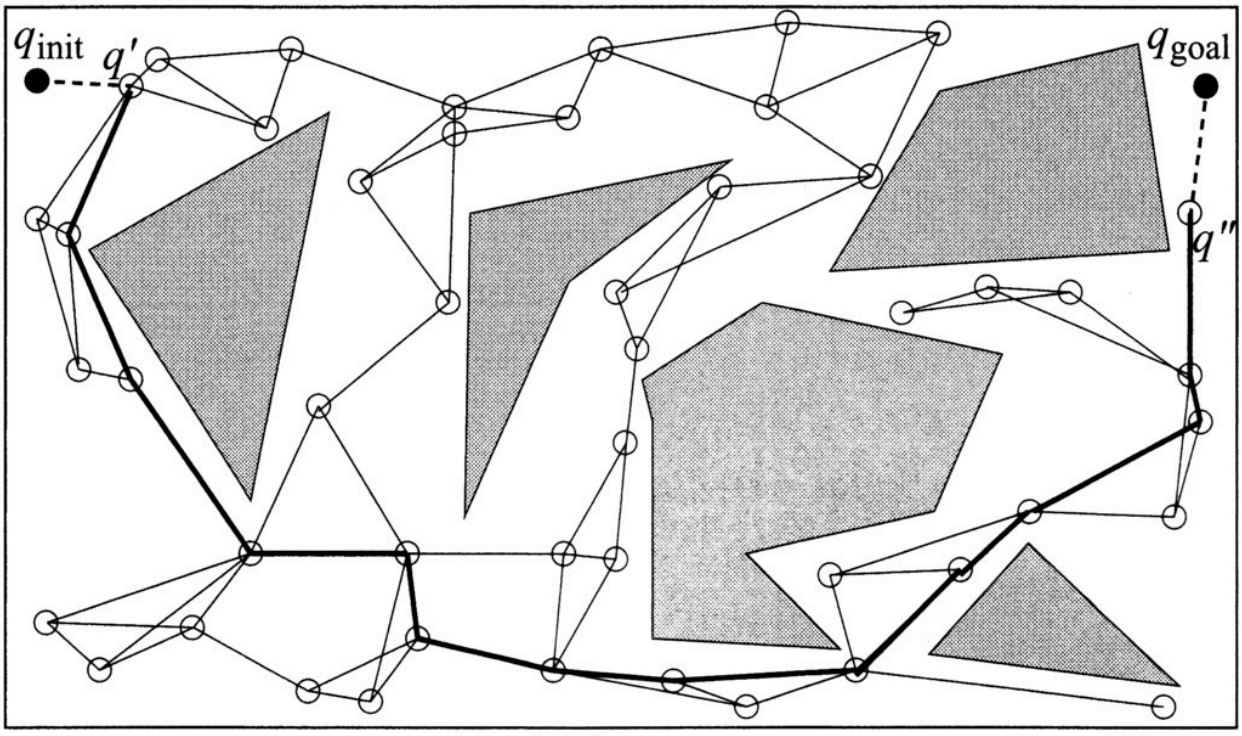

Рис. 9. Решение задачи с помощью PRM

Описание вспомогательных функций:
RandomSample() - возвращает новую случайную конфигурацию из C_{free}.
CollisionFree(X, Y) - проверяет, что путь по прямой из конфигурации X в Y не приводит к коллизиям. Для проверки может использоваться или явное представление C_{obs}, либо проверки на коллизии могут проводиться в конфигурациях из разных мест отрезка XY с некоторым шагом.
Near(G, X, k) - возвращает k вершин из графа G, ближайших к конфигурации (вершине) X. Альтернативная версия функции может вместо числа вершин k использовать расстояние dist: тогда возвращаются все вершины графа ближе чем dist к вершине X.

```
Параметры алгоритма PRM:
N - число сэмплов (чем больше тем точнее будет результат)
k - число ближайших вершин к рассмотрению (альтернативно - максимальное расстояние dist для поиска ближайших вершин)
Qinit, Qgoal - начальная и конечная конфигурации

PRM(N, k, Qinit, Qgoal):
     G - пустой граф
     Пока число вершин в графе G меньше чем N:
          Qrand = RandomSample()
          Добавить вершину Qrand в граф G
     Для каждой вершины Q из G:
          Qnear = Near(G, Q, k)
          for Qn in Qnear:
               if CollisionFree(Q, Qn):
                    Добавить ребро (Q, Qn) в граф G
     Соединить Qinit и Qgoal с графом G (добавить вершины и ребра)
     Вернуть кратчайший путь в графе G из Qinit в Qgoal
```

Rapidly Exploring Random Trees (RRT)

В данном методе из начальной конфигурации q_{Init} строится дерево, покрывающее C_{free} (Рис. 10). После построения дерева в него добавляется q_{Goal} и производится поиск кратчайшего пути из начальной конфигурации в конечную (Рис. 11). Данный метод ориентирован на поиск ответа на один запрос с конкретными q_{Init} и q_{Goal}.

Рис. 10. Построенное дерево после разного числа итераций

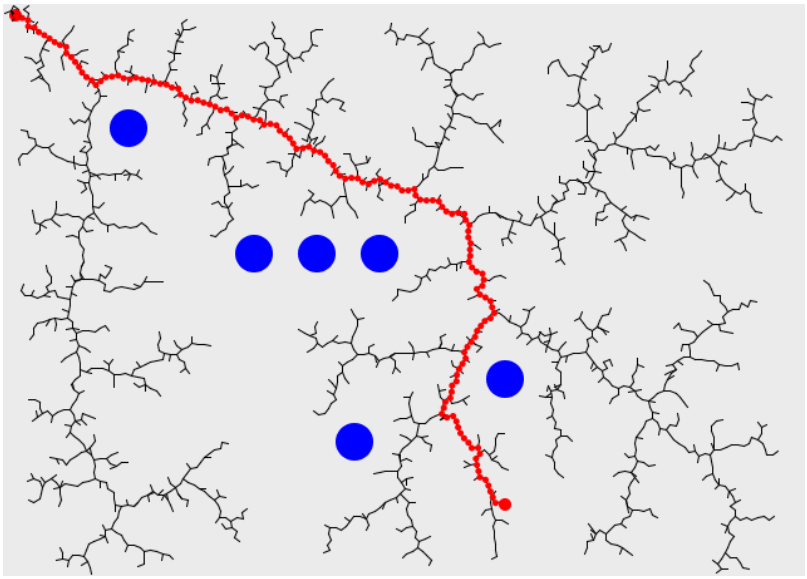

Рис. 11. Решение задачи с помощью RRT

Описание вспомогательных функций:
RandomSample() - возвращает новую случайную конфигурацию из C_{free}.
CollisionFree(X, Y) - проверяет, что путь по прямой из конфигурации X в Y не приводит к коллизиям. Для проверки может использоваться или явное представление C_{obs}, либо проверки на коллизии могут проводиться в конфигурациях из разных мест отрезка XY с некоторым шагом.
Nearest(G, X) - возвращает вершину (конфигурацию) из графа G, ближайшую к конфигурации X. Если оказывается, что ближайшая к X конфигурация лежит на ребре графа, в этом месте ребро делится на две части и в граф добавляется новая вершина (Рис. 12).
Steer(X, Y) - возвращает новую конфигурацию Z из C_{free}, лежащую на отрезке XY и ближайшую к Y, такую что CollisionFree(X, Z) (Рис. 13).

Параметры алгоритма RRT:
N - число сэмплов (чем больше тем точнее будет результат)
Qinit, Qgoal - начальная и конечная конфигурации

```
RRT(N, Qinit, Qgoal):
      G - пустой граф
      Добавить вершину Qinit в G
      for i = 1..N:
            Qrand = RandomSample()
            Qn = Nearest(G, Qrand)
            Qs = Steer(Qn, Qrand)
            Добавить вершину Qs в граф G
            Добавить ребро (Qn, Qs) в граф G
      Соединить Qgoal с графом G (добавить вершину и ребро)
      Вернуть кратчайший путь в графе G из Qinit в Qgoal
```

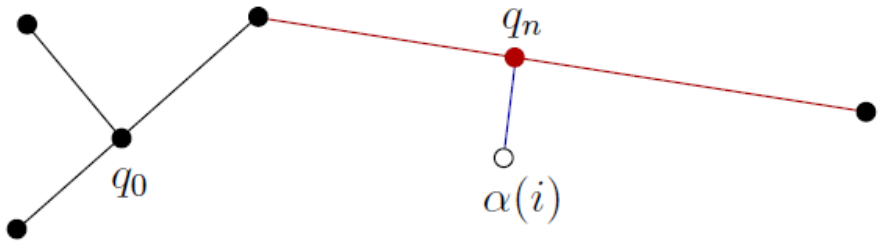

Рис. 12. Добавление новой вершины при нахождении ближайшей конфигурации на ребре

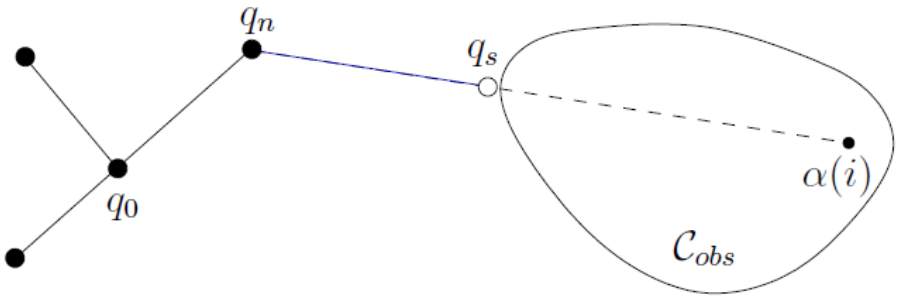

Рис. 13. Поиск ближайшей к сэмплу конфигурации, не приводящей к коллизии

В альтернативной реализации метода могут строиться сразу два дерева: из q_{Init} и q_{Goal}, после чего производится попытка объединения деревьев.

Требования к реализации

РГЗ включает в себя две части:

1. реализация алгоритмов
2. разработка GUI-приложения, позволяющего графически продемонстрировать работу реализованных алгоритмов в интерактивном режиме

Математическую часть (алгоритмы) желательно поместить в отдельный подмодуль. Этот подмодуль может использоваться как в GUI-приложении, так и, например, в обычном консольном приложении.

Основные возможности, которые должно поддерживать GUI-приложение:

- отображение сцены: препятствий, робота, начальной, конечной и промежуточных (при наличии в задании) точек пути
- интерактивное задание препятствий, размера робота, положения начальной, конечной и промежуточных (при наличии в задании) точек пути
- возможность сохранения сцены в файл и загрузки ее из файла
- отображение построенного пути или сообщение о том что путь не удалось построить
- желательна возможность показа положения робота в произвольных точках пути для проверки отсутствия коллизий

Для разработки можно использовать любой язык программирования, в т.ч. любую библиотеку/фреймворк для создания GUI.

Варианты заданий

Вариант = Алгоритм, тип препятствий, наличие промежуточных координат.

1. Граф видимости, прямоугольники, без промежуточных координат.
2. Разбиение на клетки, прямоугольники, без промежуточных координат.
3. PRM, прямоугольники, без промежуточных координат.
4. RRT, прямоугольники, без промежуточных координат.
5. Граф видимости, круги, без промежуточных координат.
6. Разбиение на клетки, круги, без промежуточных координат.
7. PRM, круги, без промежуточных координат.
8. RRT, круги, без промежуточных координат.
9. Граф видимости, треугольники, без промежуточных координат.
10. Разбиение на клетки, треугольники, без промежуточных координат.
11. PRM, треугольники, без промежуточных координат.
12. RRT, треугольники, без промежуточных координат.
13. Граф видимости, прямоугольники, с промежуточными координатами.
14. Разбиение на клетки, прямоугольники, с промежуточными координатами.
15. PRM, прямоугольники, с промежуточными координатами.
16. RRT, прямоугольники, с промежуточными координатами.
17. Граф видимости, круги, с промежуточными координатами.
18. Разбиение на клетки, круги, с промежуточными координатами.
19. PRM, круги, с промежуточными координатами.
20. RRT, круги, с промежуточными координатами.
21. Граф видимости, треугольники, с промежуточными координатами.
22. Разбиение на клетки, треугольники, с промежуточными координатами.
23. PRM, треугольники, с промежуточными координатами.
24. RRT, треугольники, с промежуточными координатами.

Литература

1. Steven M. LaValle, Planning algorithms, Cambridge University Press, 2006.
2. Howie Choset и др., Principles of robot motion: theory, algorithms and implementation, The MIT Press, 2005.
3. Jean-Claude Latombe, Robot motion planning, Springer US, 1991.

Дополнение: алгоритмы проверки геометрических фигур на пересечение

```
Вспомогательные процедуры:
dot(V1, V2) = V1.x*V2.x + V1.y*V2.y - скалярное произведение векторов V1 и V2
cross(V1, V2) = V1.x*V2.y - V2.x*V1.y - векторное произведение векторов V1 и V2

closest(A, B, P) - ближайшая к точке P точка на отрезке (A, B):
d = dot(AP, AB) / dot(AB, AB)
```

```
d = clamp(d, 0, 1) - ограничить d диапазоном [0, 1] (без ограничения получим ближайшую точку на линии проходящей через (A, B))
Вернуть A + (B - A)*d

Пересечение двух точек P1 и P2:
abs(P1.x - P2.x) < EPS и abs(P1.y - P2.y) < EPS (EPS - параметр точности, например, 1e-6)

Пересечение точки P и отрезка (A, B):
cross(AB, AP) равно 0 и dot(AB, AP) <= dot(AB, AB)

Пересечение точки P и круга (C - центр, r - радиус):
dot(PC, PC) <= r*r

Пересечение точки P и треугольника (A, B, C):
s1 = cross(AB, AP)
s2 = cross(BC, BP)
s3 = cross(CA, CP)
Все (s1, s2, s3) >= 0 или все (s1, s2, s3) <= 0

Пересечение отрезков (A, B) и (C, D):
denom = cross(AB, CD)
alpha = cross(AC, AB)
beta = cross(AC, CD)
Если denom равно 0: (отрезки коллинеарны)
     Если alpha не равно 0 - отрезки параллельны и не пересекаются
     t0 = dot(AC, AB) / dot(AB, AB)
     t1 = t0 + dot(CD, AB) / dot(AB, AB)
     (t0, t1) = (min(t0, t1), max(t0, t1))
     Интервал [t0, t1] пересекается с интервалом [0, 1]
Оба alpha/denom и beta/denom находятся в интервале [0, 1]

Пересечение отрезка (A, B) и круга (C, r):
closest(A, B, C) пересекается с кругом (C, r)

Пересечение отрезка и треугольника:
Какой-либо из концов отрезка пересекается с треугольником или отрезок пересекается с какой-либо из сторон треугольника
```