Санкт–Петербургский государственный университет
Факультет математики и компьютерных наук

Добышев Артемий Николаевич

Выпускная квалификационная работа

Исследование методов и разработка алгоритмов построения траектории многозвенного манипулятора

Уровень образования: бакалавриат

Направление 02.03.01 «Математика и компьютерные науки»

Основная образовательная программа СВ.5152.2020 «Математика, алгоритмы и анализ данных»

Научный руководитель:

доцент, Факультет математики и компьютерных наук, СПбГУ

к.ф.-м.н., Яковлев Константин Сергеевич

Рецензент:

с.н.с., ФИЦ СПб РАН

к.т.н., Савельев Антон Игоревич

Санкт-Петербург

2024 г.

Содержание

Введение

Разработка роботов-манипуляторов (в дальнейшем просто манипуляторы) является бурно развивающимся направлением автоматизации и находит широкое применение в автомобильной, химической промышленности, машиностроении, сельском хозяйстве, в любой сфере, где работа человека может быть опасна.

Также манипуляторы служат в образовательных и развлекательных целях, целях индивидуального пользования.

Основной задачей для манипуляторов является “переместить конец манипулятора в заданное положение в пространстве из текущей конфигурации”. При помощи решения задачи обратной кинематики единственной в данной работе задачей манипулятора будет переместиться в одну из целевых конфигураций из заданной.

В пространстве, в котором расположен манипулятор (далее – рабочее пространство), могут находиться препятствия, контакта с которыми необходимо будет избегать. Но препятствия не получится описать явно. Это создает дополнительную сложность при решении поставленной задачи.

При этом в целях экономии времени и энергии важно, чтобы полученная траектория была построена эффективно и оптимально. Под оптимальностью траектории подразумевается, например, наименьший суммарный угол поворота роторов манипулятора, или же наименьшее время перемещения, и т.п.

Целью данной работы является исследование задачи построения траектории движения многозвенного манипулятора и разработка алгоритма ее решения.

Для достижения данной цели необходимо выполнить следующие задачи:

1) Обзор литературы.

2) Формальная постановка задачи.

3) Реализация метода планирования траектории многозвенного манипулятора на языке Python.

4) Проведение экспериментального сравнения с одним из стандартных методов решения данной задачи в виде бенчмаркинга.

По результатам работы был реализован алгоритм Opt-Informed-RRT*, который находится в открытом доступе по ссылке: https://github.com/Komment314/opt-informed-rrt-star.

Также было проведено сравнительное экспериментальное тестирование с алгоритмом Informed-RRT*, которое показало, что представленный в работе алгоритм после нахождения траектории эффективнее улучшает сходимость длины допустимой траектории к оптимальной.

1. Предметная область

1.1. Описание задачи

Каждое положение манипулятора в рабочем пространстве задается набором из нескольких величин, которые параметризованы степенями свобод манипулятора (обычно это углы поворота роторов), причем эти величины непрерывны и обычно ограничены. Такие наборы будем называть конфигурациями манипулятора. Каждая конфигурация либо допустима, либо нет, в случае если манипулятор сталкивается с собой или препятствиями, или нарушаются ограничения, связанные с этими величинами.

Перед манипулятором будет стоять следующая задача: из текущей конфигурации дотянуться до стакана цилиндрической формы для последующего захвата. Это означает, что ему нужно будет переместить свой конец в заданное положение в рабочем пространстве.

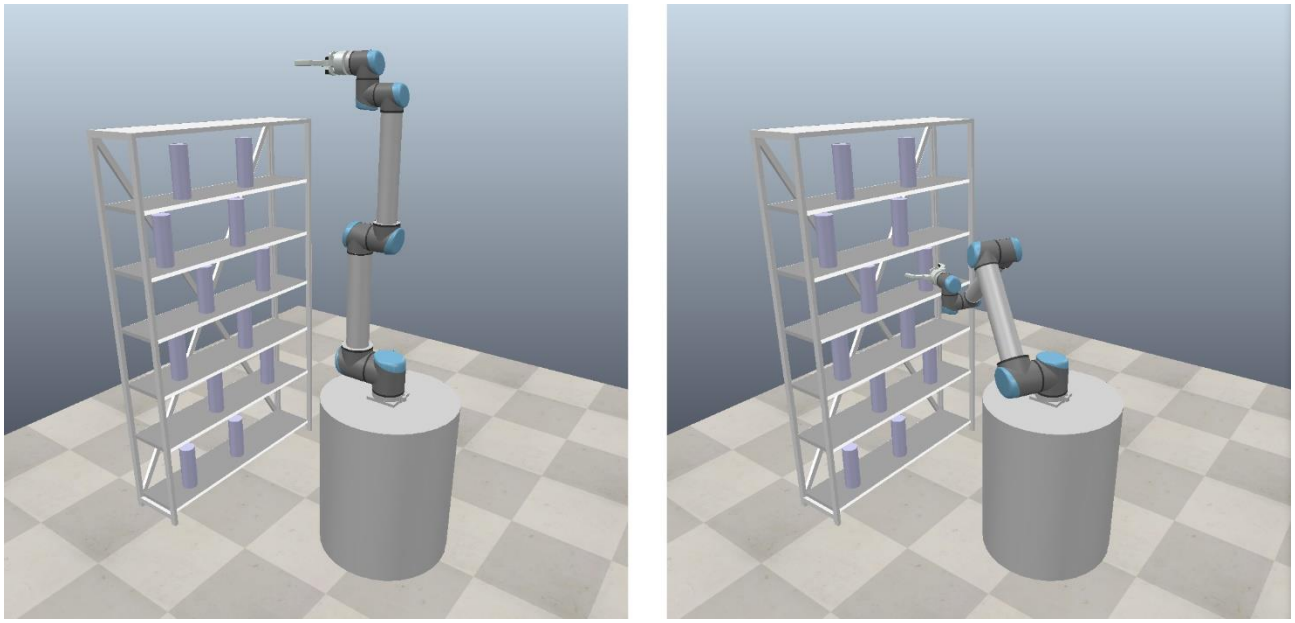


Рис 1. Расположение манипулятора в рабочем пространстве. Слева показана начальная конфигурация манипулятора, справа – конфигурация, в которой конец манипулятора перемещен к одному из стаканов

Задача “переместить конец манипулятора в заданное положение в рабочем пространстве из текущей конфигурации” с помощью решения проблемы

обратной кинематики перейдет в задачу “переместиться в одну из множества целевых конфигураций из заданной”.

Таким образом, поиск предстоит вести именно в многомерном непрерывном конфигурационном пространстве, учитывая потребность в оптимальности пути по некоему критерию.

Препятствия же находятся в рабочем пространстве, задаются там в явном виде и индуцируют препятствия в конфигурационном пространстве. Однако на данный момент не существует алгоритма, который бы хоть как-то отображал препятствия рабочего пространства в препятствия конфигурационного пространства в явном виде. Оттого задача усложняется, поскольку знания о среде получаются точечно: мы можем только узнать, является ли конкретная конфигурация допустимой или нет.

В связи с этим придется допустить некоторую дискретизацию, описанную в пункте 4 раздела 3.1.

1.2. Обзор литературы

Для решения поставленной задачи возможно использование различных методов.

Методы на основе графов аппроксимируют конфигурационное пространство равномерным полем из ячеек – вершин, в результате каждая либо свободна, либо занята препятствиями. Так создается граф, опираясь на полученные вершины и соединяя их по определенным правилам. После этого применяются алгоритмы поиска на графе, такие как алгоритм Дейкстры [1], A* [2] и т.п. Но такие методы редко используются для нашей задачи, потому что аппроксимация влечет за собой очевидную методическую погрешность, а сам метод страдает от проклятья размерности [3].

Методы на основе машинного обучения зачастую или обучают распределение препятствий или же свободного пространства, или обучают манипулятор на задачах для него самого или же для похожих по строению манипуляторов [4] [5].

Но данные методы сильно зависят от заранее подготовленных данных, а в совокупности с тем, что нейронные сети имеют в основе вероятностную структуру, а следовательно манипуляторы под их управлением становятся менее контролируемыми и более опасными для окружающих объектов и людей, применимость таких методов на практике в настоящий момент маловероятна.

Чаще в решениях данной задачи применяются сэмплирующие методы PRM [6], RRT [7], RRT-connect [8], Informed-RRT* [9], BIT* [10], а также PRM* и RRT* [11] и т.п. Они случайно генерируют точки конфигурационного пространства, по определенному правилу добавляют их в граф. Он может как быть деревом, так и содержать циклы. Граф постепенно расширяется и уплотняется, таким образом исследуется конфигурационное пространство и оптимизируются траектории пути. Алгоритмы без “*” занимаются только поиском траектории, а те, которые с “*” в названии, еще и оптимизируют ее со временем. Для таких алгоритмов обычно теоретически обосновывается и асимптотическая оптимальность: при стремлении числа добавленных в граф вершин к бесконечности длины путей в нем будут уменьшаться и стремиться к некоторой оптимальной длине. В работе будут рассматриваться асимптотически оптимальные алгоритмы, а конкретно Informed-RRT*, который генерирует точки на основе информации о длине пути. Преимуществом данных алгоритмов является менее выраженная зависимость от размерности и относительно малые затраты по времени и памяти. Но из-за большого размера пространства для сходимости к оптимуму может потребоваться большое количество генераций.

Цель данной работы заключается в предположении, что объединение сэмплирующего метода Informed-RRT* с каким-нибудь локальным оптимизатором пути помимо выгоды от прямой оптимизации так же эффективно сузит пространство исследования алгоритма и ускорит сходимость найденного пути к оптимальному. Однако время работы оптимизатора будет составлять заметную долю в общем времени работы алгоритма, значит итоговое количество сгенерированных образцов уменьшится.

Поэтому необходимо выяснить, возможен ли следующий компромисс: сужение исследуемого пространства через оптимизацию за счет уменьшения доли прямого сэмплинга.

2. Формальная постановка задачи

Рабочее пространством манипулятора обозначим за $W_{space} \subseteq R^3$.

Конфигурационным пространством манипулятора с N степенями свобод назовем назовем $C_{space} = [L_i, U_i]^N \subseteq R^N$, L и U – ограничения на значения конфигурации

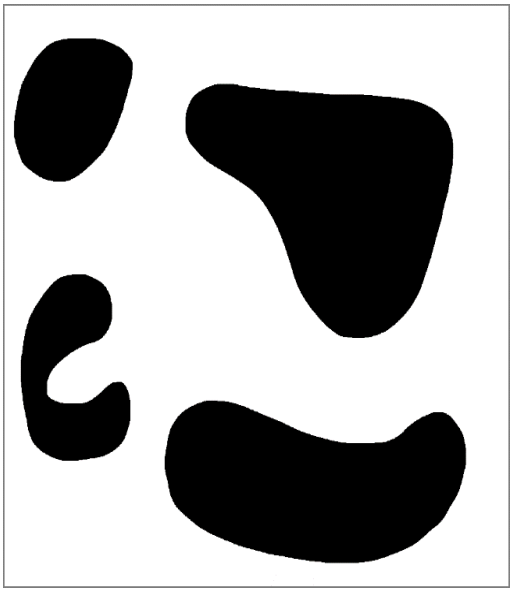


Рис. 2. Пример конфигурационного пространства $C_{space} \subseteq R^2$. C_{free} обозначено белым цветом, C_{obs} – черным.

В дальнейшем не будут вводиться какие-либо обозначения, связанные с рабочим пространством. Отметим лишь, что манипулятор, свободное место и препятствия в рабочем пространстве будут неким образом индуцировать в конфигурационном пространстве множества C_{free}, $C_{obs} \subseteq C_{space}$ такие, что $C_{free} \cap C_{obs} = \emptyset$, $C_{free} \cup C_{obs} = C_{space}$. C_{obs} – множество недопустимых конфигураций (пересечения тела манипулятора с препятствиями и самопересечения тела манипулятора), C_{free} – множество допустимых конфигураций.

Конфигурацию определим как $C = (c_1, c_2, \dots, c_N) \in C_{space}$. Из определений следует, что для любая конфигурация C принадлежит либо множеству C_{free}, либо C_{obs}.

За $target \in W_{space}$ обозначим точку рабочего пространства, в которую хотим переместить конец манипулятора $tip = Tip(C) \in W_{space}$. Функция $Tip(C)$ определяет алгоритм, который вычисляет положение конца. В работе функция $Tip(C)$ будет представлена как функция, решающая задачу прямой кинематики.

В общем случае допустимой траекторией манипулятора из конфигурации C_{start} в C_{goal} будет называться непрерывное отображение $f: [0, 1] \rightarrow C_{free}$ такое, что $f(0) = C_{start}$, $f(1) = C_{goal}$.

Далее в работе под допустимой траекторией будет пониматься та, которая порождена последовательностью допустимых конфигураций $(C_1, C_2, \dots, C_K)$ для некоторого K, а именно такая траектория $f: [0,1] \rightarrow C_{free}$, что существует набор точек $(d_1 = 0, d_2, \dots, d_K = 1)$, что $f([d_i, d_{i+1}]) = [C_i, C_{i+1}]$. Посему понятие "допустимая траектория $tr = (C_1, C_2, \dots, C_K)$" будет отождествлено с "допустимая траектория tr, порожденная последовательностью конфигураций $(C_1, C_2, \dots, C_K)$".

Для оптимизации допустимой траектории $tr = (C_1, C_2, \dots, C_K)$ введем функцию $Cost(tr) = \sum_{i=1}^{n-1} dist\ (C_i, C_{i+1})$ для некоторой функции $dist$, которая оценивает расстояние между конфигурациями. В работе же $dist(C_1, C_2) = \left|\left|C_1 - C_2\right|\right|_2$.

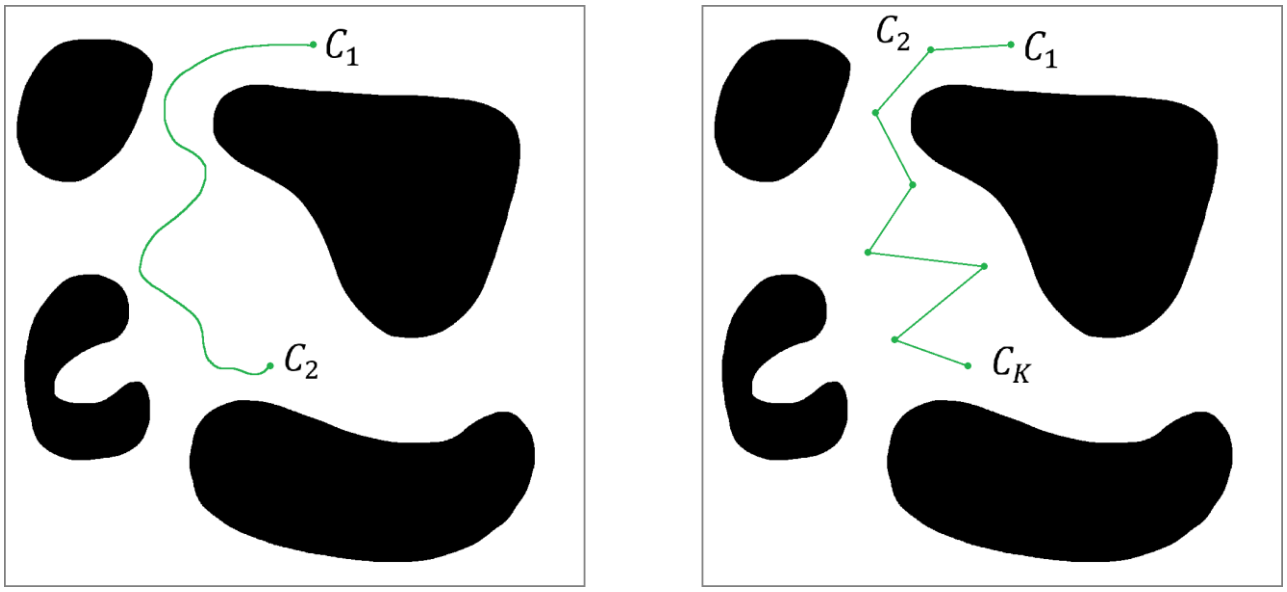


Рис. 3. Допустимые траектории для $C_{space} \subseteq R^2$. На левом рисунке допустимая траектория в общем случае. На правом – допустимая траектория, порожденная последовательностью конфигураций $(C_1, C_2, \dots, C_K)$

Задача поиска оптимальной траектории манипулятора от стартовой конфигурации C_{start} до целевой точки $target$ состоит в поиске допустимой траектории $tr_{opt} = (C_1, C_2, \dots, C_K)$ такой, что $C_1 = C_{start}$, $Tip(C_K) = target$ и для любой другой траектории tr верно $Cost(tr_{opt}) \leq Cost(tr)$.

3. Метод решения задачи

Отметим, что для соблюдения условия $Tip(C_K) = target$ нужно найти такие $C_K \in Tip^{-1}(target)$. Описанная задача называется задачей обратной кинематики, она нетривиальна, общего универсального аналитического ее решения нет. В дальнейшем будет показано, каким именно способом она будет решена. Посему для задачи поиска оптимальной траектории это условие нивелируется, и делается сужение к поиску траектории между конфигурациями C_{start} и C_{goal}. Также конфигурации иногда будут обозначаться буквой x.

3.1. Описание работы алгоритма RRT.

Для начала рассмотрим принцип работы самого базового алгоритма – RRT [7], на котором и основан алгоритм Informed-RRT*.

Algorithm 1 RRT

$iteration = 1$
$goal\_found = False$
while $iteration < maxiter$ **and** $goal\_found == False$ **do**
    $x_{sample} = Sample()$
    $x_{nearest} = Nearest(Tree, x_{sample})$
    $x_{new} = Steer(x_{sample}, x_{nearest})$
    if $CoordsConnectivity(x_{nearest}, x_{new}) = True$ **then**
        $V.add(x_{new})$
        $x_{new}.parent = x_{nearest}$
        if $x_{new} \in Goals$ **then**
            $goal\_found = True$
return $Tree, goal\_found$

1) Входные данные.

Алгоритм на вход получает C_{start}, $Goals = (C_{goal_1}, \dots, C_{goal_k})$ и $maxiter$.

2) Генерация образцов. Строка **4**.

В начале цикла происходит генерация образца $x_{sample} \sim U(C_{space})$. Помимо этого, для более быстрого поиска пути до целей будет использоваться техника $goal\ biasing$. Если цель еще не найдена, то при каждом вызове $Sample()$ с определенной вероятностью $goal\_bias\_prob$ вместо случайной генерации, описанной выше, новый образец будет браться из $Goals$ по определенному распределению.

3) Поиск наилучшего соседа и “steering”. Строки **5-6**.

Далее выполняется поиск ближайшей вершины к x_{sample} в дереве – $x_{nearest}$. А потом происходит “steering” – сближение x_{sample} к дереву, а конкретно к $x_{nearest}$, если x_{sample} далеко от нее. $x_{new} = Steer(x_{sample},\ x_{nearest}) = \underset{x \in B_{cut\_radius}(x_{nearest})}{\operatorname{argmin}} dist(x,\ x_{sample})$, B_{cut_radius} – замкнутый шар радиуса cut_radius, cut_radius – гиперпараметр алгоритма.

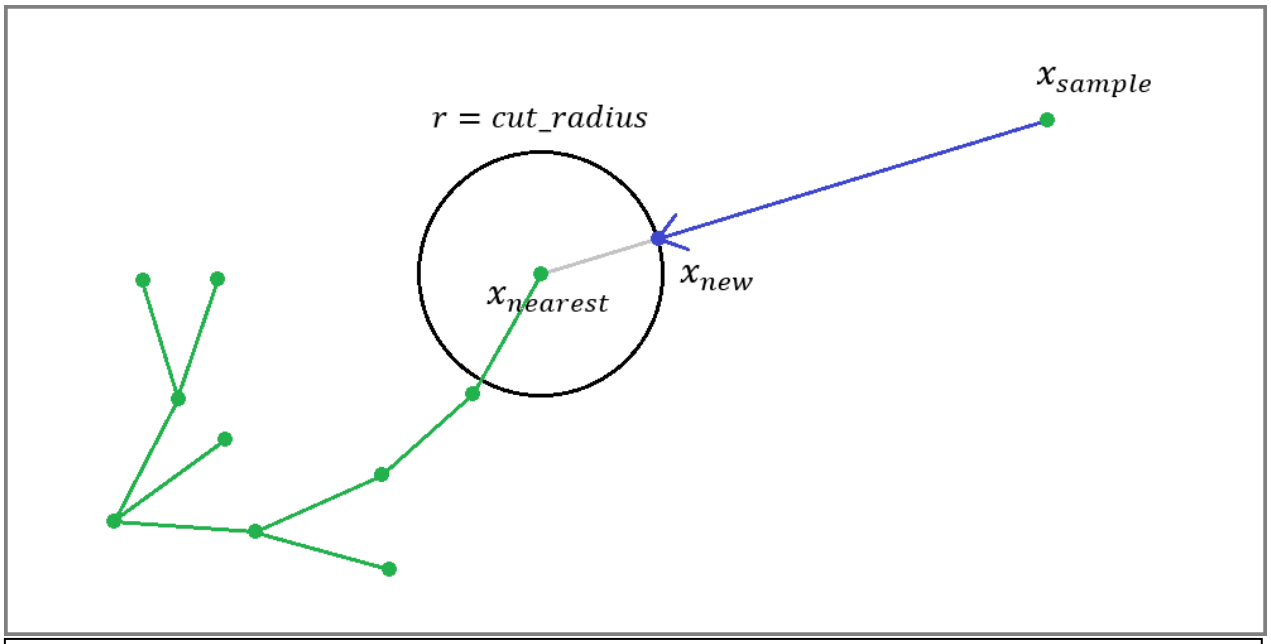


Рис 4. Steering. Дерево обозначено зеленым. Точка x_{sample} сгенерирована далеко от дерева, на расстоянии большем cut_radius, и приближена к ближайшей вершине дерева $x_{nearest}$ в виде точки x_{new}.

4) Присоединение к дереву. Строки **7 - 11**.

После приближения к дереву определяется достижимость x_{new} из $x_{nearest}$ с помощью функции $CollisionFree()$, т.е. проверяется, лежит ли отрезок $[x_{nearest}, x_{new}]$ в C_{free}.

На этом этапе используется единственное важное допущение для дискретизации в непрерывном алгоритме. За $P_{delta} = (x_{nearest} = x_1, x_2, \dots, x_k = x_{new})$ обозначим такое разбиение, что $\max\limits_{i \in \{1,2,\dots,k-1\}} |x_i - x_{i+1}| = delta$ – гиперпараметр, $\forall i, j \in \{1, 2, \dots, k-1\}\ |x_i - x_{i+1}| = |x_j - x_{j+1}|$ и k – наименьшее. Будем считать, что отрезок $[x_{nearest}, x_{new}]$ полностью принадлежит C_{free} тогда и только тогда, когда каждая точка разбиения P_{delta} принадлежит C_{free}. Чем больше, $delta$, тем быстрее будут выполняться все проверки на достижимость и присоединение к дереву. Чем меньше $delta$, тем точнее будут учитываться препятствия.

Если x_{new} недостижима из $x_{nearest}$, то x_{new} добавляться в дерево не будет. Если же достижима, то она добавится в дерево.

5) Возвращаемые переменные.

Алгоритм по окончании возвращает дерево и, если какая-то цель была достигнута, то положительный результат, иначе – отрицательный.

3.2. Описание работы алгоритма Informed-RRT*.

После описания работы базового алгоритма RRT можно перейти к рассмотрению работы алгоритма Informed-RRT* [9].

Algorithm 2 Informed-RRT*

1: **for** $iteration = 1, 2, \ldots, maxiter$ **do**
2: $x_{sample} = Sample(x_{start}, x_{goal}, c_{best})$
3: $x_{nearest} = Nearest(Tree, x_{sample})$
4: $x_{new} = Steer(x_{sample}, x_{nearest})$
5: **if** $CoordsConnectivity(x_{nearest}, x_{new}) = True$ **then**
6: $X_{near} = Near(x_{new})$
7: **for** $x_{neighbor} \in X_{near}$ **do**
8: **if** $Cost(x_{neighbor}) + dist(x_{neighbor}, x_{new}) < Cost(x_{nearest}) + dist(x_{nearest}, x_{new})$ **then**
9: **if** $CoordsConnectivity(x_{neighbor}, x_{new}) = True$ **then**
10: $x_{nearest} = x_{neighbor}$
11: $V.add(x_{new})$
12: $x_{new}.parent = x_{nearest}$
13: **if** $x_{new} \in Goals$ **then**
14: $goal\_found = True$
15: $goal\_index = |V| - 1$
16: $c\_best = Cost(x_{new})$
17: **for** $x_{neighbor} \in X_{near}$ **do**
18: **if** $Cost(x_{new}) + dist(x_{new}, x_{neighbor}) < Cost(x_{neighbor})$ **then**
19: $x_{neighbor}.parent = x_{new}$
20: **return** $Tree,\ goal\_found,\ goal\_index$

1) Входные данные.

Алгоритм на вход получает C_{start}, $Goals = (C_{goal_1}, \ldots, C_{goal_k})$, $maxiters$ и гиперпараметры.

2) Генерация образцов. Строка **2**.

В начале цикла происходит генерация образца x_{sample} – случайного вектора из C_{space}. Если никакая целевая конфигурация еще не добавлена в дерево, то $x_{sample} \sim U(C_{space})$.

Как только какая-то целевая конфигурация C_{goal} добавляется в дерево, то его расширение в сторону остальных целевых конфигураций прекращается,

дальнейшая работа алгоритма будет направлена на оптимизацию траектории из C_{start} в C_{goal}. Раз C_{goal} добавлена в дерево, то существует некая траектория tr из C_{start} в C_{goal}, имеющая стоимость c_{best}. Тогда заметим, что $dist(C_{start}, C) + dist(C, C_{goal}) \leq c_{best}$ $\forall C \in tr$, т.е. вся траектория tr лежит в эллипсе $X_{c_{best}}$ с фокусами C_{start} и C_{goal} и большой полуосью, равной ${c_{best}}/{2}$. Соответственно, все лучшие траектории будут также находиться внутри него, а посему для улучшения траектории не имеет смысла генерировать образцы вне его. В данном случае $x_{sample} \sim U(X_{c_{best}})$.

В обоих случаях функция $Sample()$ обязательно возвращает $x_{sample} \in C_{free}$, т.е. попытки генерации происходят, пока образец не будет принадлежать C_{free}, а до тех пор, все образцы из C_{obs} будут сохраняться для дальнейшего использования в оптимизаторе.

На данном этапе аналогично RRT будет использоваться техника $goal\ biasing$, которой нет в оригинальном алгоритме.

3) Поиск наилучшего соседа и “steering”. Строки **3-4**.

Этот этап происходит аналогично алгоритму RRT.

4) Присоединение к дереву и первое улучшение оптимальности. Строки **5 - 16**.

Добавление точки x_{new} к дереву происходит аналогично алгоритму RRT. Но перед ним находится множество соседей X_{near} в определенном радиусе $near\_radius$. Согласно работе Сертака Карамана и Эмилио Фраццоли [11], для поддержки оптимальной сходимости достаточно, чтобы $near\_radius \geq \min\{(2 * (1 + \frac{1}{n\_dims}))^{\frac{1}{n\_dims}} * \left(\frac{\mu(X_{free})}{\zeta_{n\_dims}}\right)^{\frac{1}{n\_dims}} * \left(\frac{\log(|V|)}{|V|}\right)^{\frac{1}{n\_dims}}, cut\_radius\}$, где $\mu(X_{free})$ – объем рассматриваемого свободного пространства на данный момент, а ζ_{n_dims} – объем единичного шара размерности n_dims.

После этого для каждого соседа $x_{neighbor} \in X_{near}$ проверяется, не выгодней ли в плане стоимости “подключить” x_{new} к нему, нежели к изначальной $x_{nearest}$. Если это так, то $x_{nearest}$ заменяется на $x_{neighbor}$. Теперь уже x_{new} добавляется в дерево с родителем $x_{nearest}$. Таким образом, x_{new} присоединена к дереву самым выгодным способом (среди соседей).

Если на момент добавления $x_{new} \in Goals$, то цель найдена, поиск остальных целей прекращается.

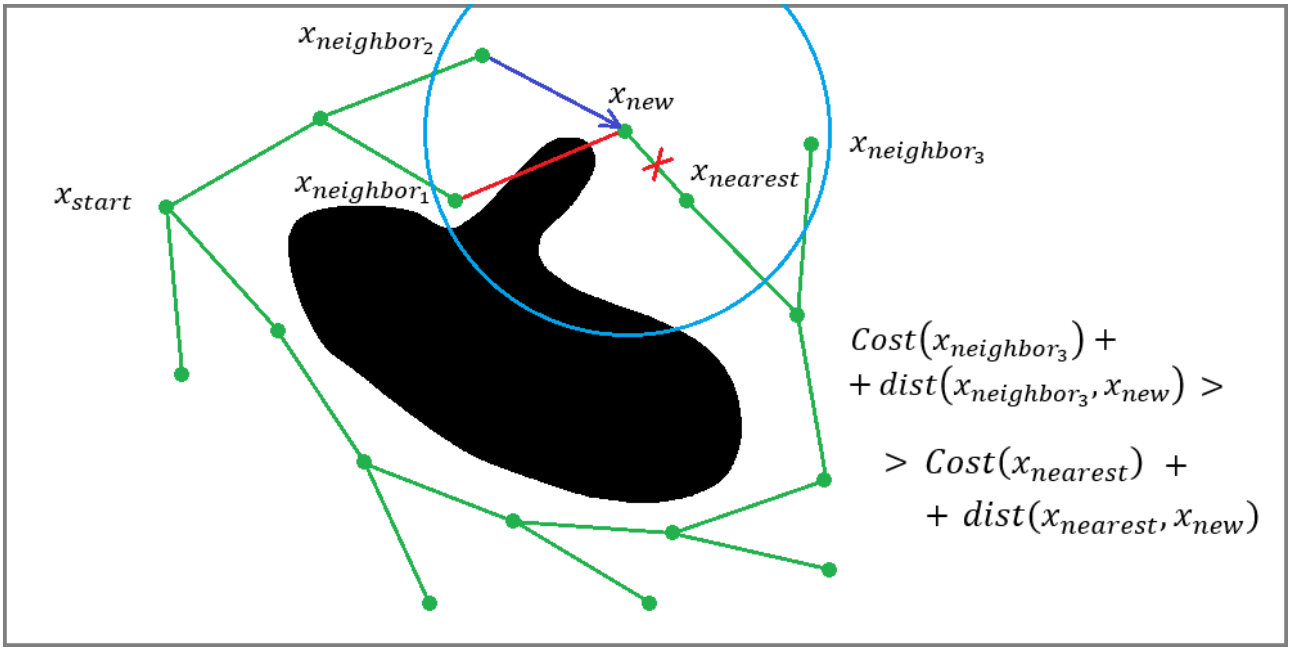


Рис 5. Первое улучшение оптимальности. Вершина x_{new} изначально подключена к $x_{nearest}$, но при проверке соседей вершины x_{new} оказалось, что она недостижима из $x_{neighbor_1}$, к $x_{neighbor_3}$ подключать ее невыгодно, а $x_{neighbor_2}$ и достижима, и более выгодна, нежели $x_{nearest}$.

5) Второе улучшение оптимальности. Строки **17 – 19**.

Далее для каждой вершины из X_{near} проверяется, не выгоднее ли попасть в нее через только что добавленную в дерево x_{new}, а не через текущего родителя. Если так, то родитель переписывается и становится x_{new}. Из-за этого каждая вершина, соседняя с только что добавленной в дерево, будет подключена в нем самым оптимальным способом (опять же, среди соседей).

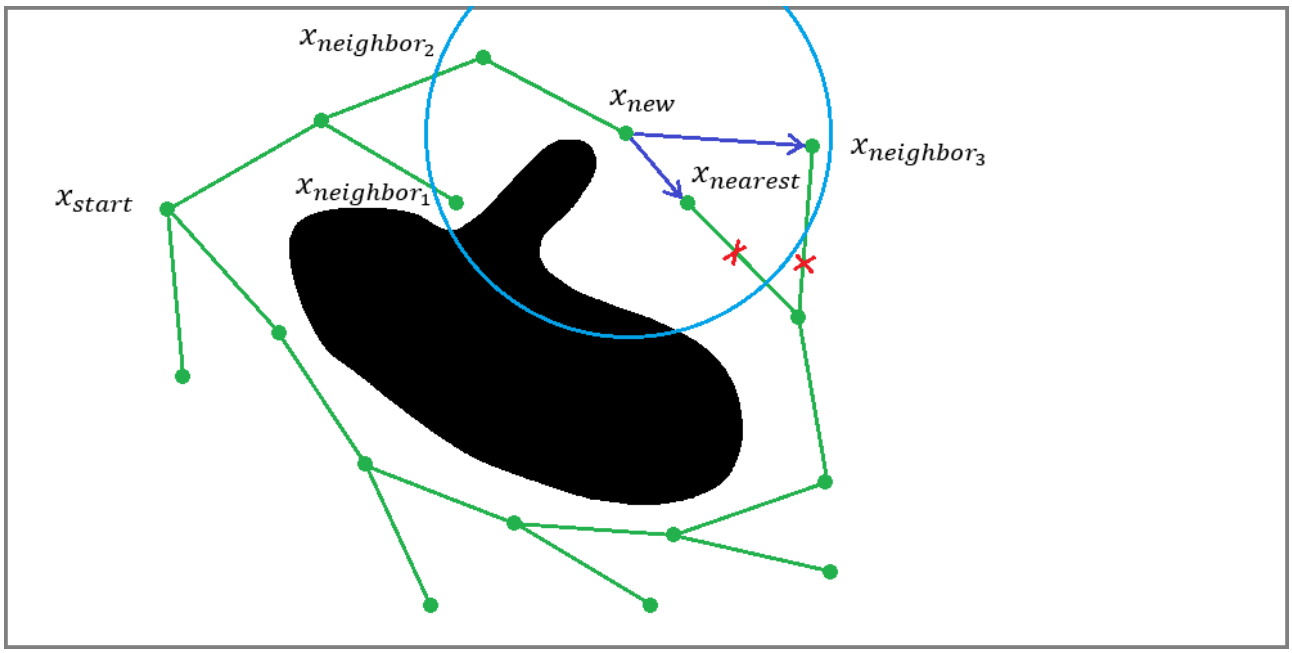


Рис 6. Второе улучшение оптимальности. В вершины $x_{nearest}$ и $x_{neighbor_3}$ выгодней попасть через x_{new}, а не через их текущих родителей. Их родители будут заменены на x_{new}.

6) Возвращаемые переменные.

По истечении числа итераций алгоритм возвращает дерево, результат поиска и индекс целевой вершины в дереве.

3.3. Описание работы алгоритма Opt-Informed-RRT*.

Теперь же изучим алгоритм Opt-Informed-RRT*. Он основан на оригинальном алгоритме Informed-RRT*, соответственно все доказательства из оригинальной статьи остаются в силе.

Пункты с 1 по 5 (соответственно, строчки с 1 по 19) аналогичны алгоритму Informed-RRT*.

6) Оптимизация (Подробнее в 3.4.). Строки **20 - 23**.

Если выполнены условия (в реализации оптимизация применяется, если много итераций прошло без нее, и, конечно, только тогда, когда цель уже найдена), то к

текущей траектории из C_{start} в C_{goal} применяется метод оптимизации, который возвращает opt_result (успешно или нет), c_{best_opt} (длину оптимизированной траектории, равна ∞, если не успешно) и саму последовательность точек opt_path (равна $None$, если не успешно). После этого c_{best} обновляется как $c_{best} = min(c_{best}, c_{best\_opt})$, потому как если оптимизация успешна (т.е. $c_{best\_opt} < c_{best}$), то оптимизированная траектория находится в меньшем эллипсе, а значит не имеет смысла генерировать образцы вне его. А opt_path сохраняется, потому что невозможно сказать, успеет ли алгоритм сгенерировать образцы для улучшения стоимости пути уже меньше, чем c_{best_opt}. Именно этот этап и должен ускорить сходимость к оптимуму.

7) Возвращаемые переменные.

По истечении числа итераций алгоритм возвращает дерево, результат поиска, индекс целевой вершины в дереве и лучшую траекторию из C_{start} в C_{goal}, полученную через оптимизатор.

3.4. Описание работы оптимизатора.

Стоит обратить отдельное внимание на процесс работы оптимизатора. Для этого перейдем к описанию функции $Opt()$ на примере $C_{space} \subseteq [L_i, U_i]^2 \subseteq R^2$. Выполнение оптимизации состоит из трех этапов:

1) Генерация образцов вдоль траектории.

Пусть уже найдена траектория из x_{start} в x_{goal}. Она разбивается на точки наподобие функции $CoordsConnectivity$, только со своим параметром нормы разбиения. Далее объявляется, что вдоль этой новой траектории будет сгенерировано $n\_samples$ образцов в радиусе $sample\_raduis$. Чем меньше $sample\_raduis$, тем больше вероятность допустимости оптимизированной траектории. Чем больше $sample\_raduis$, тем больше потенциальное улучшение стоимости траектории.

Потом проделывается следующая операция. Для каждой точки разбиения траектории, т.е. для конфигурации манипулятора, подсчитывается вероятность генерации точек именно около нее. Эта вероятность обратно пропорциональна расстоянию от манипулятора до текущих препятствий в рабочем пространстве в какой-то заранее заданной степени, большей 1. Т.е. если конфигурация далеко от препятствий, не имеет смысла генерировать вокруг нее много сэмплов для поиска точек из C_{obs}. Исходя из этих вероятностей и значения $n\_samples$, вокруг каждой точки разбиения траектории генерируются образцы.

Если они принадлежат C_{obs}, то они запоминаются для дальнейшего использования. Таким образом, получается исследовать какое-то пространство “около” траектории. При этом, к этим точкам добавятся абсолютно все те, которые получены в результате генерации образцов в процессе построения дерева. Это дает дополнительную информацию о препятствиях “не рядом” с траекторией.

2) Перевод образцов в эллипсоидные препятствия.

Поскольку для оптимизации с ограничениями препятствия должны быть гладкими, то обычно для этого используются сферические или эллипсоидные препятствия. В нашем же распоряжении есть только множество отдельных точек из C_{obs}. Самым простым вариантом будет сказать, что каждая точка траектории должна быть на расстоянии d_{safe} от каждого сгенерированного образца из C_{obs}, иными словами, представить каждый образец как сферу-препятствие с радиусом d_{safe}. Но таких образцов может быть несколько сотен или тысяч, что недопустимо замедляет работу оптимизатора. Стало быть, количество препятствий нужно уменьшить.

В этом поможет кластеризация и решение задачи минимального покрывающего эллипсоида. Сначала все образцы разбиваются на $n\_init\_clusters$ кластеров, в работе же использован метод алгомеративной кластеризации. Затем каждый кластер преобразуется в эллипсоид через решение задачи поиска минимального

покрывающего эллипсоида. Однако стоит заметить, что построенные эллипсоиды могут перекрывать точки траектории, это создает проблемы при использовании оптимизации.

Для решения этой проблемы для каждого кластера в порядке убывания количества вершин проверяется, содержит ли построенный вышеописанным способом эллипсоид точки разбиения траектории. Если не содержит, то эллипсоид добавляется в качестве препятствия, а если содержит, то кластер, порождающий его, разбивается на два меньших таким же алгоритмом кластеризации. Новые два кластера добавляются в очередь проверки. Так происходит до тех пор, пока либо все кластеры не закончатся, либо не будет достигнуто ограничение на количество препятствий.

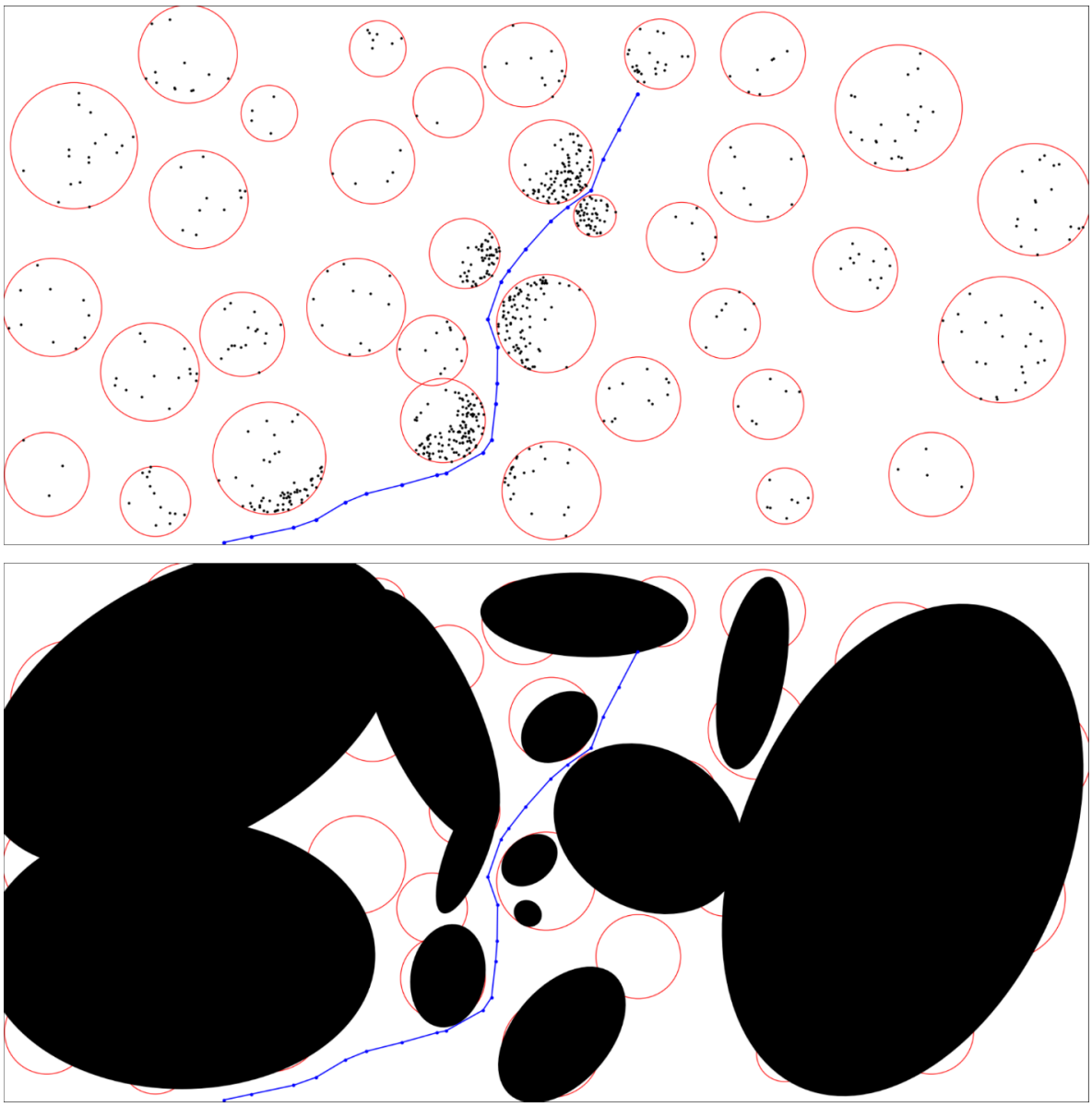


Рис 7. Построение эллипсоидных препятствий вдоль траектории на примере двухмерного конфигурационного пространства. Синим показана траектория, красным – оригинальные препятствия, которые алгоритм не знает. Сверху показаны накопленные за время работы алгоритма препятствия, а также новые сгенерированные препятствия вдоль траектории. Снизу показаны итоговые эллипсоидные препятствия для дальнейшей оптимизации.

3) Решение оптимизационной задачи.

После предыдущих шагов мы имеем траекторию из x_{start} в x_{goal} и набор эллипсоидных препятствий, которых необходимо избегать. Остается подобрать точки на траектории (переменные) таким образом, чтобы они не находились внутри препятствий (чтобы выполнялся ряд условий). Таким образом получается типичная задача оптимизации с условиями.

Но для нее нужен будет конечный набор переменных, по которым будет вестись оптимизация, а траектории принадлежит бесконечное число точек. Посему для этого вновь может быть применено разбиение траектории, но уже с новой нормой. Чем меньше эта норма, тем больше точек будет на пути, а значит увеличится вероятность допустимости оптимизированной траектории, но стоит заметить, что время оптимизации при этом увеличивается. А чем она больше, тем больше потенциальное улучшение стоимости траектории.

В такой постановке получается классическая задача нелинейного программирования:

$$\min f(x) \text{ при ограничениях } h_k(x) \geq d_k, k \in \{1, 2, \dots, n\_ineq\}$$

В данной задаче x – это разбитая траектория, количество итоговых переменных будет равно $n\_dots * n\_dims$.

$f(x)$ – функция стоимости траектории, но отметим, что у функции $Cost(tr) = \sum_{i=1}^{n-1} dist\,(C_i, C_{i+1})$, где $dist\,(C_i, C_{i+1}) = \left|\left|C_{i+1} - C_i\right|\right|_2$, зачастую дольше вычисляются производные и якобиан. Посему на практике обычно минимизируется не сумма норм, а сумма квадратов норм. Итого, $f(x) = \sum_{i=1}^{n-1}\left|\left|C_{i+1} - C_i\right|\right|_2^2$.

Набор $h_k(x)$ – функции ограничения. В нашей ситуации они имеют вид $(x_i - c_j)A_j(x_i - c_j)^T \geq 1$, где x_i – точка траектории, A_j, c_j – матрица и вектор, задающие эллипсоидное препятствие.

Сразу отметим, что если все переменные непериодические, т.е. если никакой ротор манипулятора не может проворачиваться неограниченное число раз, то функции $f(x)$ и $h_i(x)$ являются выпуклыми, посему видится возможным использование выпуклой оптимизации для нахождения глобального оптимума.

Однако в силу недостаточной информации обо всех препятствиях конфигурационного пространства и большой эмпирической вероятности того,

что полученная в ходе выпуклой оптимизации траектория не была бы допустимой, было решено отказаться в работе от ее применения и решать задачу как задачу нелинейного программирования.

Для решения поставленной задачи был использован популярный квази-ньютоновский метод “SLSQP”, описанный Дитером Крафтом [12] и реализованный в библиотеке *scipy*. По ходу оптимизации алгоритм запоминает траекторию на каждой итерации и возвращает первую, начиная с конца, допустимую траекторию. Если же такой траектории нет, то алгоритм возвращает отрицательный результат оптимизации.

Стоит заметить, что работа оптимизатора может быть гибко настроена через изменение следующих величин:

- Параметр нормы разбиения из пункта 1. Чем меньше норма, тем точнее будет учитываться траектория относительно препятствий. Чем норма больше, тем быстрее будет работать оптимизатор.
- Количество образцов $n\_samples$ из пункта 1 и максимальное количество эллипсов из пункта 2. Чем они больше, тем точнее будут учитываться препятствия. Чем они меньше, тем быстрее будет работать оптимизатор.
- Количество итераций самого алгоритма SLSQP из пункта 3 и частота вызовов непосредственно влияют на время работа оптимизатора.

4. Тестирование

4.1. Постановка задач в симуляторе Coppeliasim.

Coppeliasim – это программное обеспечение для моделирования роботов-манипуляторов и проведения симуляций. В нем можно создавать модели роботов и программировать их движения.

Для работы с алгоритмами была создана специальная сцена в Coppeliasim, изображенная на рисунке 8. В ее центре на цилиндрической подставке расположен манипулятор UR10, имеющий 6 степеней свобод, с прикрепленной к нему рукой-наконечником. Напротив него на шкафу с полками находятся 12 стаканов в форме цилиндра. Задачей манипулятора будет переместить руку-наконечник к одному из стаканов для последующего его захвата. Отметим, что со стаканами также нельзя будет сталкиваться, как и в реальной жизни, этот факт усложнит конфигурационное пространство.

Coppeliasim имеет свои внутренние алгоритмы для различных целей и задач. Например, возможно проверить, пересекается ли в данной конфигурации манипулятор с самим собой и всеми препятствиями в рабочем пространстве, также можно узнать расстояние от манипулятора до любого из препятствий в рабочем пространстве. Для решения задачи обратной кинематики был выбран внутренний алгоритм, основанный на классической схеме псевдо-обратного якобиана.

Таким образом, с использованием Coppeliasim мы имеем все необходимое для корректной работы алгоритмов.

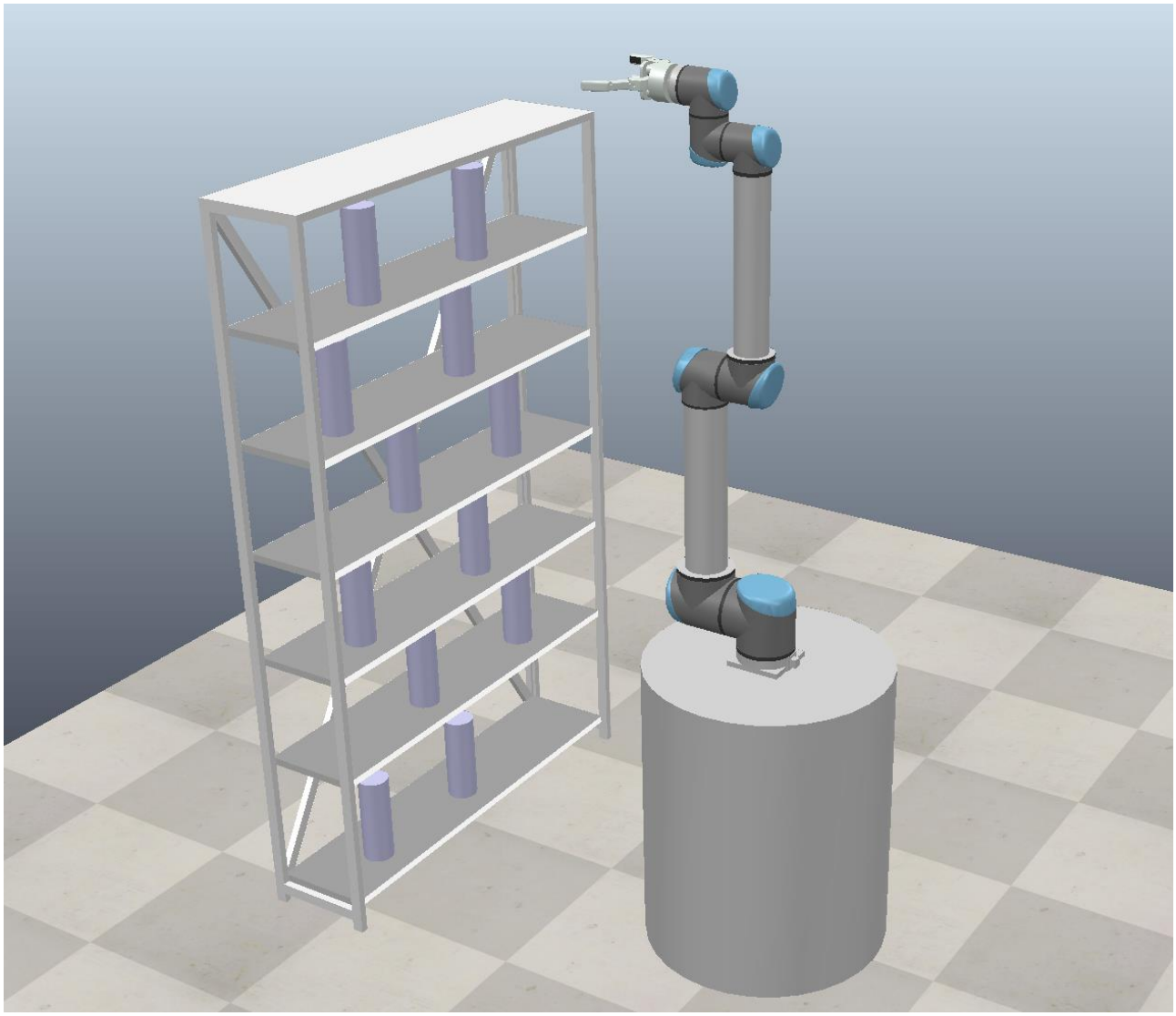


Рис 8. Сцена для тестирования в Coppeliasim.

4.2. Проведение тестов в симуляторе Coppeliasim.

Поскольку до момента нахождения целевой конфигурации оба алгоритма работают аналогично, целесообразным рассматривается их сравнение только после ее нахождения и при полностью совпадающем построенном дереве для исследования пространства.

Для этого выбрана следующая схема:

1) Для каждого из 12 стаканов ищется 5 начальных деревьев, построенных алгоритмом Informed-RRT* ровно на момент нахождения допустимой траектории от начальной конфигурации до одной из 100 целевых конфигураций, которые получены через решение задачи обратной кинематики. Соответственно, для каждого из этих деревьев цель уже найдена, и есть своя траектория со своей стоимостью, готовая к улучшению стоимости.
2) Для каждой из $12 * 5 = 60$ начальных траекторий проводится 20 запусков каждого алгоритма до тех пор, пока каждый не отработает 1000 итераций и 2 минуты. Это означает, что если алгоритм уже сделал 1000 итераций, то он будет работать, пока не истекут 2 минуты. А если же алгоритм проработает 2 минуты, но не сделает 1000 итераций, он будет работать до момента, пока не пройдут 1000 итераций. Этот момент замедлит тестирование примерно в 1.5 раза. В течение работы каждого алгоритма будет сохранена следующая информация на каждой итерации в виде списка: $[n\_iter,\ time,\ c_{best}]$.
3) Все полученные данные впоследствии нормализуются для преобразования в графики, параметризованные итерациями и временем.

Ожидаемое время тестирования: ~ 120 часов.

Для тестирования был использован стационарный компьютер с процессором AMD Ryzen 5 7600X и видеокартой NVIDIA GeForce RTX 3060Ti.

С исходным кодом и результатами экспериментов можно ознакомиться в репозитории по ссылке: https://github.com/Komment314/opt-informed-rrt-star.

4.3. Результаты тестирования.

В данном разделе будут представлены 5 примеров вариантов работы с полученными в ходе тестирования данными.

1) Прямая визуализация для каждой начальной траектории.

Для каждой из 60 начальных траекторий среди 20 проведенных запусков как Informed-RRT*, так и Opt-Informed-RRT* считаются следующие величины: $default(tree_i,\ test_j, t) = d(tree_i, test_j, t) = d_{i,j}(t)$ – значение стоимости траектории при запуске j и начальной траектории i в момент времени t ($t \in [0, 120]$ секунд) для Informed-RRT* ($i \in \{1, 2, \dots, 60\}, j \in \{1, 2, \dots, 20\}$), $m_{i,j}(t)$ – то же самое, но для Opt-Informed-RRT*.

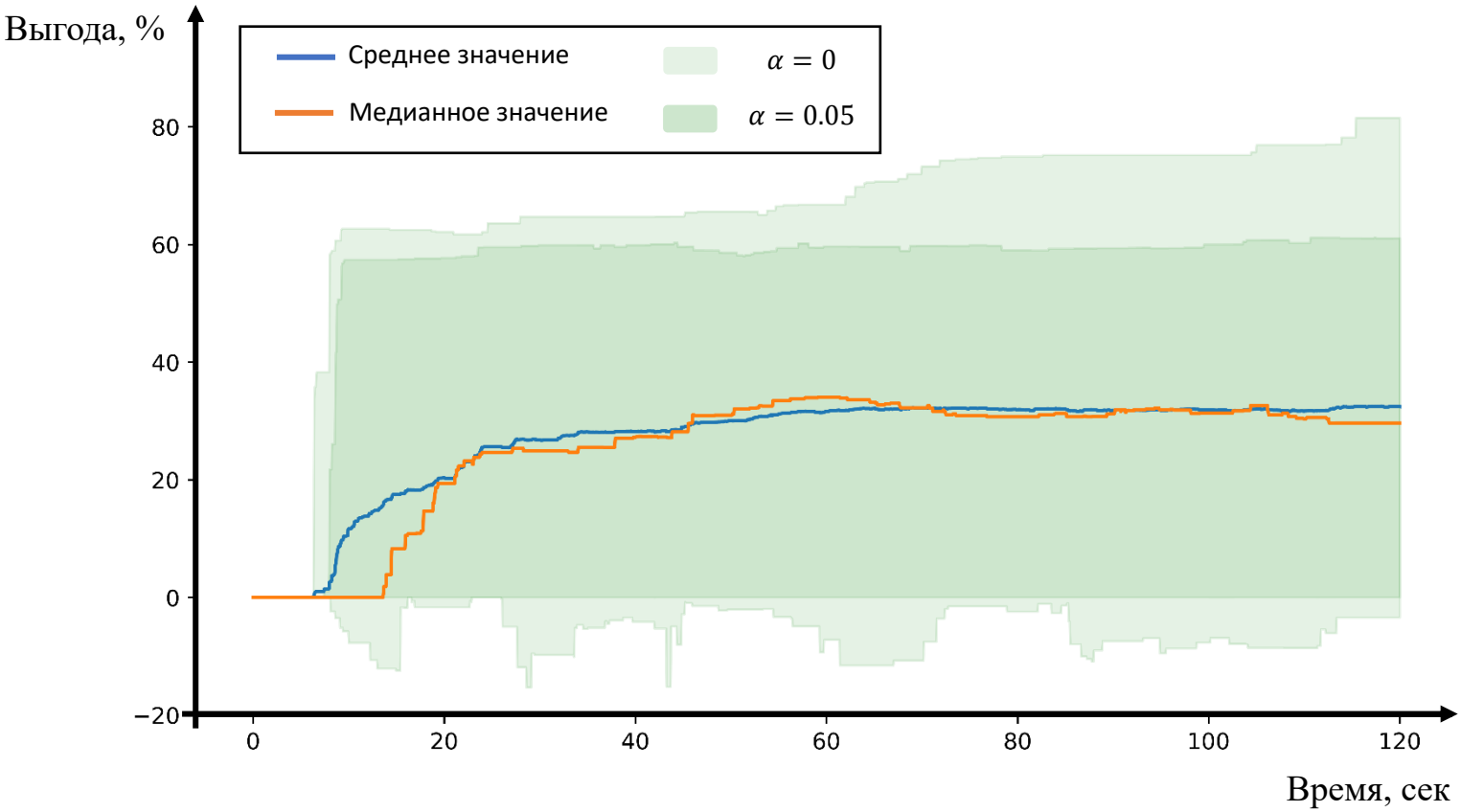
На первом графике будут изображены обе величины $d_{i,j}(t)$ и $m_{i,j}(t)$ течением времени, а на втором – $\left(\underset{j}{mean}\, d_{i,j}(t) - \underset{j}{mean}\, m_{i,j}(t)\right) / \underset{j}{mean}\, d_{i,j}(t) * 100\%$. $Mean$ берется по переменной j как усреднение каждой траектории по 20 запускам, а значит величины параметризованы только i, т.е. графиков будет 120. Вышеописанная величина показывает, на сколько процентов в данный момент времени Opt-Informed-RRT* “дешевле” в плане стоимости траектории Informed-RRT*. С соответствующими графиками можно ознакомиться в приложении.

2) Итоговый относительный выигрыш в процентах.

Для вычисления этой величины необходимо только посчитать $\underset{i}{median}(\left(\underset{j}{mean}\, d_{i,j}(t) - \underset{j}{mean}\, m_{i,j}(t)\right) / \underset{j}{mean}\, d_{i,j}(t) * 100\%)$ – очередное усреднение, только уже и по всем траекториям. Медиана используется для устойчивости к выбросам.

Читать данный график следует так: “в момент времени x алгоритм Opt-Informed-RRT* имеет стоимость пути на y % меньше в сравнении с Informed-RRT*”

Представим итоговый график, выраженный в виде среднего и медианного значений, а также центральных 100% и 90% доверительных интервалов в каждый момент времени t, изображенный на рисунке.



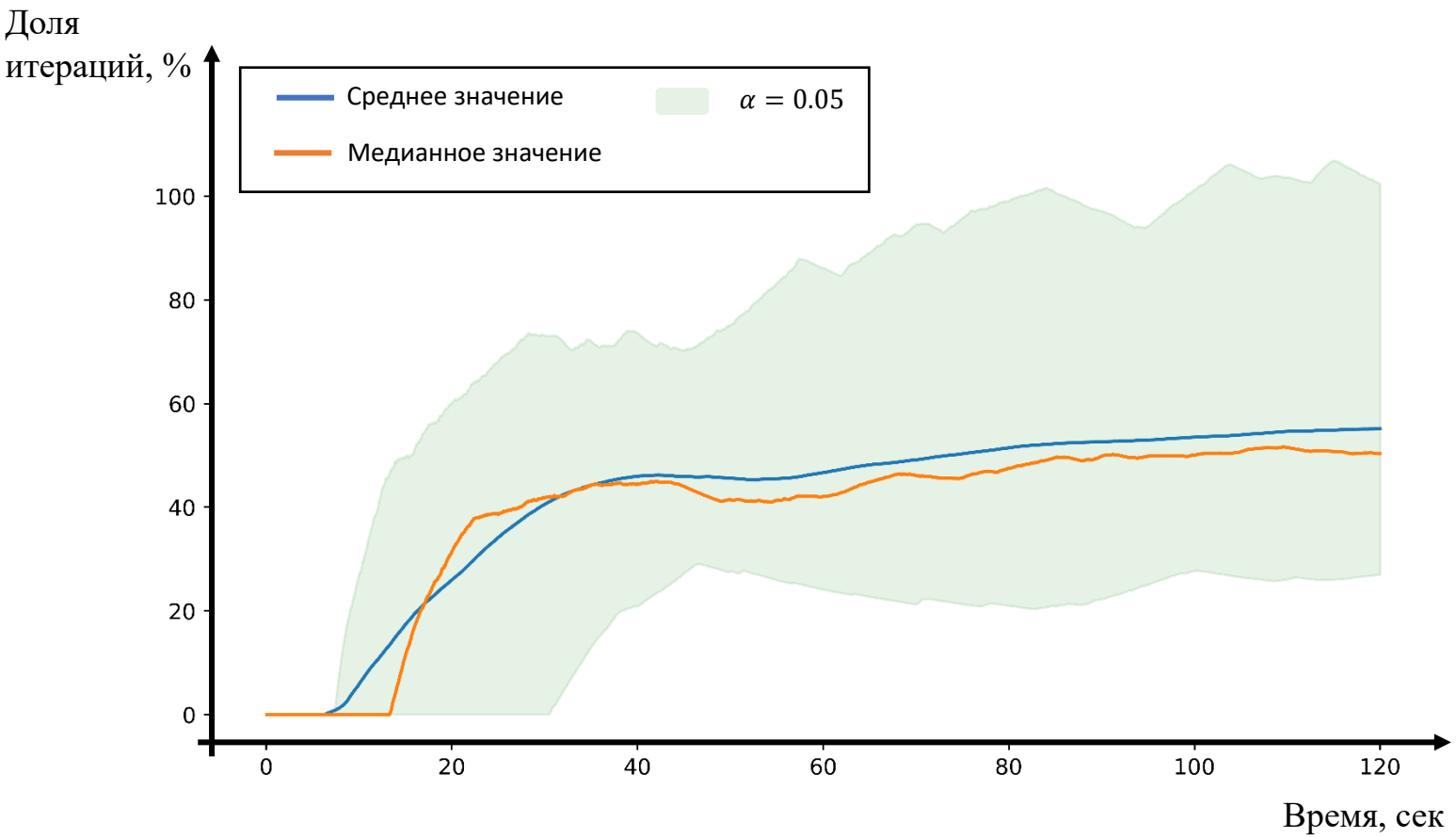
Как видно из графика, в 90% случаев оптимизация не вредит скорости сходимости, причем в среднем и медианном значении наоборот, только помогает улучшить ее. Таким образом, использование локального оптимизатора пути в целом по такой метрике может улучшить сходимость на ~30%, если сравнивать с алгоритмом без оптимизатора.

3) Относительное число итераций.

График строится совершенно аналогично пункту 2, только в его основе не стоимость траектории с течением времени, а количество итераций с течением времени.

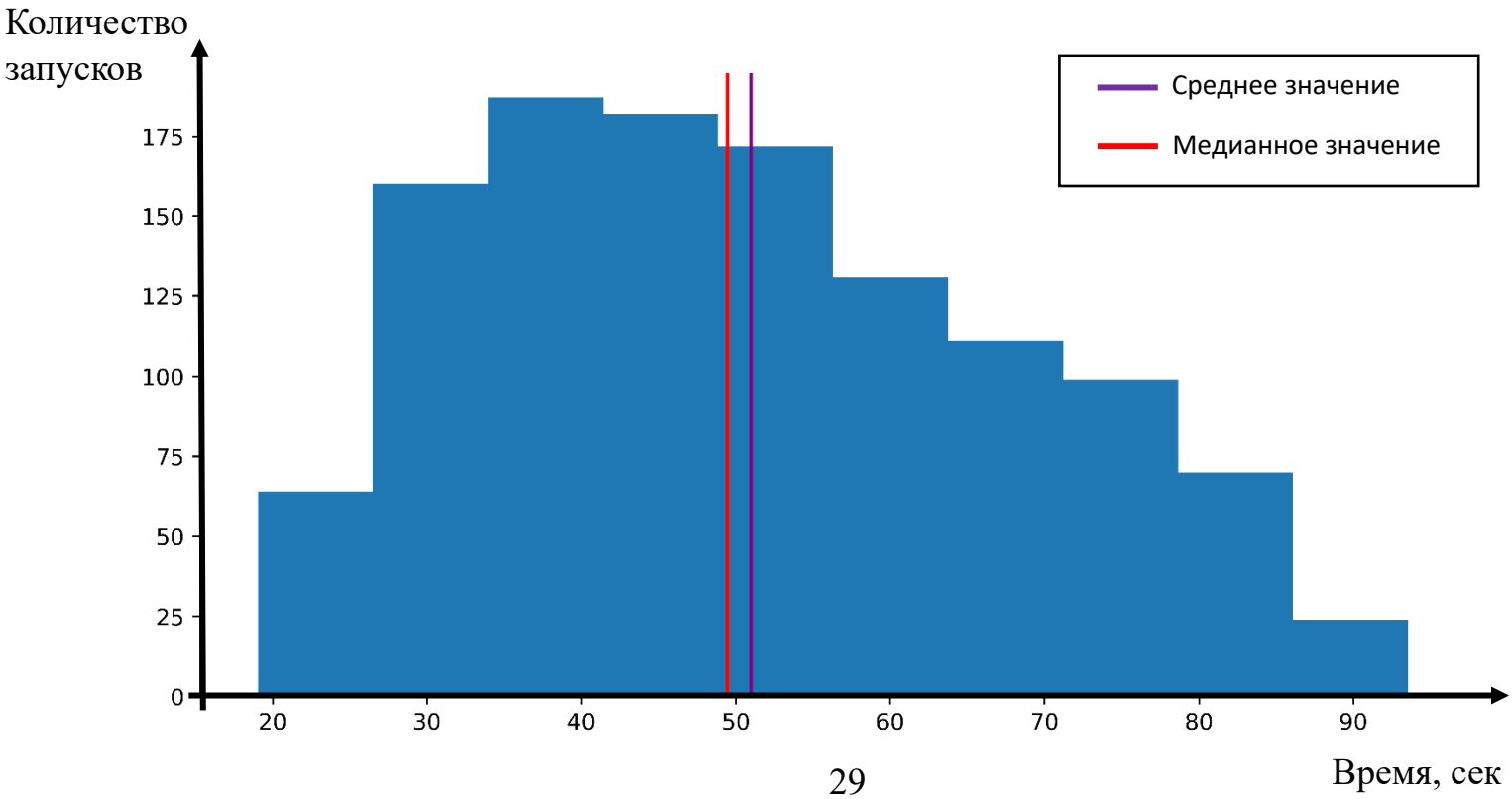
Читать данный график следует так: “в момент времени x количество итераций алгоритма Opt-Informed-RRT*составляет y % от числа итераций, совершенных Informed-RRT*”

График выглядит следующим образом:



Как можем наблюдать, алгоритм Opt-Informed-RRT* совершает примерно в 2 раза меньше итераций, чем Informed-RRT*. Это может быть полезно, если итерации и проверки на допустимость конфигурации будут дорогостоящими, но в работе данный аспект не изучался.

4) Относительное время оптимизации.

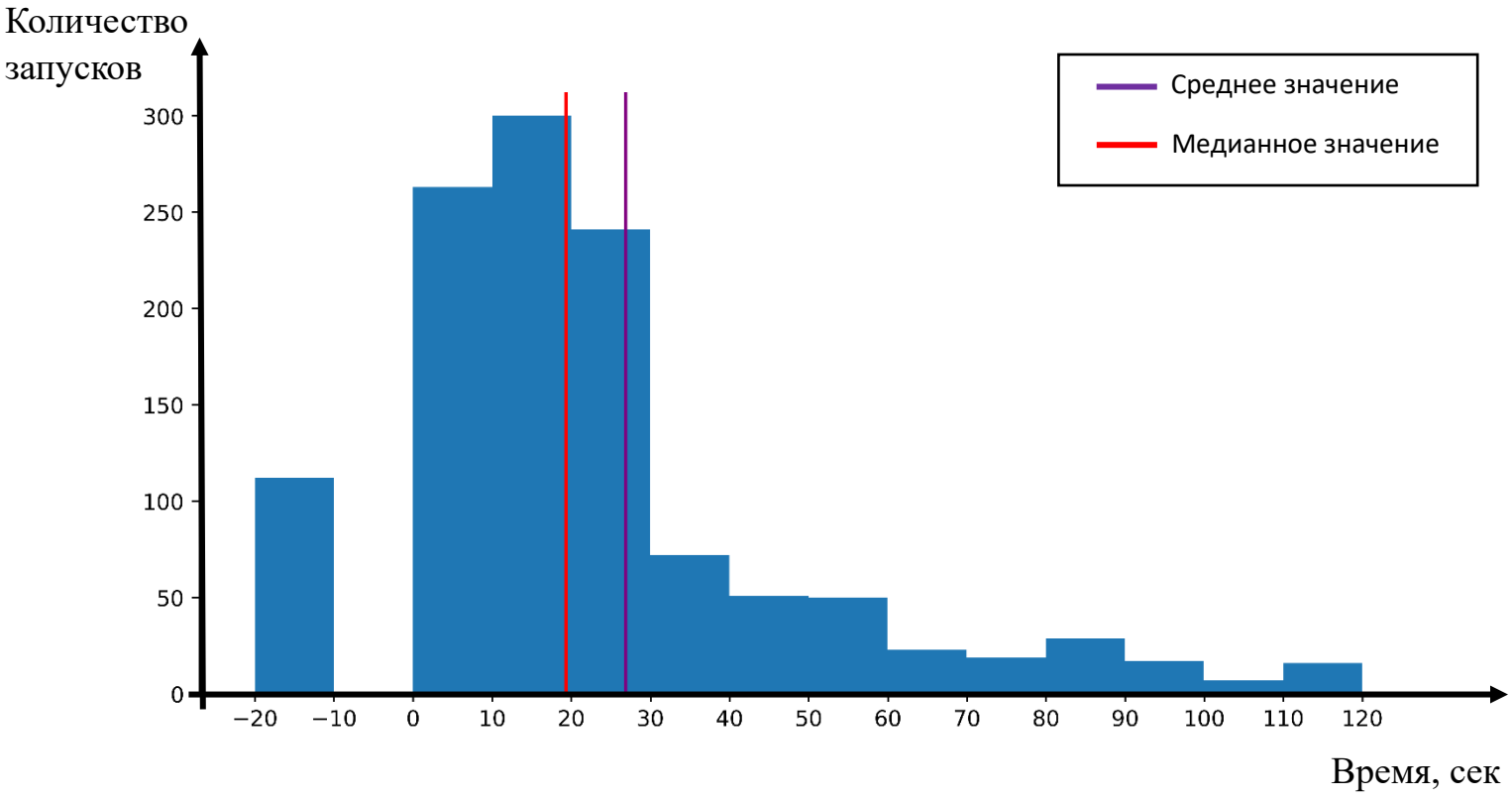


Для каждого из 1200 запусков алгоритма Opt-Informed-RRT* считается, сколько времени в секундах занимает работа оптимизатора.

По полученной гистограмме можно определить среднее значение с отклонением, а также медианное значение времени работы оптимизатора: 51 ± 17.4 и 49.4 секунд соответственно.

5) Время для достижения лучшего результата.

Этот график получается следующим образом: для каждого из 60 усредненных графиков алгоритма Informed-RRT* из пункта 1 смотрится итоговое значение стоимости траектории спустя 120 секунд работы. Потом для каждого из 1200 запусков алгоритма Opt-Informed-RRT* значение получается ответом на вопрос: “за сколько секунд Opt-Informed-RRT* будет иметь меньшую стоимость траектории, нежели средний Informed-RRT* за 120 секунд?”. Если же ему не удалось найти траекторию лучше за 120 секунд, то значение выставляется -20 секунд. Полученная гистограмма выглядит так:



Заметим, однако, что значений -20 получилось $112 / 1200 = 9.3\,\%$. Если не учитывать их и исходить только из $90.7\,\%$, где оптимизатор действительно что-то улучшил, то среднее значение с отклонением и медианное значения будут равны 26.8 ± 24 и 19.3 секунд соответственно.

Таким образом мы получаем несколько разносторонних оценок эффективности работы алгоритма, по которым можно судить, что Opt-Informed-RRT* справляется с задачей оптимизации пути лучше, чем Informed-RRT*.

5. Заключение.

5.1. Результаты работы.

В результате работы был проведен обзор литературы, поставлена формальная постановка задачи, реализован и выложен в открытые источники код алгоритма Opt-Informed-RRT*, который доступен по ссылке https://github.com/Komment314/opt-informed-rrt-star. Для него было проведено экспериментальное сравнение с алгоритмом Informed-RRT*, которое показало, что представленный в работе алгоритм после нахождения траектории эффективнее улучшает сходимость длины допустимой траектории к оптимальной.

5.2. Дальнейшие направления исследования.

Хочется отметить, что в данной работе есть несколько незатронутых направлений, в которых можно проводить дальнейшие исследования.

1) Добавление весов.

В работе функция $dist$ рассматривалась как евклидово расстояние в конфигурационном пространстве. Это означало, в частности, что все роторы равноправны. Но в реальности повернуть самые первые роторы обычно дороже в плане энергии, нежели повернуть роторы ближе к концу манипулятора, т.е. хотелось бы вращать самые большие звенья меньше, чем самые маленькие. Для этого может быть использовано взвешенное расстояние, т.е. каждому ротору сопоставится его “эмпирическая важность”, и расстояния уже будут измеряться взвешенно. Это в теории бы позволило искать более реалистичные траектории.

2) Оптимизация по нескольким целям

Также, после нахождения первой допустимой траектории от старта до цели расширение дерева в сторону других целевых конфигураций прекращалось. Это было связано с тем, что в случае большого числа целевых конфигураций, техника генерации образцов в эллипсе никак не уменьшает исследуемое пространство, а

значит для заметного уменьшения стоимости траектории потребовалось бы большее число итераций, соответственно, больше времени потребовалось бы на бенчмаркинг. Но в теории изучение многоцелевого планирования возможно, если эффективно реализовать равномерную генерацию образцов уже в объединении эллипсоидных областей и иметь больший запас времени на бенчмаркинг.

3) Выпуклая оптимизация.

Поскольку итеративные алгоритмы выпуклой оптимизации в работе не использовались, то стоит также рассмотреть их применение в алгоритме.

4) Бесконечные провороты.

Помимо этого, возможно иное поведение оптимизации для манипулятора, у которого некоторые роторы могут проворачиваться бесконечное число раз, иными словами, некоторые оси конфигурационного пространства будут периодическими. Этот факт влечет за собой отсутствие выпуклости и гладкости у функции расстояния в таком пространстве. Вследствие этого, например, невозможно использование выпуклой оптимизации, а якобианы и гессианы в задаче невыпуклой минимизации будут выглядеть по-другому.

Все эти направления рассматриваются как основные для возможного дальнейшего исследования.

Список литературы.

[1] Dijkstra E. W. A note on two problems in connexion with graphs. (1959)

[2] Hart P. E., Nilsson N. J., Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. (1968)

[3] Bellman, Richard Ernest; Rand Corporation. Dynamic programming. (1957)

[4] M. Andrychowicz, F. Wolski, AlexRay, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, OpenAI. Hindsight Experience Replay. (2018)

[5] M. Zucker, J. Kuffner, J. A. Bagnell. Adaptive workspace biasing for sampling-based planners. (2008)

[6] Kavraki L. E., Svestka P., Latombe J.C., Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. (1996)

[7] LaValle, Steven M. Rapidly-exploring random trees: A new tool for path planning. (October 1998)

[8] J.J. Kuffner, S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. (2000)

[9] J. D. Gammell, S. S. Srinivasa, T. D. Barfoot. Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic. (2014)

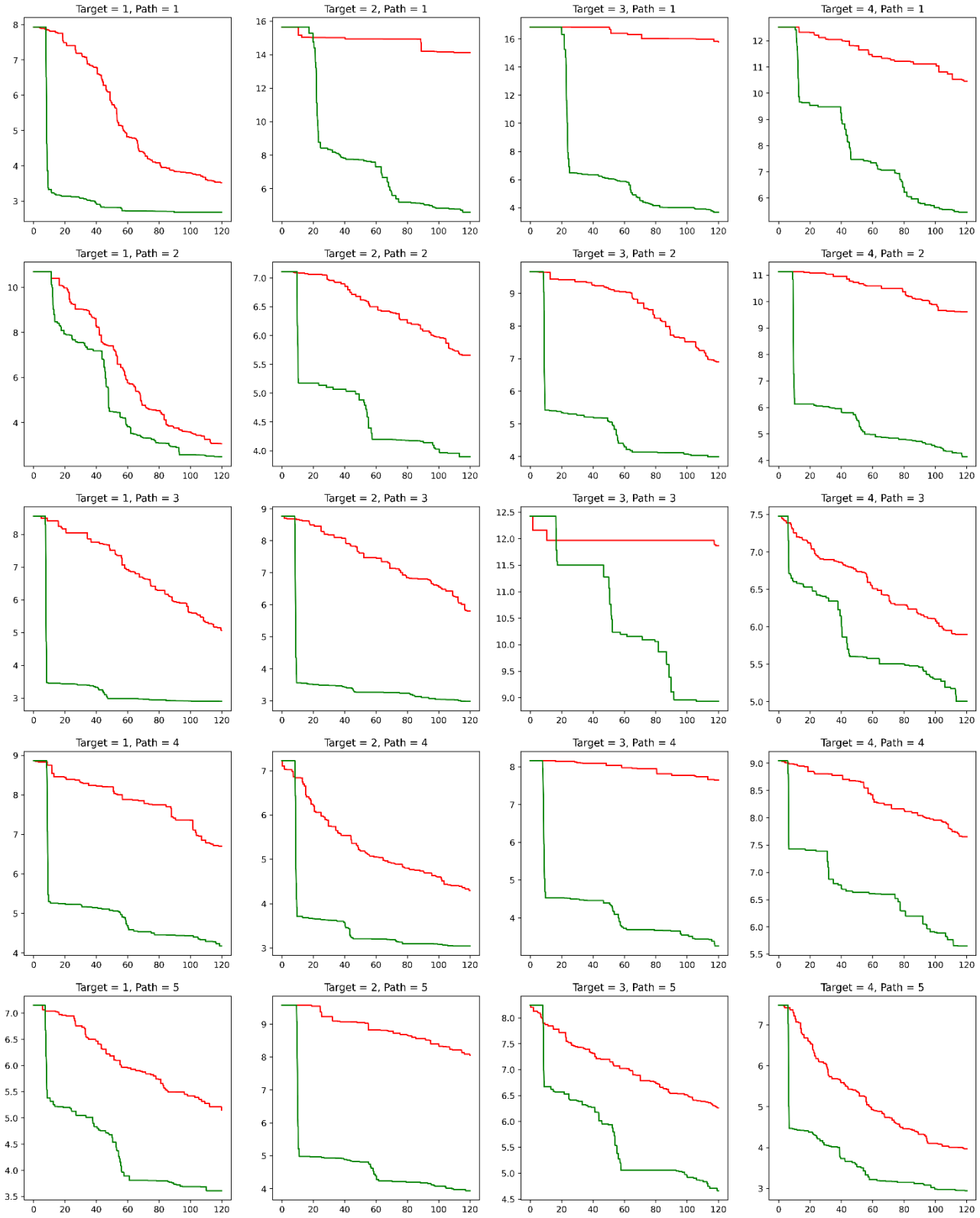
[10] J. D. Gammell, S. S. Srinivasa, T. D. Barfoot. Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. (2015)

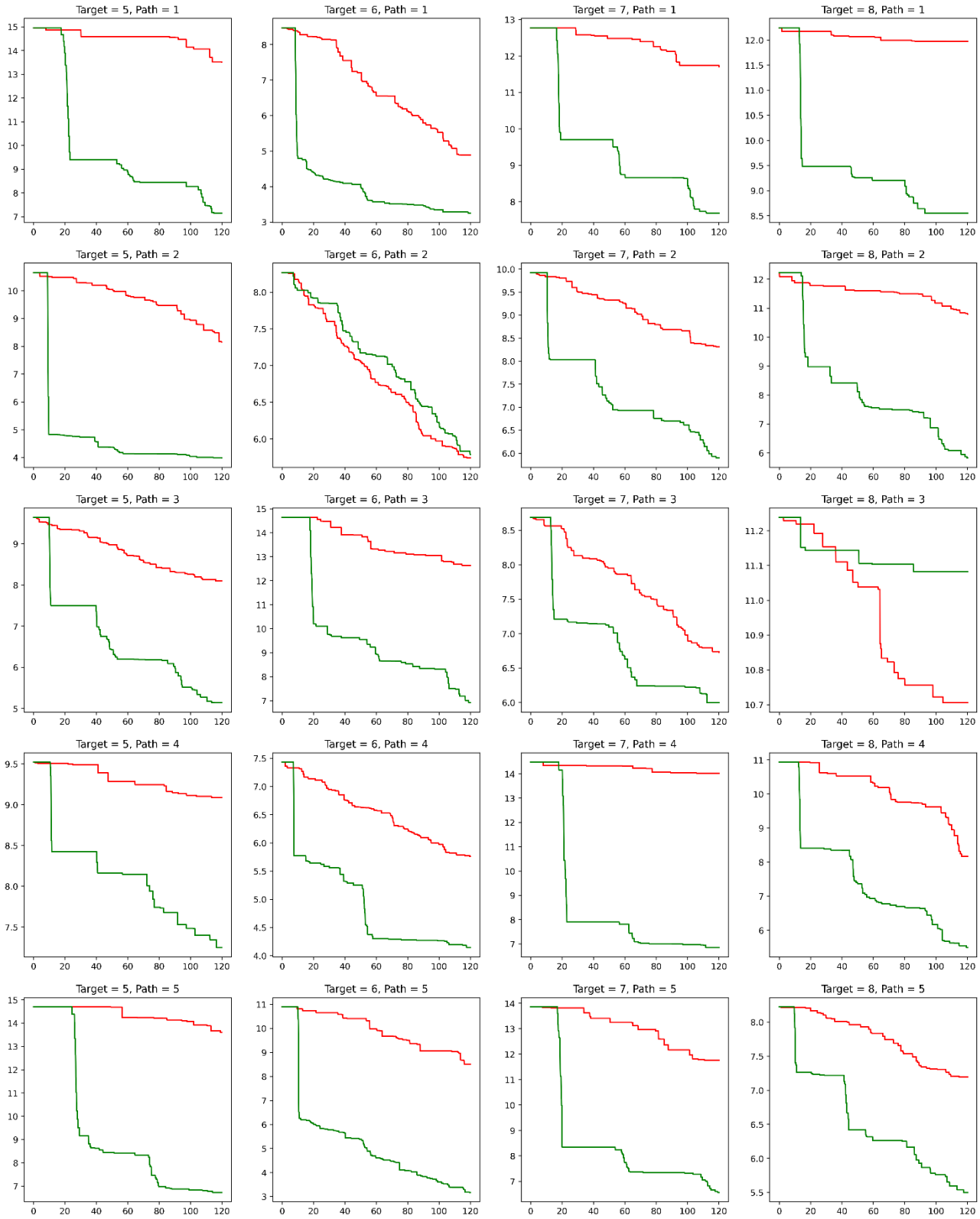
[11] S. Karaman, E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. (2010)

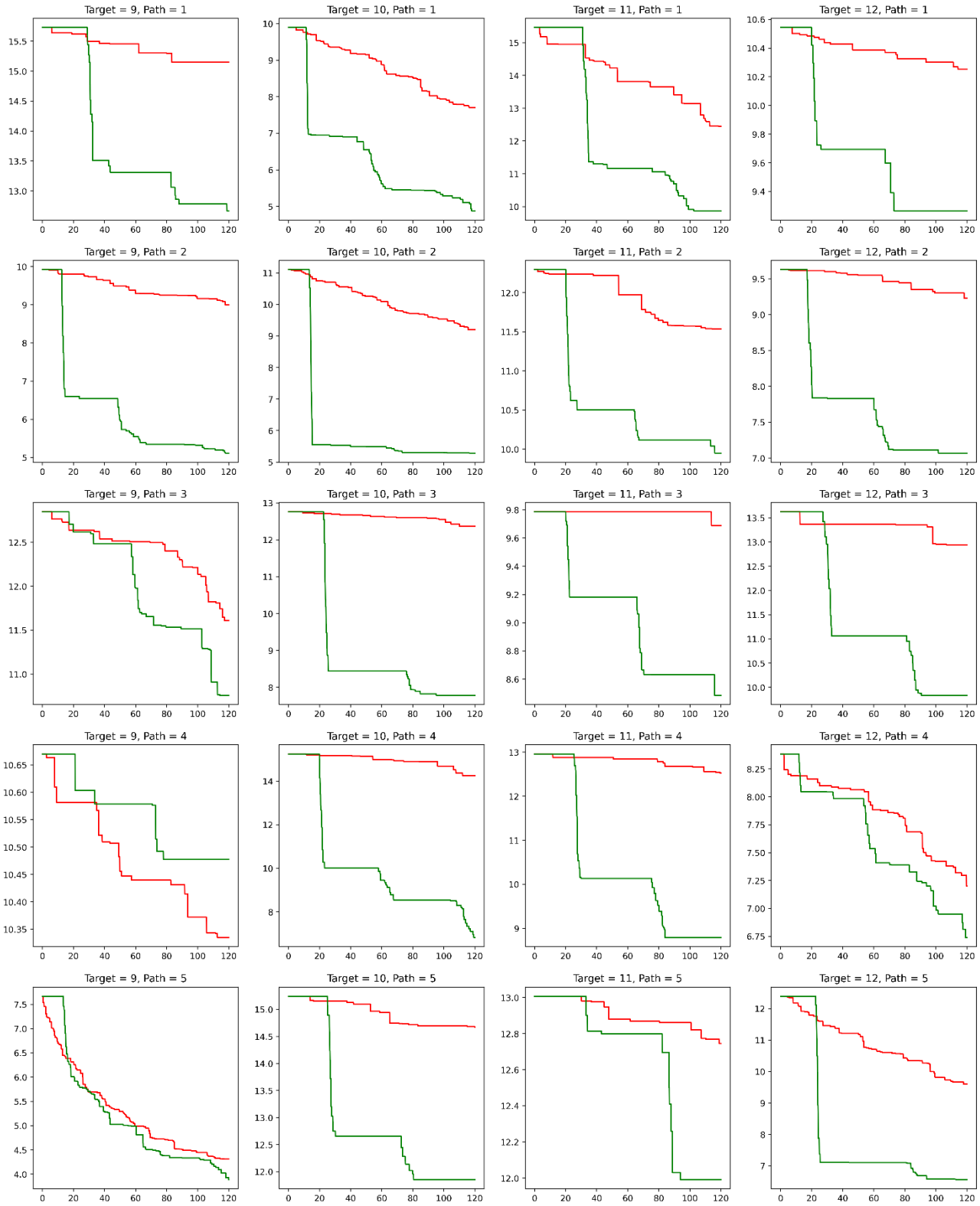
[12] D. Kraft. A software package for sequential quadratic programming. (1988)

Приложение.

Графики для каждой из 60 начальных траекторий. Ось X — это время в секундах, ось Y— это стоимость траектории на данный момент. Красным отмечен алгоритм Informed-RRT*, зеленым – Opt-Informed-RRT*.




Target = 5, Path = 1
Target = 6, Path = 1
Target = 7, Path = 1
Target = 8, Path = 1
Target = 5, Path = 2
Target = 6, Path = 2
Target = 7, Path = 2
Target = 8, Path = 2
Target = 5, Path = 3
Target = 6, Path = 3
Target = 7, Path = 3
Target = 8, Path = 3
Target = 5, Path = 4
Target = 6, Path = 4
Target = 7, Path = 4
Target = 8, Path = 4
Target = 5, Path = 5
Target = 6, Path = 5
Target = 7, Path = 5
Target = 8, Path = 5


Target = 9, Path = 1
Target = 10, Path = 1
Target = 11, Path = 1
Target = 12, Path = 1
Target = 9, Path = 2
Target = 10, Path = 2
Target = 11, Path = 2
Target = 12, Path = 2
Target = 9, Path = 3
Target = 10, Path = 3
Target = 11, Path = 3
Target = 12, Path = 3
Target = 9, Path = 4
Target = 10, Path = 4
Target = 11, Path = 4
Target = 12, Path = 4
Target = 9, Path = 5
Target = 10, Path = 5
Target = 11, Path = 5
Target = 12, Path = 5