Название: Алгоритм реконфигурации роя роботов

*Аннотация*. В статье описан алгоритм перестройки роя роботов в предписанную формацию. Основное требование к реконфигурации таково: движение к целевым точкам не должно создавать коллизий, то есть, в каждый момент времени попарные расстояния между роботами должны быть не меньше некоторого фиксированного минимального допустимого значения. Наш подход основан на ряде методов, заимствованных из вычислительной геометрии. Прежде всего мы отмечаем, что евклидово соответствие с минимальной ценой (Euclidean minimum cost bipartite matching) часто дает возможность реконфигурации без коллизий с прямолинейными траекториями. Простые примеры показывают, что такое происходит не всегда, поэтому мы разрабатываем дополнительный инструментарий: обмен целевыми точками, назначение задержек, гомотетии, а также прием "разделяй-и-властвуй".

*Ключевые слова*: навигационный алгоритм, групповое управление, робототехнические комплексы.

1. Введение

Задачи реконфигурации роя роботов являются актуальными в современном мире. Их исследуют и стараются реализовать на практике. В таких задачах, как правило, задействованы автономные мобильные роботы, выполняющие широкий спектр гражданских и военных задач, таких как анализ огромной территории, спасательные операции и многие другие задачи в различных сферах деятельности. Реконфигурация автономных мобильных роботов в пространстве является основополагающим моментом в системе управления роя роботов. В данной статье представлены алгоритмы для расчёта оптимальных траекторий перемещения роботов к своим целевым точкам. Прежде чем описать разработанные алгоритмы, проанализируем уже существующие методы решения похожих задач.

В литературе описано множество версий задачи о реконфигурации с различными сценариями, правилами и подходами, см. например, [1, 2, 3, 4, 5]. Традиционным является подход, основанный на взаимодействиях между роботами, то есть имеющий дело с робототехнической нейросетью. Также традиционны алгоритмы градиентного спуска, см., например, [1].

Часто предполагается, что рабочее пространство роя роботов – некоторый (возможно, невыпуклый) многогранник. В настоящей статье мы не ставим подобного ограничения. Имеются некоторые свидетельства о том, что в общей постановке задача может оказаться неразрешимой (или, по крайней мере, сильно усложниться) в связи с существованием самозаклинивающихся структур, см. [6, 7, 8]. Однако упомянем и положительные результаты о реконфигурации в ограниченном пространстве, см. [9].

Нам особенно хочется выделить серию статей [10], [11] и [12], рассматривающие почти идентичную задачу. А именно, авторы рассматривают конфигурации взаимно непересекающихся непронумерованных дисков. Реконфигурация состоит в серии последовательных прямолинейных перемещений дисков по одному ("шагов"). Задача состоит в минимизации числа шагов. Полученные результаты в этом направлении таковы: (1) Для n дисков всегда достаточно 2n − 1 шага. (2) Задача о реконфигурации в модели переносов NP-сложна. (3) Для некоторых конфигураций [5n/3]-1 – необходимое число шагов, то есть мы не можем всегда рассчитывать на реконфигурацию без коллизий с исключительно прямолинейными траекториями.

Предложенные нами алгоритмы отличаются от вышеуказанных в том, что у нас все роботы движутся одновременно, а не по очереди. Кроме того, мы минимизируем не число шагов, а энергозатраты реконфигурации, то есть суммарную длину траекторий. Мы ожидаем, что наилучшие практические результаты появятся при комбинации нашего метода с виртуальными взаимно отталкивающими силами между роботами, действующими при малых расстояниях. Подобное комбинирование сделает наш алгоритм робастным. Однако, детальное развитие этой темы мы оставляем за рамками статьи.

1. Постановка задачи

В представленной работе в результате реконфигурации роя, роботы должны покрыть с заданной плотностью определенную поверхность. В этой работе поверхность представляет из себя выпуклый многоугольник – ромб. В этой задаче отсутствуют препятствия и все перемещения происходят в двумерном пространстве, что позволяет предложить относительно простые алгоритмы реконфигурации.

Подробное описание поставленной задачи. Входные данные:

* Дано *N* одинаковых(гомогенных) объектов-роботов, представляющие из себя рой. Каждый из них имеет форму шара с заданным радиусом *R*. Случайным образом задаются координаты центров этих роботов.
* Минимальное допустимое расстояние между центрами роботов – *min\_dist*.
* Скорость движения роботов – *v*. Предполагается, что объекты движутся либо с этой скоростью, либо неподвижны.
* Выпуклая поверхность – *T*. Поверхность задана набором вершин *t’ = (x’,y’,z’)* и списком рёбер.
* Плотность размещения – *density.*

В таких условиях необходимо, избегая коллизий, переместить роботы к целевым (таргентным) точкам, из которых состоит выбранная поверхность. Выходными данными являются:

* Конечные координаты роботов, участвующих в реконфигурации.
* Оценка времени перестроения объектов в заданную фигуру.
* Рассчитанные траектории перемещения роботов к целевым точкам.

Необходимо учесть следующие условия:

* Количество объектов достаточно для покрытия поверхности.
* Плотность и минимальное допустимое расстояние связаны соотношением:

. (1)

Время реконфигурации роя зависит от расположения роботов и поверхности. Новизна алгоритмов состоит в том, что все траектории роботов прямолинейны. Это достигается выбранным инструментарием, включающим в себя математические и геометрические операции, метод назначения задержек и многое другое.

Некоторые особенности работы алгоритма:

* Работа алгоритма начинается с генерации координат центров роботов, координат точек, составляющих поверхность.
* Далее все входные данные подвергаются проверке на их корректность.
* Каждой точке поверхности присваивается робот, что позволяет называть эту точку таргетной (или целевой) для этого робота.

Основополагающая цель алгоритмов - избежать коллизии (столкновения роботов из-за меньшего расстояние между центрами роботов, чем минимальное допустимое).

Другими словами, главная задача — это найти алгоритм реконфигурации для любых наперед заданных допустимых конфигураций *S* и *T*. Реконфигурация должна иметь максимально возможную низкие энергозатраты (суммарное перемещение роботов должно быть возможно мало), а также занимать минимально возможное время (задержки роботов должны быть минимальны).

1. Основные определения и понятия

Мы предполагаем, что заданы две равномощные допустимые конфигурации *S = {s1,...,sN}* (стартовая конфигурация) и *T = {t1,...,tN}* (целевая конфигурация). Соответствие между *S* и *T* есть произвольная биекция между конфигурациями. Соответствие мы выражаем индексированием точек, то есть мы предполагаем, что точка *si* соответствует точке *ti.* Если мы хотим поменять соответствие, мы меняем нумерацию целевых точек. Суммарной длиной соответствия называется сумма длин всех соединяющих отрезков: TL(*M*) = . Соответствие короткое, если его суммарная длина TL(*M*) минимальна. Алгоритм навигации (или реконфигурации) без коллизий для конфигураций *S* и *T* есть кусочно-линейная гомотопия в классе допустимых конфигураций, переводящая *S* в *T*. (Последнее означает отсутствие коллизий. В каждый момент конфигурация должна быть допустимой)

Соответствие свободно, если расстояние между всякими двумя отрезками [*siti*] и [*sjtj*] не меньше *min\_dist* при *i* ≠ *j*. При свободном соответствии роботы могут двигаться одновременно по прямолинейным траекториям, и коллизий не возникнет. Простой пример показывает, что свободное соответствие существует не всегда, даже если число роботов всего два, см. рисунок 1. Однако, если один из роботов в приведенном примере начнет свое движение с задержкой, коллизии не произойдет.

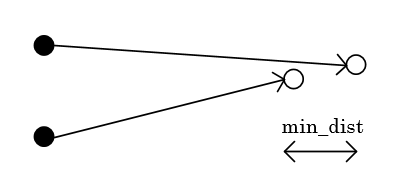


Рисунок 1 – Чёрные точки обозначают стартовую конфигурацию (роботы), белые, соответственно, целевую конфигурацию.

Зависимые пары индексов можно разделить на следующие категории:

1. Коллизия в целевой точке. Коллизия в точке ti означает выполнение этого условия:

*dist(ti,[sj,tj])* < *min\_dist* и *dist(si,[sj,tj])* > *min\_dist*.

Это отражено на рисунке 2 (а). Коллизия в точке *t*j определяется аналогично.

1. Коллизия в стартовой точке. Коллизия в точке si означает выполнение этого условия:

*dist(ti,[sj,tj])* > *min\_dist* и *dist(si,[sj,tj])* < *min\_dist*.

Это отражено на рисунке 2 (b). Коллизия в точке *sj* определяется аналогично.

1. Косая коллизия. В этом случае коллизии в стартовых и целевых точках *ti,tj,si,sj* отсутствуют, то есть коллизия происходит во внутренних точках, см. рисунок 2, (с).
2. Сильно зависимая пара: коллизия вдоль. Коллизия вдоль отрезка номер i означает, что

*dist(ti,[sj,tj]) < min\_dist* и *dist(si,[sj,tj]) < min\_dist*.

Это показано на рисунке 2 (d).

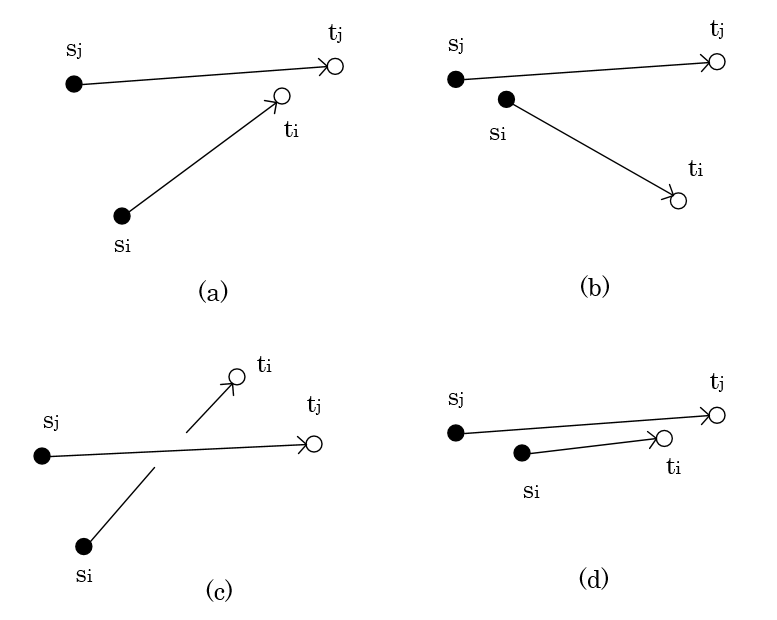


Рисунок 2 – Зависимые пары

1. Инструментарий
2. Гомотетия и параллельный перенос. Отметим то, что:

* Если конфигурация *S* лежит "далеко от" *T*, то число коллизий при прямолинейной реконфигурации больше, чем в случае, когда *S* лежит "близко от" *T*. Следовательно, имеет смысл вначале переместить конфигурацию S в другое, более близкое положение *S′*, а затем реконфигурировать *S′* в *T*. Рисунок 3 поясняет эту идею: для (a) нет свободного соответствия, тогда как после параллельного переноса (b) таковое появляется.
* Любой коллизии можно избежать с помощью подходящей гомотетии: при достаточно большом растяжении (а также при условии общего положения) всякое соответствие становится свободным. Однако, это средство крайне неэкономно, так как гомотетия с небольшим коэффициентом приводит к длинным перемещениям роботов.
* Композиция параллельного переноса и гомотетии есть опять гомотетия, с тем же самым коэффициентом, но с другим центром. Тем самым, связанные с ними реконфигурации комбинируются в одну реконфигурацию с прямолинейными траекториями.

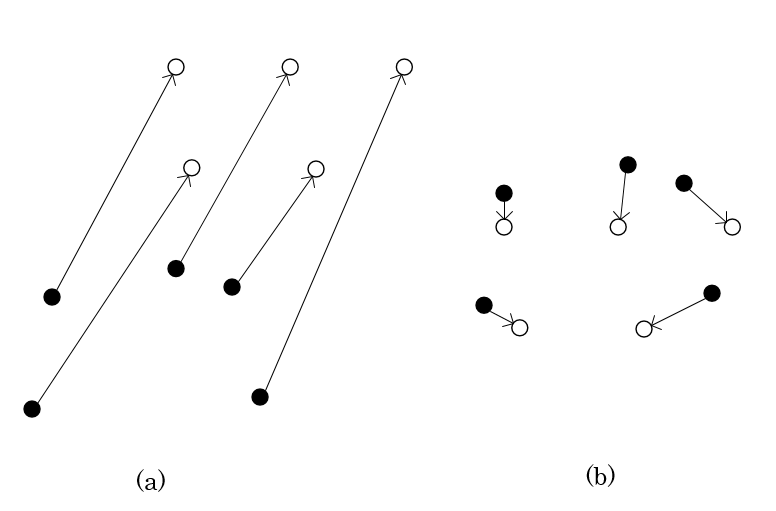


Рисунок 3 – Пример использования гомотетии. (a) Короткое соответствие не свободно, (b) после параллельного переноса короткое соответствие свободно.

Придадим вышесказанному точный смысл. Напомним вначале, что центральной точкой (centerpoint) множества точек называется такая точка, что всякая гиперплоскость, содержащая ее, делит множество на две "примерно равные" части: меньшая часть содержит по крайней мере *1/d+1* долю всего множества. Неформально говоря, средняя точка есть "геометрическая медиана" множества. Во многих вычислительных задачах и геометрически обусловленных алгоритмах выбор центральной точки приводит к лучшим результатам, чем выбор центра тяжести. Центральная точка существует всегда, но может быть не единственной. Найти центральную точку плоской конфигурации можно за линейное время, см. [16], а в размерности 3 – с помощью квадратичного алгоритма [17].

Для точек A, B и положительного числа *λ* обозначим через DILA,B,λ гомотетию с коэффициентом *λ* такую, что DILA,B,λ(A) = B (это условие определяет центр гомотетии).

Пусть S – центральная точка конфигурации *S* , а *T* – центральная точка конфигурации T. Рассмотрим конфигурации *S*, DILS,T,λ(*S*), DILT,T,λ(*T* ), и *T* .

По построению, *S* и DILS,T,λ(*S*) связаны гомотетией. Естественная реконфигурация S в DILS,T,λ(*S*) такова: все роботы начинают двигаться одновременно и прямолинейно, но движутся с разными скоростями. Скорости должны быть пропорциональны расстоянию до центра гомотетии. Подобная реконфигурация существует для DILT,T,λ(*T*), *T* . Конфигурации DILS,T,λ(*S*) и DILT,T,λ(*T*) являются растянутыми исходными конфигурациями, сдвинутые таким образом, что их центральные точки совпадают. Если для них имеется алгоритм реконфигурации, то мы получаем комбинацию реконфигураций:

*S* → DILS,T,λ(*S*) → DILT,T,λ(*T* ) → *T* .

1. Граф коллизий и назначение задержек.

Предположим, что имеется соответствие между двумя конфигурациями точек. Назначение задержек стартовым точкам свободно от коллизий, если прямолинейное движение к целевым точкам с предписанными задержками не создает коллизий.

Устранение коллизий для одной отдельной пары назначением задержки. Пусть имеется зависимая (но не сильно зависимая) пара. Тогда существует (легко вычислимая) задержка *D* для одного из роботов, устраняющая коллизию. В случае косой коллизии задержка может быть назначена любому из двух роботов, то есть возникают сразу две задержки *Di* и *Dj*. Такую задержку назовем локальной.

Для двух конфигураций с соответствием между ними граф коллизий Γ(*S*,*T*) есть частично ориентированный граф с N вершинами, занумерованными индексами 1,...,N. С каждой из вершин мы ассоциируем точки *si* и *ti.* Ребра графа (некоторые из которых ориентированы, а некоторые – нет) заданы следующим правилом:

1) Вершины *i* и *j* соединены ориентированным ребром −→ *ij* если (*i,j*) – зависимая пара с коллизией либо в целевой в точке *ti,* либо в стартовой точке *sj* (также возможно, что эти коллизии происходят одновременно).

2) Вершины *i* и *j* соединены неориентированным ребром, если (*i,j*) – зависимая пара с косой коллизией.

3) В случае сильно зависимой пары (*i,j*) вершины *i* and *j* соединены двумя ребрами с противоположными ориентациями.

4) Наконец, в случае свободной пары (*i,j*) ребро отсутствует. Пример графа коллизий – см. рисунок 8. Соответствие между конфигурациями точек *S* и *T* почти свободно, если граф коллизий ацикличен, то есть, не содержит ориентированных циклов. В дальнейшем нам понадобится следующий вспомогательный алгоритм:

Алгоритм дополнительных ориентаций.

Предположим, что имеется частично ориентированный ацикличный граф. Следующий квадратичный алгоритм ориентирует все неориентированные ребра графа с сохранением ацикличности:

1) Возьмем произвольное неориентированное ребро графа *ij*. Зададим на нем произвольную ориентацию.

2) Ищем ориентированный цикл в графе, содержащий ребро *ij*, пользуясь, например, алгоритмом из [15].

a) Если ориентированный цикл найден, поменяем ориентацию ребра *ij*. Перейдем к следующему неориентированному ребру.

b) Если ориентированный цикл не найден, сохраним выбранную ориентацию *ij* и перейдем к следующему неориентированному ребру.

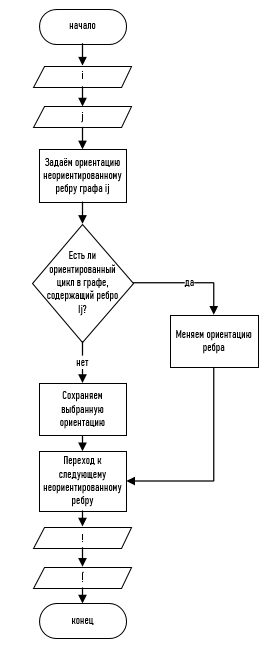


Рисунок 4 - Алгоритм дополнительных ориентаций

Для двух конфигураций с почти свободным соответствием существует назначение задержек, гарантирующее реконфигурацию с прямолинейными траекториями без коллизий:

Алгоритм назначения задержек при почти свободном соответствии.

1) Построим граф коллизий Γ(*S,T* ). Согласно предположению, он ацикличен.

2) Пользуясь алгоритмом дополнительных ориентаций, присвоим ориентацию каждому из неориентированных ребер, сохраняя ацикличность графа. Полученный граф обозначим через .

3) Для каждого ребра − вычислим задержку D = D(*si,j,ti,j*). В случае косой коллизии, выберем одну из двух задержек *Di* или *Dj* в соответствии с выбранной ориентацией ребра.

4) Введем отношение частичного порядка на вершинах графа : будем говорить, что *i ≥ j*, если в графе найдется ориентированный путь *i → j*.

5) Возьмем все стоки (вершина ориентированного графа без исходящих ребер. Изолированная вершина также является стоком.) *j* в графе и присвоим соответствующим стартовым точкам нулевую задержку *Del(j) :=* *0*. Пометим все стоки. Начиная с этого момента, некоторые вершины графа помечены, а некоторые – нет, при этом помеченным вершинам присвоены задержки. Мы собираемся постепенно присвоить некоторые задержки всем вершинам, и, соответственно, пометить все вершины.

6) Пока не все вершины помечены,

a) Возьмем непомеченную вершину *i*, такую, что все вершины, меньшие i помечены.

b) Рассмотрим множество *J* вершин выходящих ребер *ij*. Согласно конструкции, все вершины в множестве *J* помечены, и, следовательно, им уже присвоена некоторая задержка *Del(j).* Положим *Del(i)* = *maxj∈J(Del(j) + D(si,j,ti,j)).*

c) Пометим вершину *i*.

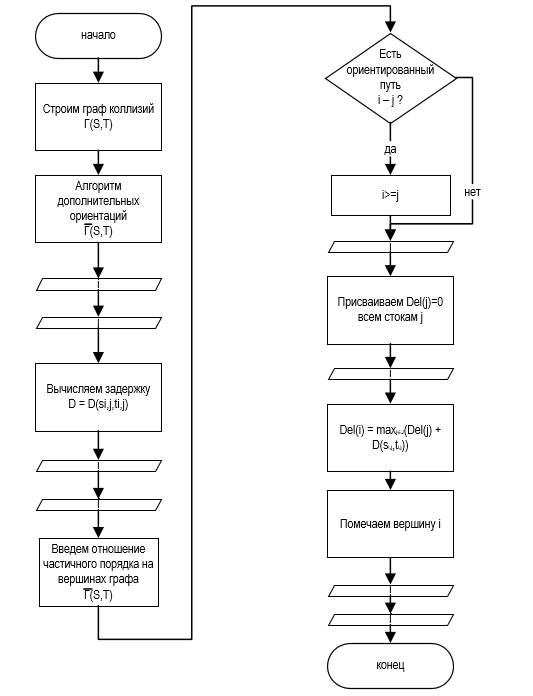


Рисунок 5 - Алгоритм назначения задержек при почти свободном соответствии

1. Обмен целевыми точками.

Пусть имеется сильно зависимая пара (скажем, с зависимостью вдоль i-го отрезка). Обмен целевыми точками дает пару, которая больше не является сильно зависимой, см. рисунок 6. Сочетая обмен целевыми точками с назначением задержек, мы получаем алгоритм, но к сожалению, он работает не во всех ситуациях, поэтому его необходимо комбинировать с другими приемами, о чем и пойдет речь в дальнейшем.

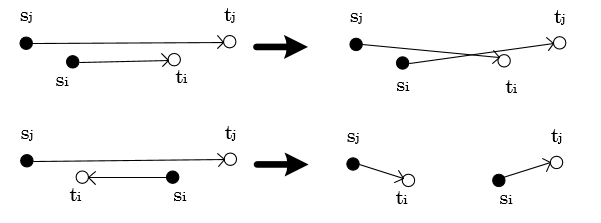


Рисунок 6 - Устранение коллизии вдоль отрезка

Алгоритм для ациклического случая.

1) Найдем короткое соответствие для стартовой и целевой конфигураций, пользуясь алгоритмами [13, 14].

2) Последовательно удалим все сильно зависимые пары, пользуясь обменом целевых точек. Если это невозможно, рапортуем, что "алгоритм не работает".

3) Построим граф коллизий. Если в нем имеются циклы, рапортуем, что "алгоритм не работает".

4) Присвоим стартовым точкам задержки, пользуясь алгоритмом назначения задержек. Полученное соответствие (вместе с задержками) дает нам реконфигурацию с прямолинейными траекториями. То есть, точки (роботы) должны двигаться прямолинейно непосредственно к целевой точке с одной и той же скоростью *v*, начав движение с предписанными задержками.

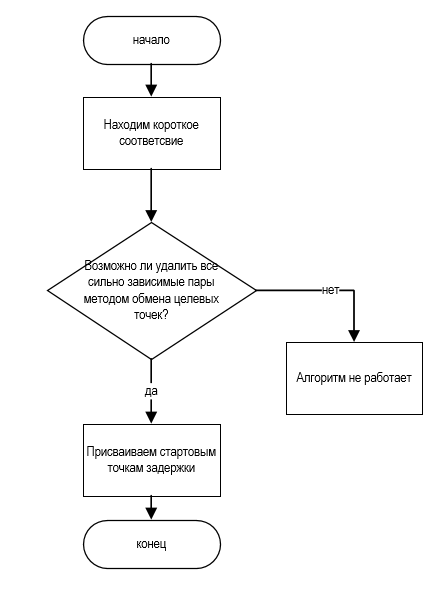


Рисунок 7 - Алгоритм для ациклического случая

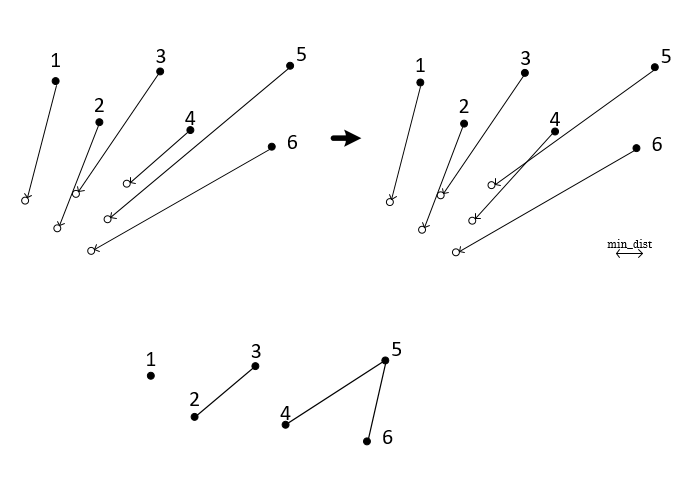


Рисунок 8 - Короткое соответствие, устранение сильно зависимой пары и граф коллизий полученного соответствия

1. Приём «разделяй и властвуй» (“*divide and conquer*”).

Идея приема разделяй и властвуй такова: мы делим конфигурацию на части, и реконфигурируем части одновременно и независимо. Для того, чтобы избежать коллизий между роботами из разных частей, мы раздвигаем части на расстояние *min\_dist*. Это проиллюстрировано на рисунке \_. Теорема о бутерброде с ветчиной (Ham sandwich theorem) утверждает, что всякие две точечные конфигурации *S* и *T* могут быть разделены на равномощные части одним прямолинейным разрезом (или же плоским разрезом в случае *d = 3*). Такой разрез называется разрез, делящий поровну (ham sandwich cut); он может быть найден за *O(NlogN)* шагов.

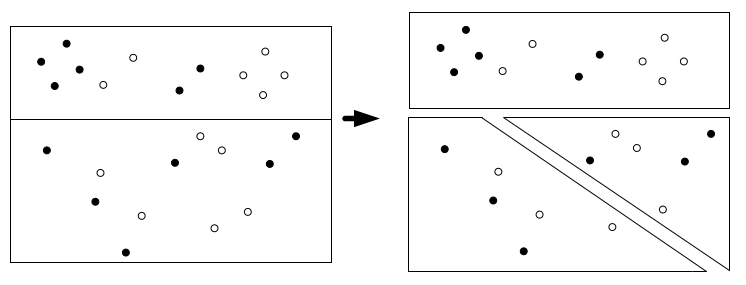


Рисунок 9 - Иллюстрация принципа «разделяй и властвуй»

Алгоритм «разделяй и властвуй»

1. Найдем разрез, делящий поровну конфигурации *S* и *T*. Предположим, что они оказываются разделенными как *S* = и *T* = . Может оказаться, что одна или несколько точек попали на плоскость разреза (Если число роботов нечетно, такое случится обязательно). В этом случае отнесем каждую из пограничных точек к тому или иному фрагменту конфигурации так, чтобы было выполнено: |*S1*| = |*T1*|, и |*S2*| = |*T2*|.

2) Найдем короткое соответствие для каждой из частей.

3) Если алгоритм для ациклического случая работает для пары *S1* и *T1*, а также для пары *S2* и *T2*, запустим его. В противном случае найдем следующий разрез, и будем продолжать до тех пор, пока алгоритм для ациклического случая не будет успешно работать для всех пар фрагментов *Si* и *Ti*.

4) Для каждого индекса i найдем подходящий параллельный перенос фрагментов *Si* и *Ti* так, чтобы все попарные расстояния между выпуклыми оболочками фрагментов были не меньше *min\_dist*. Обозначим перенесенные фрагменты через *S′i* и *T ′i* .

5) Мы получаем желаемую реконфигурацию: *S* = → → → = *T*.

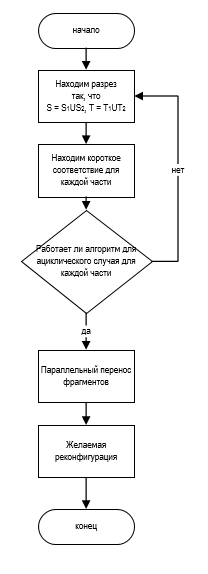


Рисунок 10 - Алгоритм «разделяй и властвуй»

1. Итоговые алгоритмы

Существуют разнообразные комбинации описанного выше инструментария в один алгоритм реконфигурации. В этой работе представлены два итоговых алгоритма; выбор между тем или иным делается в зависимости от конкретной задачи.

Первый алгоритм хорошо работает в случае, когда число роботов небольшое, и когда конфигурация относительно компактна (то есть, максимальное расстояние между роботами невелико).

Алгоритм №1 («лёгкая версия»)

1. Найдем центральные точки конфигураций (центры двух множеств, состоящих из точек). Применим к стартовой конфигурации роботов (т.е. начальных положений) параллельный перенос TR, совмещающий центральные точки *S* и *T*. Обозначим *S’ =* TR*(S).*
2. Найдём короткое соответствие между *S’* и *T*, пользуясь алгоритмом [13,14].
3. Вычислим минимальное расстояние δ между точками, предполагая, что точки начинают двигаться одновременно и прямолинейно, с одной и той же заданной скоростью *v*, от *S’* к *T*. Рассчитаем коэффициент гомотетии *λ* как

*.*

1. Произведем гомотетию с рассчитанным коэффициентом *λ*. Положим

*S’’ = DILS,T,λ(S), T’ = DILS,T,λ(T)*

1. Имеем *S→S’→S’’→T’→T* – реконфигурация без коллизий.

В сложных ситуациях число δ может оказаться малым, и предложенный алгоритм будет работать, но крайне неэффективно с точки зрения энергозатрат. Второй алгоритм работает всегда:

Алгоритм №2 (для худшего случая)

1. Первые два шага повторяются из Алгоритма №1. В результате получаем конфигурации *S’* и *T* с общей центральной точкой, а также короткое соответствие между конфигурациями.
2. Запустим алгоритм для ацикличного случая. Если он сработает, то на выходе мы получаем реконфигурацию без коллизий, соединяющую *S’* и *Т*.
3. Иначе, применяем приём «разделяй и властвуй» до тех пор, пока п.2 не будет работать для всех фрагментов *S’i* и *Ti*.
4. Получаем реконфигурацию без коллизий.

Обобщённый алгоритм представлен на рисунке 11. Здесь совмещены два разработанных алгоритма в единый. Именно на основе этого была написана программа для тестирования и анализа полученных данных.

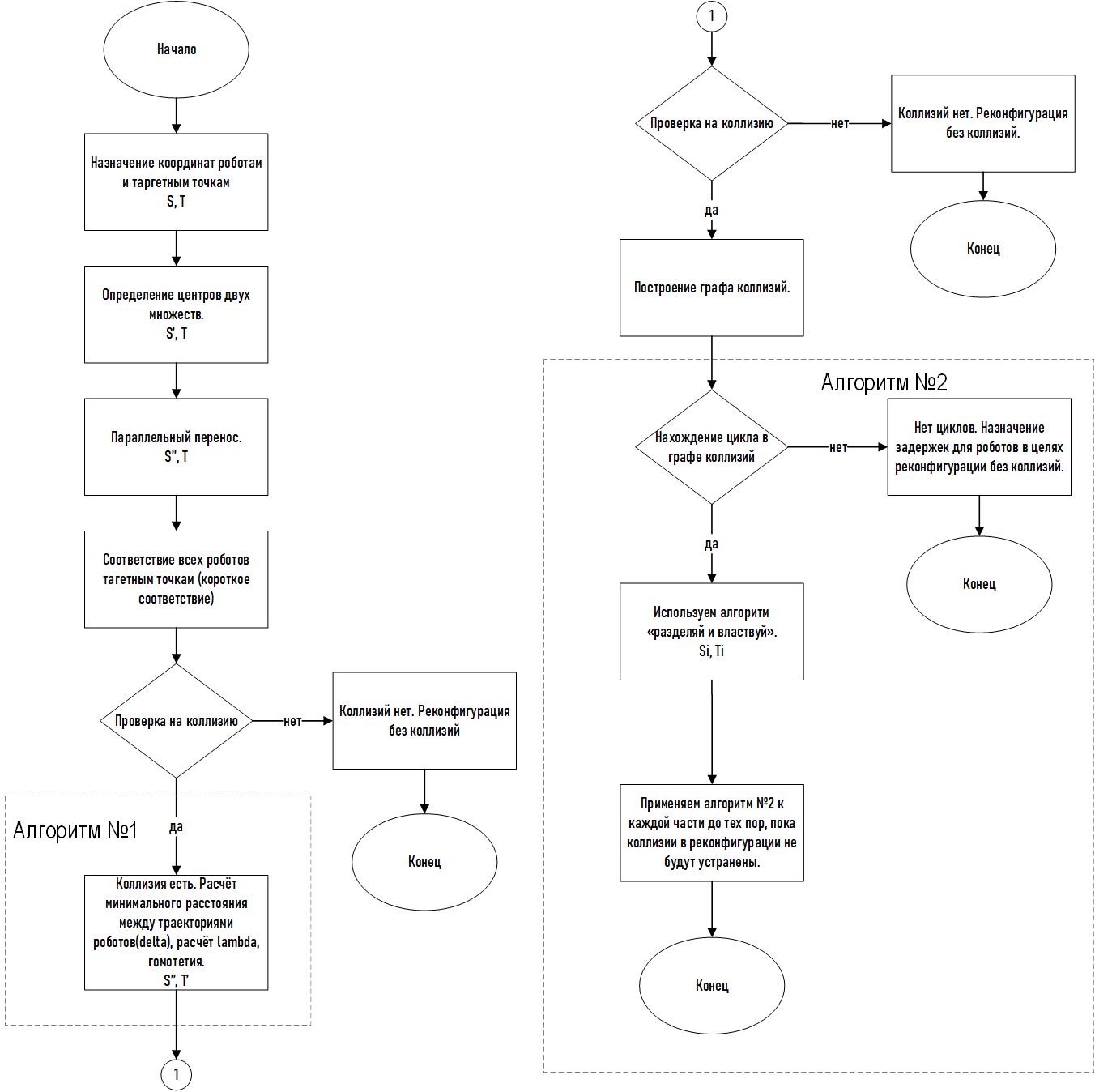


Рисунок 11 - Разработанный алгоритм

1. Результаты работы алгоритмов

Целью проведенных исследований и экспериментов являлась оценка времени, затраченного на перестроение роя роботов и оценка общей длины перемещения этих роботов. Для проведения двух экспериментов была выбрана поверхность – ромб.

Задачей первого эксперимента было выявление минимального количества роботов в рое для реконфигурации при помощи первого алгоритма. Также в этом эксперименте оценивалось среднее время вычислений и общая длина перемещений роботов.

За минимальное число объектов роя было принято число 4. Для каждого количества роботов начиная с 4 проводилось 10 тестов. В конце каждого теста, разработанная программа выводила на экран информацию о том, какой алгоритм был использован.

Все выходные данные были записаны в таблицу. С увеличением числа роботов в рое частота использования «лёгкой версии» алгоритма уменьшалась (частота использования была посчитана как отношение количества срабатываний алгоритма №1 к общему числу испытаний). И напротив, частота использования второго, более сложного алгоритма, увеличивалась. Из этого можно сделать вывод о том, что первый алгоритм стабильно работает при небольшом количестве роботов. При большом числе объектов в системе, рассчитанный коэффициент гомотетии становится очень большим, и из-за этого алгоритм становиться энергозатратным, т.к. общая длина перемещений всех роботов увеличивается до огромных значений.

Рисунок 12 - График частоты использования алгоритмов

Среднее время вычислений уменьшается с увеличением количества объектов. Это можно объяснить тем, что с увеличением роботов необходимость использования первого алгоритма сводится к минимальной, соответственно не нужно рассчитывать некоторые параметры, необходимые для алгоритма №1.

Рисунок 13 - График среднего времени вычислений

Средняя длина перемещения всех роботов увеличивается по тому, как увеличивается количество роботов и расстояние, которое им нужно преодолеть.

Рисунок 14 - Средняя длина перемещения всего роя

Второй эксперимент проводился при разном количестве роботов от 5 до 1000. Размер конечной конфигурации был рассчитан для каждого случая исходя из плотности размещения роботов по поверхности. Рассматривался случай расположения роботов в начальной конфигурации вне поверхности конечной конфигурации.

Время вычислений возрастает с увеличением числа роботов, т.к. увеличивается количество графов коллизий с циклами. Время затрачивается на переориентацию рёбер в графе и на обмен целевыми точками. В большинстве случаев был использован метод «разделяй и властвуй».

Рисунок 15 - Время вычисления алгоритма при большом числе объектов

Аналогично первому эксперименту, во втором эксперименте общая длина перемещений роя роботов увеличивается с увеличением количества объектов в этом рое.

Рисунок 16 - Общая длина перемещений роя роботов

1. Заключение

Еще раз отметим существование прочих сценариев комбинации разработанного нами инструментария. Мы ожидаем, что понимание критериев выбора между тем или иным сценарием появится после ряда компьютерных экспериментов.

Очевидным недостатком нашего подхода является низкая робастность метода. Поэтому в дальнейшем рекомендуется комбинировать предложенные алгоритмы с базовыми более традиционными подходами. Так, например, учет даже очень слабых взаимодействий между роботами существенно повышает робастность навигации.

*Список литературы*

[1] Balch, T., Social potentials for scalable multi-robot formations. // In: Proc. ICRA ’00. IEEE International Conference on Robotics and Automation, 2000, №1, С. 73–80.

[2] Bullo F., Cortez J., Martinez S. Distributed Control of Robotic Networks. Princeton University Press, 2009.

[3] Chen Y.Q., Wang Z. Formation control: a review and a new co nsideration. // In: Proc. IEEE Int. Conf. Intell. Robots Syst., 2005б С. 3181–3186.

[4] Hsieh M.A., Kumar V., Chaimowicz L. Decentralized controllers for shape generation with robotic swarms. // Robotica, 2008, №26 (5), С. 691–701.

[5] Yan X., Chen J., Sun D., Multilevel-based topology design and shape control of robot swarm. // Automatica, 2012, №48, С. 3122–3127.

[6] Schaare S., Dyskin A. V., Estrin Y., Arndt S., Pasternak E., Kanel-Belov A. Point loading of assemblies of interlocked cube-shaped elements // Internat. J. Eng. Sci., 2008, №46 (12), С. 1228–1238.

[7] Dyskin A.V., Estrin Y., Pasternak E., Khor H.C., Kanel-Belov A.J., The principle of topological interlocking in extraterrestrial construction // Acta Astronautica, 2005, №57 (1), С. 10–21.

[8] Dyskin A.V.,Estrin Y.,Kanel-Belov A.J., Pasternak E., Interlocking properties of buckyballs. // Physics Letters A, , 2003, №319, С. 373–378.

[9] Adler, A., de Berg, M., Halperin, D., Solovey, K., Eﬃcient Multi-Robot Motion Planning for Unlabeled Discs in Simple Polygons. // IEEE T. Automation Science and Engineering, 2015, №12 (4), С. 1309–1317.

[10] Abellanas, M., Bereg, S., Hurtado, F., Olaverri, A. G., Rappaport, D., Tejel J., Moving coins. // Computational Geometry: Theory and Applications, 2006, №34, С. 35–48.

[11] Bereg, S., Dumitrescu, A., and Pach, J., Sliding disks in the plane. // Int. J. Comput. Geometry Appl., 2008, №18, (5), С. 373–387.

[12] Dumitrescu, A., Jiang, M., On reconﬁguration of disks in the plane and related problems. // Computational Geometry: Theory and Applications, 2013, №46, 3, С. 191– 202.

[13] Vaidya P.M., Approximate Minimum Weight Matching on Points in k-Dimensional Space. // Algorithmica, 1989, №4, С. 569–583.

[14] Agarwal P.K., Efrat A., Sharir M., Vertical Decomposition of Shallow Levels in 3Dimensional Arrangements and Its Applications. // SIAM Journal on Computing, 1999, №29 (3), С. 912–953.

[15] Tarjan R. E. Depth-ﬁrst search and linear graph algorithms. // SIAM J. on Computing,1972, № 1 (2), С. 146–160.

[16] Jadhav S., Mukhopadhyay, A., Computing a centerpoint of a ﬁnite planar set of points in linear time. // Discrete and Computational Geometry, 1994, №12 (1), С. 291–312.

[17] Clarkson K., Eppstein D., Miller G., Sturtivant C., Teng S.-H., Approximating center points with iterated Radon points. // Int. J. Comp. Geometry and Applications, 1996, №6 (3), 357–377.

[18] Edelsbrunner H., Waupotitsch R, Computing a ham sandwich cut in two dimensions. // J. Symbolic Computing, 1986, № 2, С. 171–178.