Объектно-ориентированная среда для разработки приложений планирования движения[1]

[1] *К.А. Казаков <kazakov@ispras.ru>*
[1,2] *В.А. Семенов <sem@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*
[2] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. Обсуждаются принципы организации и функционирования инструментальной среды для программной реализации моделей, методов и приложений теории планирования движения. Среда предоставляет развитый набор готовых к использованию программных компонентов для автоматического построения бесконфликтных траекторий для робота, перемещаемого в статическом и динамическом трехмерном окружении. Организация среды в виде объектно-ориентированного каркаса обеспечивает развитие, адаптацию и гибкое конфигурирование разработанных программных компонентов в составе целевых приложений. Благодаря выделенным интерфейсам разного уровня и предусмотренным точкам расширения среда допускает интеграцию со сторонними прикладными системами.

Ключевые слова: планирование движения; поиск пути; определение столкновений; программная инженерия; объектно-ориентированное программирование.

DOI: 10.15514/ISPRAS-2017-29(5)-11

Для цитирования: Казаков К.А., Семенов В.А. Объектно-ориентированная среда для разработки приложений планирования движения. Труды ИСП РАН, том 29, вып. 5, 2017 г., стр. 185-238. DOI: 10.15514/ISPRAS-2017-29(5)-11

1. Введение

Под планированием движения обычно понимается поиск бесконфликтного пути для перемещения твердого тела или кинематической конструкции в пространственно-трехмерной сцене. Искомый путь строится в конфигурационном пространстве объекта с учетом его степеней свободы и представляет собой непрерывную кривую, которая соединяет его начальное и

[1] Работа поддержана РФФИ (грант 16-07-00606)

конечное положения, исключает столкновения с препятствиями сцены и удовлетворяет всем установленным кинематическим и динамическим ограничениям [1, 2].

Задачи планирования движения возникают в разнообразных предметных областях, таких как машиностроение, робототехника, геоинформатика, транспорт, строительство, и часто связаны с автоматизацией и интеллектуализацией производственных процессов. Повышенный интерес к данным задачам обусловлен также и развитием современных технологий математического моделирования, компьютерной графики, виртуальной и дополненной реальности, допускающих конструктивное комплексное применение в составе целевых прикладных систем.

К подобным системам следует отнести системы автоматизированного проектирования, производства и инженерии (CAD/CAM/CAE), которые получили широкое распространение благодаря возможностям математического моделирования технологически сложных продуктов и процессов. Функционал данных систем охватывает многие математические и инженерные дисциплины, среди которых важное место занимают геометрическое моделирование и планирование движения.

Например, системы визуального моделирования индустриальных проектов, такие как Synchro Professional, Autodesk Navisworks, Trimble Vico, Rib iTWO, Bentley ConstructSim, реализуют функции навигации в трехмерном пространстве и времени, а также определения коллизий в динамических сценах. Благодаря данным функциям удается промоделировать ход проектных работ, выявить проблемы их координации в условиях ограниченных рабочих пространств и жестких временных сроков и, тем самым, снизить проектные риски и затраты. Не менее важными являются функции моделирования способов доставки материалов и оборудования к месту использования посредством различных грузоподъемных механизмов, а также моделирование процессов монтажа и сборки проектных конструкций. Реализация данных функций связана с решением задач планирования движения в сложном динамическом окружении, состоящем из сотен тысяч объектов с собственными геометрическими моделями и поведением [3, 4].

Другими интересными приложениями теории планирования движения являются задачи автоматической сборки или разборки машиностроительных изделий с целью верификации технологических процессов и предоставления интерактивных инструкций по производству и эксплуатации. Не менее актуальными являются задачи управления манипуляционными роботами, в которых требуется построить согласованные траектории звеньев манипуляторов.

Востребованными в последние годы являются средства навигации мобильных роботов, в частности, колесных транспортных механизмов с небольшим числом степеней свободы и неголономными связями. Колесные механизмы удовлетворяют специальным геометрическим ограничениям, которые приводят

к траекториям движения вдоль кривых Дьюбинса [5], Ридса-Шеппа [6], Балккома-Мейсона [7]. В некоторых постановках требуется прокладывать относительно протяженные траектории в псевдотрехмерном окружении (2.5D), характерном, например, для транспортных и архитектурно-строительных моделей. Информация о моделируемом окружении может быть заранее известна, либо поступать с сенсоров робота в режиме реального времени. В последнем случае возникает необходимость перманентного перепланирования движения объекта в ходе его перемещения [8, 9].

Содержательные примеры планирования движения предоставляют системы реалистичной компьютерной анимации неголономных систем со сложными кинематическими ограничениями. С помощью подобных систем успешно решаются задачи анимации человеческих персонажей, востребованные, например, в компьютерных играх и киноиндустрии, в том числе, с применением технологий виртуальной и дополненной реальности. Воссоздание реалистичной модели движения, учитывающей геометрические, кинематические и дифференциальные ограничения, представляется невозможным без использования соответствующих программных средств.

Перечисленные выше задачи планирования движения являются PSPACE-трудными, и поэтому их решение в индустриально значимых постановках высокой размерности представляет серьезную вычислительную проблему. Разработка необходимого математического и программного обеспечения с самого начала крайне сложна, требует широких компетенций и серьезных ресурсов. Использование же существующих программных средств, как коммерческих, так и с открытым исходным кодом сопряжено с рядом принципиальных ограничений и проблем. К числу наиболее значимых недостатков следует отнести:

— специализацию средств и невозможность их использования для решения задач общего вида, например, задач глобального планирования движения в сложном динамическом окружении;

— ограниченные инструментальные возможности для развития программных средств и реализации новых моделей, методов и приложений теории планирования движения;

— отсутствие виртуализации данных, приводящее к избыточному представлению модели окружения и быстрому исчерпанию оперативной памяти, необходимой для основных вычислений;

— наличие зависимостей от компонентов третьих сторон, часто приводящих к неопределенности и недостоверности результатов, а также порождающих дополнительные трудности при конфигурировании целевых приложений.

Поясним указанные недостатки на примере популярных библиотек планирования движения Motion Planning Kit (MPK) [10], OpenRave [11] и Open Motion Planning Library (OMPL) [12].

Функции данных библиотек, главным образом, предназначены для решения задач локального планирования движения и моделирования неголономных механических систем в режиме реального времени. Их математический арсенал в основном базируется на сэмплинг методах, которые демонстрируют высокую эффективность в приложениях управления промышленными роботами. Однако данные методы несостоятельны в случаях, когда требуется определить протяженные бесконфликтные траектории в трехмерном окружении со сложной топологией и динамическим поведением. Подобные задачи возникают, в частности, при визуальном моделировании архитектурно-строительных проектов и требуют эффективные средства для решения задач планирования движения в глобальных динамических постановках. При этом архитектура библиотек и особенности организации прикладных программных интерфейсов (API) препятствуют реализации в их составе новых методов и алгоритмов планирования движения для решения иных классов задач. Предусмотренные в библиотеках возможности модификации локальных алгоритмов качественно не меняют ситуацию.

Другим недостатком библиотек являются способы организации программных интерфейсов доступа к данным окружения, которые зачастую препятствуют их интеграции в целевые CAD/CAM/CAE системы. Данные системы обычно оперируют масштабными сценами, состоящими из сотен тысяч и миллионов объектов с индивидуальными геометрическими и динамическими характеристиками, и поэтому организация эффективного доступа к ним приобретает ключевое значение. В подобных системах используется свое внутреннее представление трехмерных моделей, которое диктуется их функционалом и прикладной спецификой. Например, трехмерные CAD системы поддерживают работу с аналитическими геометрическими кривыми и поверхностями, граничным и твердотельным представлениями тел. Системы визуального моделирования проектов ориентированы на работу с упрощенными полигональными моделями, предназначенными для быстрой растеризации сцен и локализации пространственных коллизий. CAM и PLM системы в большей степени оперируют кинематическими моделями, предназначенными для моделирования технологических операций.

В силу указанных причин интерфейс доступа к данным окружения должен учитывать альтернативные представления трехмерных моделей, а также допускать возможность непосредственного использования функций целевых систем при определении столкновений и анализе согласованности объектных конфигураций. Данные требования приводят к необходимости иметь точки расширения программной среды (hotspots), которые бы позволили настроить, сконфигурировать или доработать соответствующие компоненты для поддержки альтернативных моделей и реализации операций с ними.

В самом деле, жесткая привязка интерфейса к конкретной модели целевой системы, во-первых, требует написания большого объема адаптирующего кода, а, во-вторых, приводит к нерациональному расходованию оперативной памяти.

С другой стороны, полная независимость от геометрической и кинематической моделей также порождает трудности при интеграции в целевую систему. Например, при использовании библиотеки OMPL, реализующей сэмплинг методы, на разработчика ложится ответственность за предоставление релевантных метрик многомерных конфигурационных пространств, в которых решаются пользовательские задачи. Кроме того, разработчиком должны быть реализованы и предоставлены эффективные средства определения столкновений и анализа согласованности конфигураций, что также является серьезной проблемой с учетом высокой размерности типовых задач.

Наконец, зависимость от компонентов третьих сторон и их сильное зацепление по данным и управлению также является недостатком существующих библиотек планирования движения. С одной стороны, использование детально специфицированных и тщательно отлаженных сторонних компонентов является хорошей практикой при создании сложных программных систем. Однако она оправдывает себя, если только компоненты реализуют независимые функции, оперирующие с единым представлением данных. В противном случае возникает необходимость их перманентной конвертации и согласования. Кроме того, использование компонентов разных версий часто порождает конфликты, которые затрудняют сборку, тестирование и дальнейшее сопровождение целевых систем.

Например, инструментальная среда MoveIt, входящая в состав ROS, построена на основе сразу всех упомянутых библиотек планирования движения. Причем OMPL предлагается в качестве основного средства построения бесконфликтных траекторий, OpenRave используется как средство решения обратной кинематической задачи, а MPK применяется для быстрого разрешения одиночных запросов на основе двухнаправленных маршрутных сетей с отложенным определением столкновений на основе SBL алгоритма. Очевидно, что систематическое построение приложений планирования движения с помощью данной инструментальной среды проблематично, а полученные результаты могут быть недостоверными.

Множественные зависимости среды от ядра ROS, библиотеки определения столкновений FCL, а также FLANN, MongoDB и boost создают дополнительные трудности при ее практическом использовании.

2. Назначение и общая организация среды

Разработанная инструментальная среда предназначена для программной реализации моделей, методов и приложений теории планирования движения. Среда предоставляет развитый набор готовых к использованию программных компонентов для автоматического построения бесконфликтных траекторий в статическом и динамическом трехмерном окружении для робота, имеющего произвольное число степеней свободы и функционирующего как на основе априорных знаний о сцене, так и в неизвестном окружении в режиме реального времени. Среда реализуется не только как специализированная библиотека для

решения типовых задач планирования движения, но и как библиотека программных компонентов для построения приложений в разных предметных областях. Именно данная возможность принципиально отличает ее от упомянутых выше решений. При этом декларируемая универсальность среды не препятствует эффективности разрабатываемых целевых приложений, поскольку предусматривает возможности гибкого конфигурирования программных компонентов, их развития, адаптации и настройки с учетом специфики решаемых прикладных задач.

Среда спроектирована и реализована в виде каркаса (архитектурного шаблона) на основе технологий объектно-ориентированного и компонентно-ориентированного программирования. Данные технологии широко применяются при создании сложных программных систем с развиваемым функционалом и при разработке серий программных приложений. Ожидается, что благодаря каркасной организации среда позволит существенно сократить сроки и затраты на создание приложений и при этом обеспечит их надежность и эффективность, необходимые для решения практических вычислительно сложных задач. Среда программно реализована на языке Си++, поэтому в дальнейшем ее компоненты описываются в терминах абстрактных и конкретных классов.

Проектированию каркаса предшествовали исследовательские работы, связанные с систематизацией и концептуализацией задач и методов теории планирования движения. В ранее опубликованных авторами работах [2, 13] был проведен анализ современных методов планирования движения, который позволил выделить основные постановки задач, ключевые подходы к их решению, перспективные семейства методов и эффективные алгоритмы. В дальнейшем мы выделяем две основные постановки задач и связанные с ними вычислительные стратегии, а именно: локальное и глобальное планирование движения.

Глобальное планирование предполагает априорное знание о моделируемом окружении и подразумевает предварительный анализ трехмерной сцены с последующим разрешением множественных запросов поиска пути. Данная постановка реализуется посредством предварительного построения маршрутного графа, отражающего топологию моделируемой сцены. Маршрутный топологический граф агрегирует множество локальных маршрутов в рабочем пространстве сцены, оценки их стоимости, а также дополнительную информацию, которая может быть полезна для разрешения последующих запросов на основе эвристических правил. Глобальное планирование на основе маршрутных сетей предполагает быстрый поиск перспективных маршрутов движения, которые затем могут быть проверифицированы и при необходимости скорректированы с помощью алгоритмов локального планирования.

Локальное планирование подразумевает разрешение одиночных запросов поиска бесконфликтных траекторий для конкретного объекта (твердого тела

или кинематической системы), учитывая его конкретное геометрическое представление, а также наложенные кинематические и динамические ограничения. Для программной реализации средств локального планирования было выбрано семейство сэмплинг методов, зарекомендовавших себя при решении задач в сложных многомерных конфигурационных пространствах.

Для решения широких классов задач в локальной и глобальной постановке с использованием альтернативных методов и алгоритмов был проведен объектный анализ теории планирования движения. Он позволил выделить ее ключевые абстракции, их взаимосвязи, и тем самым, определить необходимый состав, структуру и возможные точки расширения среды. В ее общей структуре удобно выделить группы классов для представления моделируемого трехмерного окружения (пакет WorkSpace), для решения вспомогательных задач в конфигурационных пространствах объектов (пакет ConfigurationSpace) и для работы с графами и маршрутными сетями (пакет DiscreteSpace).

Классы пакета WorkSpace определяют интерфейсы доступа к объектам моделируемого окружения, которое включает в себя и перемещаемый объект и наложенные на него кинетические ограничения. В этом же пакете реализуется подсистема классов для определения столкновений с использованием структур пространственно-временной индексации. Последние применяются для быстрой локализации столкновений и ускорения работы основных алгоритмов. Поскольку некоторые CAD/CAM/CAE системы имеют собственные средства определения столкновений, оперирующие непосредственно с внутренним представлением данных окружения, классы пакета определяют единые интерфейсы для формирования и исполнения соответствующих запросов. Благодаря интерфейсам разработчики целевых приложений могут предоставить альтернативные реализации средств определения столкновений.

Пакет ConfigurationSpace составляют классы, предназначенные для решения задач в конфигурационных пространствах объектов. Данные классы позволяют задать множество допустимых конфигураций для объекта планирования, сформировать и исполнить запросы поиска бесконфликтных траекторий движения. Специальные классы решателей реализуют различные семейства сэмплинг методов для планирования движения в локальной постановке.

Пакет классов DiscreteSpace предназначен для представления графов и решения задач поиска путей в них. Обобщенные реализации классов позволяют использовать их в качестве базовых при специализации графов в виде быстрорастущих деревьев и маршрутных сетей. Заметим, что последние могут строиться как в трехмерном пространстве окружения, так и в конфигурационном пространстве объекта планирования. Тем самым, обеспечивается возможность многоцелевого использования графовых структур данных и алгоритмов поиска без необходимости дублирования кода.

Пакеты используют единый набор вспомогательных типов данных, предназначенных в основном для работы с примитивами компьютерной графики и реализации векторно-матричных операций.

Наконец, среда включает в себя подсистему классов для глобального планирования движения. Подсистема реализует общую вычислительную стратегию, заключающуюся в последовательной редукции исходной вычислительно сложной задачи планирования движения к типовым, относительно простым задачам поиска в графах. Наши предыдущие исследования показали эффективность и перспективность подобной стратегии [3, 13, 14]. Примечательно, что упомянутые выше сэмплинг методы, маршрутные сети и алгоритмы теории графов являются элементами данной общей стратегии, а ее программная реализация базируется на согласованном использовании рабочего, конфигурационного и дискретного представлений, поддерживаемых соответствующими классами среды. При изменениях в моделируемом окружении, данные представления инкрементально обновляются с возможностью оперативного разрешения последующих запросов маршрутизации. Подсистема глобального планирования встраивается в цикл работы целевой системы и, работая в фоновом режиме, формирует и обрабатывает очередь запросов, связанных с изменениями в моделируемом окружении.

Классы среды тесно связаны друг с другом и составляют единый инструментарий для программной реализации приложений теории планирования движения. Предусмотренные точки расширения в виде абстрактных классов позволяют адаптировать существующие компоненты и реализовать новые с учетом специфики прикладных задач и возможностей применения эффективных методов и алгоритмов.

3. Пакет классов WorkSpace

Рассмотрим более подробно организацию пакета классов WorkSpace. Моделируемое трехмерное окружение представляет собой множество разнородных геометрических объектов. С каждым объектом связан уникальный идентификатор, геометрическая модель и локальная система координат, определяющая его положение в окружении. Предполагается, что с каждым объектом может быть ассоциирована альтернативная геометрическая модель, применяемая, например, для быстрой локализации столкновений или растеризации сцены. В дальнейшем рассматривается два вида объектов: простые твердые тела и кинематические системы взаимосвязанных твердотельных звеньев.

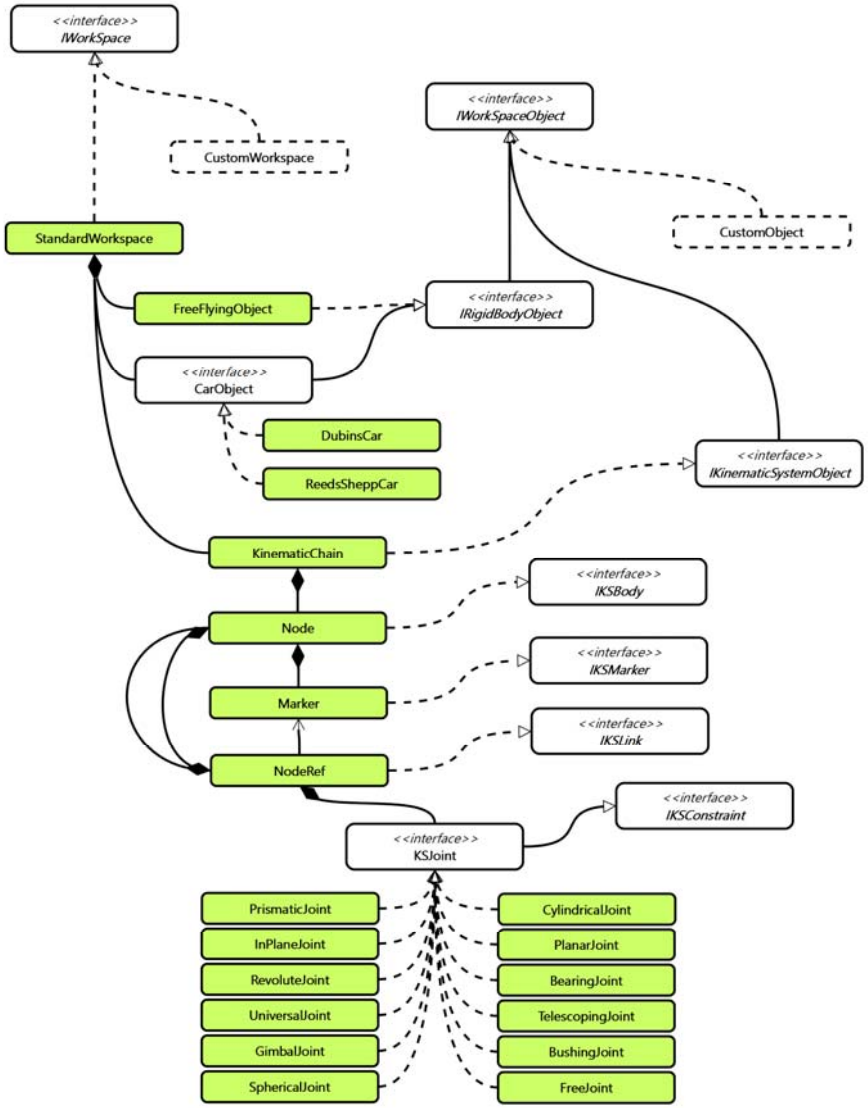


Рис. 1. Диаграмма классов трехмерного окружения.
Fig. 1. 3D WorkSpace class diagram.

Заметим, что в CAD/CAM/CAE приложениях часто используются, так называемые, сложные или составные объекты, которые представляются композицией дочерних объектов и организованы в самостоятельную иерархию. Сложные объекты могут, например, определять слои, соответствующие поэтажному плану здания или составу его конструктивных элементов. В машиностроительных отраслях сложные объекты часто используются для определения сборок деталей. Для обсуждаемых задач планирования движения композиционная структура объектов не важна и далее во внимание не принимается. Последнее не исключает, что объекты окружения могут иметь сложное геометрическое представление.

Поскольку среда предназначена для разработки приложений планирования движения, в которых может использоваться свое собственное представление данных окружения, классы пакета WorkSpace определяют лишь общий интерфейс доступа к ним. Ответственность за реализацию интерфейса целиком ложится на разработчика приложения. Интерфейс позволяет избежать привязки к прикладным типам данных, при этом предоставляя необходимые общие методы обхода объектов, получения их индивидуальных геометрических и поведенческих моделей, а также текущего состояния. Обсудим вопросы организации интерфейса более подробно.

3.1 Доступ к объектам окружения

Абстрактные классы ***IWorkSpace*** и ***IWorkSpaceObject*** определяют интерфейс доступа к моделируемому окружению.

Интерфейс ***IWorkSpace*** включает в себя следующие виртуальные методы:

- ***getModellingBounds()-> aabb_t***
  возвращает границы моделируемого окружения в виде AABB (Axis Aligned Bounding Box) параллелепипеда. Возвращаемый параллелепипед может охватывать как всю сцену, так и любую ее подобласть, внутри которой необходимо построить маршрутную сеть.
- ***createObjectIterator()-> abstract_iterator<IWorkSpaceObject>***
  создает итератор для однонаправленного обхода разнородных геометрических объектов с общим интерфейсом ***IWorkSpaceObject***.
- ***getObject(uid_t*** *objectID* ***)-> IWorkSpaceObject***
  предоставляет доступ к объекту окружения по заданному идентификатору.
- ***getCurrentState(uid_t*** *objectID* ***)-> IState***
  предоставляет доступ к текущему состоянию объекта по заданному идентификатору.
- ***resolveState(uid_t*** *objectID*, ***IState*** *objectState*, ***resolve_state_callback_t*** *resolveStateCallback* ***)-> bool***

объект с заданным идентификатором устанавливает в положение, соответствующее приписанному состоянию в конфигурационном пространстве. В качестве последнего параметра метода выступает заданная функция применения рассчитанных трансформаций к индивидуальным объектам ***resolve_state_callback_t(uid_t*** *bodyID****, mat4_t*** *bodyTransform* ***)***. В случае задания в качестве объекта кинематической системы метод реализует решение прямой кинематической задачи.

Абстрактный класс ***IWorkSpaceObject*** определяет следующий интерфейс доступа к индивидуальному объекту окружения:

- ***getID()->uid_t***
  получить уникальный идентификатор объекта
- ***createCollisionShape()->ICollisionShape***
  создать альтернативную геометрическую модель объекта для определения столкновений
- ***createStateSpace(aabb_t*** *modellingBounds* ***)->IStateSpace***
  сформировать множество допустимых конфигураций объекта

Интерфейс предусматривает две ключевые функции, а именно: конструирование альтернативной геометрической модели объекта, предназначенной, прежде всего, для эффективного определения столкновений, а также формирование множества допустимых конфигураций объекта для анализа его согласованных состояний и бесконфликтных переходов между ними.

Специальные классы ***IRigidBodyObject*** и ***IKinematicSystemObject*** расширяют базовый интерфейс ***IWorkSpaceObject***, определяя операции для твердотельных объектов и кинематических систем. В ряде случаев необходимо различать подтипы сконструируемых объектов на базовом уровне ***IWorkSpaceObject***, поэтому в интерфейсе предусмотрены соответствующие методы для получения подтипов объектов и приведения к ним объектных ссылок.

Абстрактный класс ***IKinematicSystemObject*** служит для доступа к внутренней структуре кинематической системы, предоставляя методы получения отдельных твердотельных звеньев и кинематических сочленений:

- ***getNumberOfBodies()-> int***
  получить число звеньев
- ***getBody(kuid_t*** *bodyID* ***)-> IKSBody***
  найти звено по идентификатору
- ***createBodyIterator()-> abstract_iterator<IKSBody>***
  создать итератор для обхода звеньев
- ***getNumberOfLinks()-> int***
  получить число сочленений
- ***getLink(kuid_t*** *linkID* ***)-> IKSLink***

  найти сочленение по идентификатору
- ***createLinkIterator()-> abstract_iterator<IKSLink>***
  создать итератор для обхода сочленений

Приведенные методы позволяют выполнить обход звеньев и сочленений кинематической системы и получить необходимый доступ к их параметрам. Такая организация класса преследует сразу несколько целей. Во-первых, могу быть заданы целевые положения звеньев системы, необходимые для формирования запросов планирования движения. Во-вторых, упрощается и унифицируется процедура формирования множеств допустимых конфигураций для сложных кинематических систем. В-третьих, становится возможным идентифицировать конфигурации, приводящие к самопересечениям кинематических систем. В-четвертых, благодаря выделенным абстракциям звена и сочленения обеспечивается возможность развития среды в направлении поддержки средств физического моделирования.

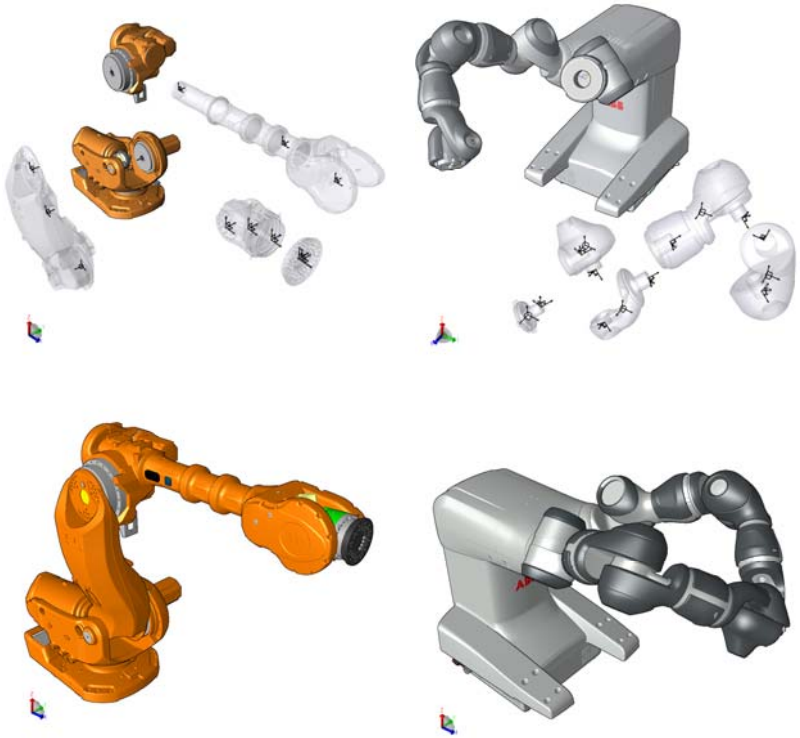


Рис. 2. Примеры моделей манипуляционных роботов: IRB7600 и IRB14000
Fig. 2. Example manipulation robot models: IRB7600 и IRB14000.

Звено кинематической системы предоставлено в среде абстрактным классом ***IKSBody***. Он позволяет получить основные параметры звена, такие как локальная система координат, масса, центр масс и матрица тензора инерции. Кроме того, он предоставляет возможность обхода точек сочленения звена, представленных абстрактным классом ***IKSMarker***. Точка сочленения (или маркер) является уникально идентифицируемым объектом, атрибутом которой является локальная система координат, заданная в базисе звена.

Интерфейс класса ***IKSBody*** определяет следующие виртуальные методы:

- ***getID()-> uid_t***
  получить уникальный идентификатор
- ***getFrame()-> frame_t***
  получить локальную систему координат
- ***getMass()-> float***
  получить значение массы тела
- ***getCenterOfMass()-> vec3_t***
  получить координаты центра масс
- ***getInertiaTensor()-> mat3_t***
  получить матрицу тензора инерции
- ***createCollisionShape()-> ICollisionShape***
  построить альтернативную геометрическую модель
- ***getMarkerCount()-> int***
  получить число точек сочленения
- ***getMarker(uid_t*** *markerID* ***)-> IKSMarker***
  найти точку сочленения по идентификатору
- ***createMarkerIterator()-> abstract_iterator<IKSMarker>***
  создать итератор для обхода точек сочленения

Для представления кинематических связей используется абстрактный класс ***IKSLink***, через интерфейс которого можно получить доступ к звеньям и точкам сочленения кинематической пары. Одна из точек сочленения рассматривается в качестве ведущей (*Master*), а другая — ведомой (*Slave*). Интерфейс ***IKSLink*** включает в себя следующий набор методов:

- ***getID()-> uid_t***
  получить уникальный идентификатор
- ***getMasterBodyID()-> uid_t***
  получить идентификатор ведущего звена
- ***getMasterBodyMarkerID()-> uid_t***
  получить идентификатор точки сочленения с ведущим звеном
- ***getSlaveBodyID()-> uid_t***
  получить идентификатор ведомого звена

- ***getSlaveBodyMarkerID()-> uid_t***
  получить идентификатор точки сочленения с ведомым звеном
- ***getConstraint()-> IKSConstraint***
  получить ограничения кинематической связи

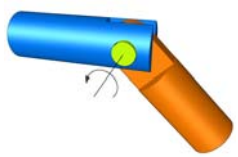
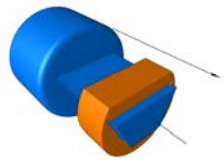
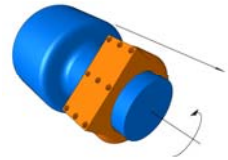
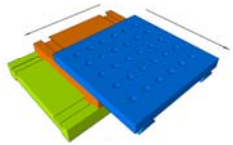
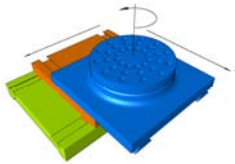
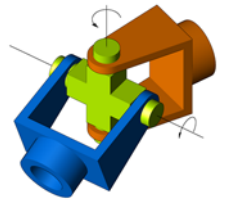
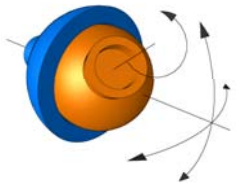
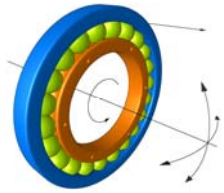
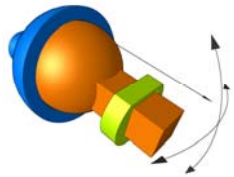
Название класса / Число степеней свободы		Название класса / Число степеней свободы		Название класса / Число степеней свободы	
RevoluteJoint	1	***PrismaticJoint***	1	***CylindricalJoint***	2
					
InPlaneJoint	2	***PlanarJoint***	3	***UniversalJoint***	3
					
SpherialJoint	3	***BearingJoint***	4	***TelescopingJoint***	4
					

Рис. 3. Некоторые классы кинематических ограничений
Fig. 3. Some classes of kinematic constraints.

Согласованное относительное положение звеньев кинематической пары определяется наложенными алгебраическими ограничениями. Для их получения можно воспользоваться виртуальным методом ***getConstraint()***, возвращающим ссылку на объект типа ***IKSConstraint***. Данный абстрактный

класс предусматривает методы, необходимые для определения множества допустимых конфигураций кинематической пары, а также для проверки конфигурации на согласованность с наложенными ограничениями. На основе ограничений, полученных для отдельных кинематических пар, можно сформировать множество допустимых конфигураций для всей кинематической системы. Данная функция естественным образом реализуется на уровне базового класса ***IKinematicSystemObject***.

В состав среды включены конкретные классы ограничений, служащие для задания подвижных соединений с различным количеством степеней свободы и различными комбинациями поступательного и вращательного движения. Данные классы реализуются как наследники базового класса ***IKSConstraint*** (рис. 3).

3.2 Подсистема для определения столкновений

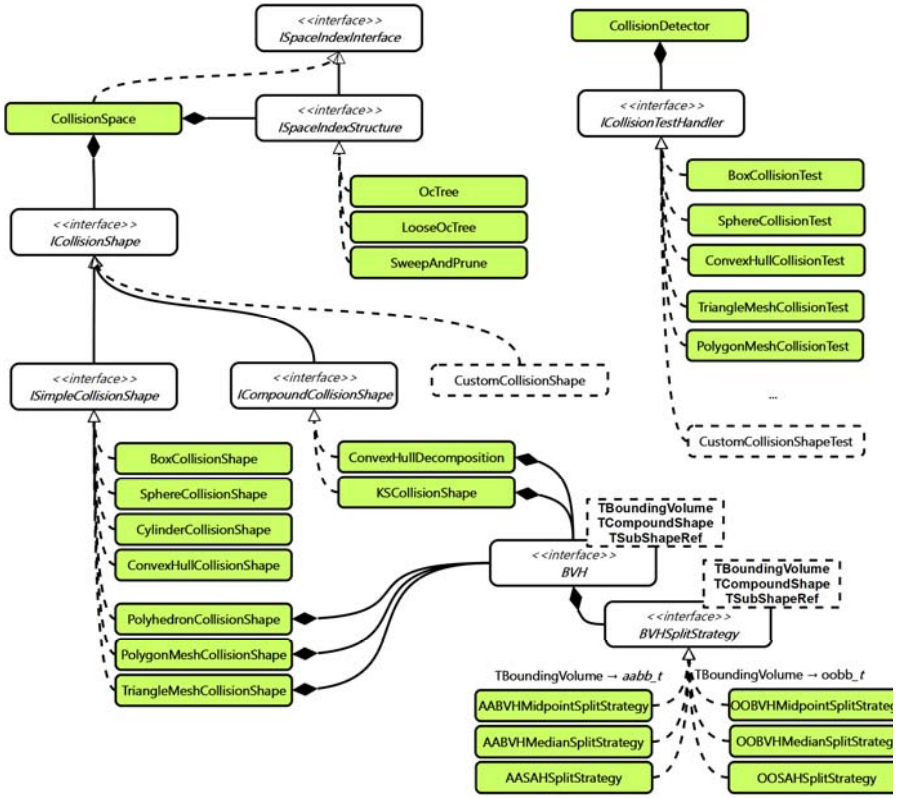


Рис. 4. Диаграмма классов подсистемы определения столкновений
Fig. 4. Collision detection subsystem class diagram.

Подсистема для определения столкновений реализуется как часть среды и представлена классами для задания геометрических моделей типа ***ICollisionShape***, классами представления окружения как композиций объектов и их геометрических моделей — ***StandardWorkSpace*** и ***CollisionSpace*** соответственно, а также классом реализации методов определения столкновения ***CollisionDetector***.

К числу первых относится уже упоминаемый абстрактный класс геометрических моделей ***ICollisionShape***, а также наследуемые от него конкретные классы геометрических примитивов ***BoxCollisionShape***, ***SphereCollisionShape***, ***CylinderCollisionShape***, класс многогранников ***PolyhedronCollisionShape***, классы представления полигональных и триангулированных сеток ***PolygonMeshCollisionShape*** и ***TriangleMeshCollisionShape***, класс выпуклых полигональных оболочек ***ConvexHullCollisionShape***. Обсудим вопросы организации и функционирования подсистемы определения столкновений более подробно.

3.2.1 Геометрические модели

Абстрактный класс ***ICollisionShape*** предназначен для определения интерфейса доступа к альтернативному геометрическому представлению объекта, которое следует использовать для быстрого определения столкновений в ходе исполнения запросов планирования движения. Заметим, что данное представление не обязано совпадать с оригинальной геометрией объекта. Поскольку идентификация столкновений является вычислительно затратной операцией, а методы планирования движения используют ее в качестве базовой, в ряде случаев целесообразно упростить геометрическую модель объекта. Например, она может быть заменена полигональным граничным представлением с меньшим числом граней или примитивными ограничивающими объемами, для которых известны эффективные алгоритмы пересечения.

Интерфейс класса ***ICollisionShape*** представлен следующими виртуальными методами:

- ***getObjectID()->uid_t***
  получить идентификатор исходного объекта (***IWorkSpaceObject***)
- ***getBoundingBox()->aabb_t***
  получить ограничивающий AABB параллелепипед
- ***setTransform(uid_t bodyID, mat4_t bodyTransform)***
  применить трансформацию к объекту с заданным идентификатором
- ***getMargin()->float***
  получить оценку точности геометрического представления (минимальная глубина проникновения, при которой следует идентифицировать пересечение объектов)

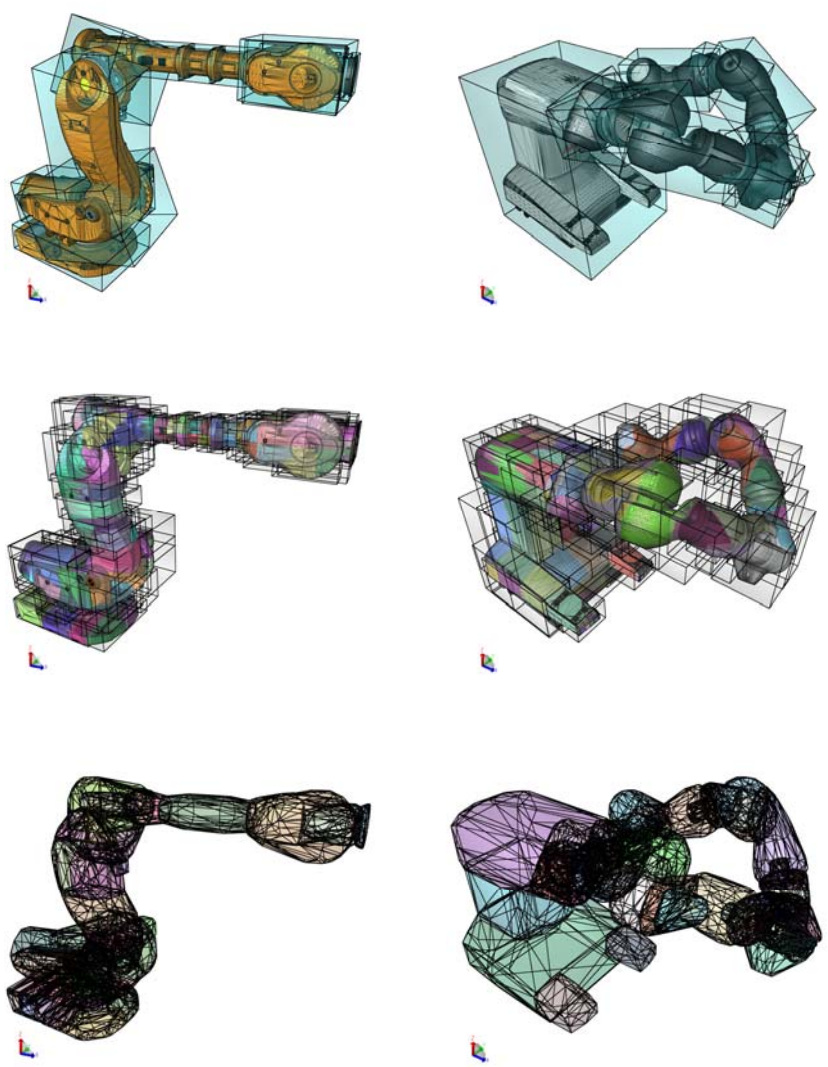


Рис. 5. Альтернативные геометрические модели составных трехмерных объектов: OOBB-дерево, AABB-дерево, декомпозиционное представление выпуклыми оболочками
Fig. 5. Alternative geometry models of compound 3D object: OOBB-tree, AABB-tree, convex hull decomposition.

Поскольку объект окружения и его геометрические представления взаимосвязаны, при обнаружении пересечений могут быть указаны конфликтующие объекты, а также уточнен характер их пересечений.

Среда предоставляет набор готовых к использованию классов геометрических моделей, которые могут быть выбраны разработчиком в качестве альтернативных представлений объектов окружения. Во внимание могут приниматься особенности целевого приложения, способы представления трехмерных данных, требуемая точность локализации столкновений, а также имеющиеся вычислительные ресурсы.

Классы геометрических моделей организованы в виде единой иерархии, наследуемой от базового класса ***ICollisionShape***. Абстрактные классы простых и составных моделей ***ISimpleCollisionShape*** и ***ICompoundCollisionShape*** являются специализациями базового и уточняют его методы.

Простые модели представлены конкретными классами геометрических примитивов ***BoxCollisionShape***, ***SphereCollisionShape***, ***CylinderCollisionShape***, классами многогранников ***PolyhedronCollisionShape***, ***ConvexHullCollisionShape*** и классами сеток ***TriangleMeshCollisionShape***, ***PolygonMeshCollisionShape***. Как правило, в качестве альтернативных геометрических представлений используются полигональные сетки. Гранями таких сеток обычно являются треугольники, четырехугольники или другие простые многоугольники, для которых операции взаимного пересечения в пространстве реализуются относительно просто. Реализации упомянутых классов многогранников и полигональных сеток рассчитаны на более общий случай и допускают задание граней в виде невыпуклых многоугольников и многоугольников с дырками.

Составные геометрические модели представлены классами ***KSCollisionShape*** и ***ConvexHullDecomposition***. Первый реализует составную геометрическую модель кинематической конструкции, второй — декомпозиционное представление произвольного многогранника на основе выпуклых оболочек [15].

Геометрическая модель всего окружения реализуется конкретным классом ***CollisionSpace***. Данный класс позволяет выполнить обход моделей всех объектов, а также синхронизовать их с текущим представлением окружения как результат реакции на происходящие в нем события.

Обновления геометрических моделей в классе ***CollisionSpace*** реализуются с помощью следующих методов:

- ***rebuild()***
  выполняет полное обновление моделей для всех объектов окружения
- ***rebuildObject(uid_t*** *objectID* ***)***
  обновляет модель заданного объекта
- ***updateObjectState(uid_t*** *objectID* ***)***
  устанавливает модель объекта в заданное положение

- ***removeObject(uid_t** objectID **)***
  удаляет заданный объект

3.2.2 Определение столкновений

Определение столкновений в сложном масштабном окружении представляет собой серьезную проблему. Вычислительная сложность определения столкновений может быть существенно уменьшена при использовании пространственных индексов. Основное назначение индексов — локализация потенциальных столкновений за относительно небольшое время на, так называемой, широкой фазе. Выявленные возможные столкновения затем анализируются с использованием точных и вычислительно сложных алгоритмов на узкой фазе. Тем самым, минимизируются затраты на определение столкновений за счет дешевых негативных тестов на пересечения, а точные алгоритмы применяются избирательно только для выявленных пар объектов, допускающих пересечения.

Для реализации подобной стратегии подсистема поддерживает два вида пространственных индексов. Первый использует пространственную декомпозицию всего моделируемого окружения на основе регулярных сеток и позволяет выделить потенциально пересекающиеся группы объектов, которые принадлежат одним пространственным ячейкам [16]. Другой вид индексов — иерархии ограничивающих объемов (BVH) (рис. 5), которые строятся индивидуально для каждой геометрической модели объекта и позволяют выделить пары объектов, элементы которых допускают пересечения. В качестве ограничивающих объемов обычно применяют AABB и OOBB параллелепипеды [17,18]. Рассмотрим вопросы реализации и применения индексов более подробно.

Индексы пространственной декомпозиции строятся для модели всего окружения ***CollisionSpace*** и реализуются классами с общим интерфейсом ***ISpaceIndexStructure***. Данный интерфейс определяет методы построения, инкрементального обновления и применения индекса при поиске ближайших соседей и локализации столкновений независимо от алгоритмических и программных особенностей его реализации. Поведение индекса делегируется соответствующему объекту ***CollisionSpace***, который поддерживает его в состоянии согласованном с геометрической моделью окружения. При изменениях окружения индекс автоматически перестраивается.

Запросы поиска столкновений вынесены в отдельный абстрактный интерфейс ***ISpaceIndexSearchInterface***, который наследуют оба класса ***ISpaceIndexStructure*** и ***CollisionSpace***. Запросы представлены следующими виртуальными методами:

- ***intersectionTest(ICollisionShape** object**, function<bool(ICollisionShape** neighbour**)>** callback **)-> bool***
  устанавливает факт пересечения заданного объекта с окружением

- ***clearanceTest(ICollisionShape** object**, function<float(ICollisionShape** neighbour**)>** callback **) -> float***
  осуществляет поиск кратчайшего расстояния между заданным объектом и окружением

Метод ***intersectionTest*** осуществляет поиск объектов, которые потенциально пересекаются с заданным. Результаты поиска возвращаются через функцию обратного вызова, в которой выполняется точное пересечение геометрических моделей. Это позволяет вызывающему коду прервать операцию при обнаружении первого пересечения и повысить эффективность исполнения запросов. Аналогичным образом метод ***clearanceTest*** принимает в качестве входного параметра функцию точного определения расстояния между объектами. Промежуточные результаты используются для динамического уменьшения радиуса поиска и исключения вызовов вычислительно сложной операции для объектов, расположенных на значительном удалении от заданного и не влияющих на конечный результат.

Разработчику предоставляется возможность использовать альтернативные реализации пространственных индексов, основанных на октодеревьях (***OcTree***), октодеревьях с релаксацией границ (***LooseOcTree***) [19] и сортированных списках ограничивающих объемов (***SeepAndPrune***) [20]. При необходимости разработчик может реализовать собственные методы пространственной локализации объектов и определения стокновений с учетом особенностей решаемых прикладных задач.

За реализацию узкой фазы определения столкновений отвечает класс ***CollisionDetector***. Он обеспечивает регистрацию обработчиков столкновений для каждой пары геометрических моделей, используемых в представлении объектов окружения и имеющих тип ***ICollisionShape***. Использование для этого хэш-таблицы с ключом в виде пары идентификаторов геометрических моделей позволяет ускорить поиск и применение обработчиков при анализе окружения, состоящего из разнотипных объектов. Сами обработчики наследуют общий интерфейс ***ICollisionTestHandler***:

- ***getKey()-> unique_pair<type_index,type_index>***
  получить уникальный ключ обработчика
- ***intersect(ICollisionShape** firstObject**, ICollisionShape** secondObject **) -> bool***
  выполнить проверку пары объектов на пересечение
- ***distance(ICollisionShape** firstObject**, ICollisionShape** secondObject **) -> float***
  найти расстояние между объектами

Обобщенная реализация ***CollisionDetector*** позволяет разработчику поддерживать в подсистеме определения столкновений собственные геометрические модели и регистрировать для них соответствующие функции пересечения и определения расстояния. Реализация данных функций для

определенных типов геометрических объектов имеет свои особенности. Например, классы многогранников, полигональных и треугольных сеток помимо внутреннего представления агрегируют вспомогательный пространственный индекс в виде иерархии ограничивающих объектов, реализуемой шаблонным классом ***TBoundingVolumeHierarchy***. Иерархии ограничивающих объемов строятся единожды при конструировании объектов. При изменении положения объектов нет необходимости перестраивать иерархии заново, поскольку при локализации столкновений соответствующие трансформации могут применяться непосредственно к ограничивающим объемам. Для пересечения выпуклых многогранников, а также оценки возможной глубины проникновения применяется алгоритм расширенных политопов [21], являющийся развитием известного алгоритма Гилберта-Джонсона-Керти [22].

3.3 Моделируемое окружение

Конкретный класс ***StandardWorkSpace*** предоставляет типовую реализацию моделируемого окружения с использованием таких объектов как твердое тело, свободно движущееся в пространстве (***FreeFlyingObject***), машина Дьюбинса [5] (***DubinsCar***) и Ридса-Шеппа [6] (***ReedsSheppCar***), кинематическая цепь (***KinematicChain***). Используемая в классе ***StandardWorkSpace*** фабрика объектов позволяет разработчику добавлять реализации новых типов объектов и, тем самым, расширять возможные постановки задач планирования движения. Поскольку класс представления моделируемого окружения наследуется от ***IWorkSpace***, все реализуемые средой функции, включая методы планирования движения, распространяются и на новые типы объектов.

4. Пакет классов DiscreteSpace

В основе большинства методов планирования движения лежит идея редукции исходной вычислительно сложной задачи к задаче поиска маршрута в графе, разрешимой известными алгоритмами Дейкстры или A* за приемлемое время [2,23]. Вершинам графа ставятся в соответствие точки в рабочем пространстве окружения или в конфигурационном пространстве объекта, а ребрам — бесконфликтные переходы между точками. Элементам графа могут быть приписаны дополнительные данные о стоимости переходов, расстояниях до препятствий окружения, об успешных или неуспешных прецедентах перемещения некоторых объектов и т.п.

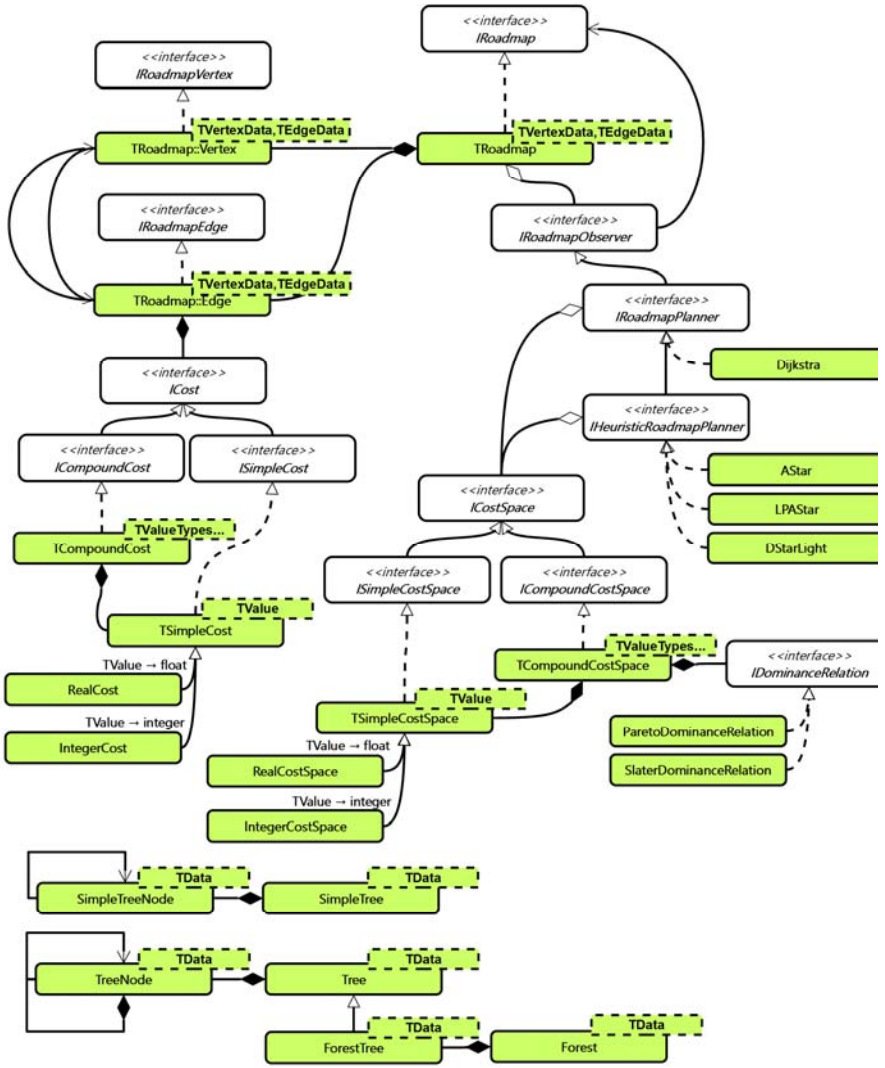


Рис. 6. Диаграмма классов пакета DiscreteSpace

Fig. 6. DiscreteSpace class diagram.

Основным назначением пакета классов DiscreteSpace является задание условий и решение задач теории графии. Обобщенные реализации классов позволяют использовать их в качестве базовых при специализации графов в виде быстрорастущих деревьев и маршрутных сетей. Тем самым обеспечивается возможность многоцелевого использования пакета для представления графов и программной реализации алгоритмов поиска путей в них. В частности, среда предоставляет готовые к использованию классы для представления таких математических объектов как дерево, лес и маршрутная сеть. Обсудим их реализации в виде шаблонных классов более подробно.

4.1 Деревья

Деревья представлены шаблонным классом ***Tree<TData>*** с параметризуемым типом узлов. Класс реализует базовые операции, необходимые для построения и модификации деревьев:

- ***createNode(Node*** *parent* ***)->Node***
  создать новый узел дерева
- ***deleteNode(Node*** *node* ***)***
  удалить узел дерева
- ***moveNode(Node*** *node*, ***Node*** *newParent* ***)***
  перенести ветвь дерева
- ***rotateTo(Node*** *node* ***)***
  развернуть дерево относительно узла node
- ***randomNode()->Node***
  выбрать случайный узел дерева

Приведенные операции удаления узлов и переноса ветвей необходимы, в частности, для построения оптимальных путей RRT* алгоритмом [24].

В других случаях, например, при реализации RRT и RRT-connect алгоритмов [25,26], операции трансформации не требуются и структура представления дерева может быть существенно упрощена за счет хранения однонаправленных ассоциаций узлов на родителей и исключения обратных ассоциаций. Подобный компактный способ представления дерева с необходимым набором базовых операций реализуется в классе ***SimpleTree<TData>***.

4.2 Лес деревьев

Для реализации некоторых алгоритмов сэмплирования требуются операции над множеством деревьев. Для этих целей в среде предусмотрен шаблонный класс ***Forest<TData>***, агрегирующий коллекцию уникально идентифицируемых деревьев класса ***ForestTree<TData>***. Последний является наследником рассмотренного выше класса ***Tree<TData>***. Класс представления леса реализует специфические операции разбиения и слияния деревьев,

необходимые, в частности для реализации алгоритмов с отложенной верификацией ребер [27]. Интерфейс класса включает следующие методы:

- ***findTree(uid_t*** *treeID* ***)-> ForestTree***
  найти дерево по уникальному идентификатору
- ***createTree()-> ForestTree***
  создать дерево
- ***deleteTree(ForestTree*** *tree* ***)***
  удалить дерево
- ***insertEdge(ForestTree*** *sourceTree*, ***Node*** *sourceNode*, ***ForestTree*** *targetTree*, ***Node*** *targetNode* ***)***
  выполнить слияние деревьев *sourceTree* и *targetTree* путем создания ребра, ведущего из узла *sourceNode* в узел *targetNode*
- ***deleteEdge(ForestTree*** *sourceTree*, ***Node*** *node* ***)***
  выполнить разбиение дерева *sourceTree* путем удаления ребра, ведущее в узел *node*

4.3 Маршрутные сети

Для построения маршрутных сетей предназначен шаблонный класс ***TRoadmap<TVertexData,TEdgeData>***, агрегирующий коллекции вершин и ребер соответствующих классов ***TRoadmap::Vertex*** и ***TRoadmap::Edge***. В качестве параметров шаблона выступают пользовательские типы данных, которые используются для представления атрибутов, приписанных вершинам и ребрам. Данный класс ***TRoadmap*** реализует основные операции, необходимые для построения и модификации маршрутных сетей:

- ***createVertexIterator()-> iterator<IRoadmapVertex>***
  создать итератор для обхода вершин
- ***createEdgeIterator()-> iterator<IRoadmapEdge>***
  создать итератор для обхода ребер
- ***createVertex()-> Vertex***
  создать вершину
- ***createEdge(Vertex firstVertex,Vertex secondVertex)-> Edge***
  создать ребро
- ***deleteVertex(Vertex vertex)***
  удалить вершину
- ***deleteEdge(Edge edge)***
  удалить ребро
- ***updateEdgeCost(Edge edge, ICost cost)***
  установить вес ребра

- ***createPath()->TRoadmapPath<Vertex>***
  создать путь
- ***clear()***
  очистить структуру

Шаблонные классы маршрутных сетей и элементов сети наследуются от соответствующихся абстрактных классов ***IRoadmap***, ***IRoadmapVertex***, ***IRoadmapEdge***. В конечном счете, это обеспечивает возможность обобщенной реализации алгоритмов планирования движения на уровне абстрактных классов независимо от способов представления сетей и особенностей доступа к их атрибутам.

4.4 Исчисление стоимости

В задачах поиска оптимальных путей обычно ребрам маршрутной сети приписывают веса или стоимости переходов между смежными вершинами. В зависимости от прикладной постановки стоимость переходов может определять разные критерии поиска. В большинстве случаев решается задача поиска наикратчайшего маршрута в рабочем пространстве, а стоимость переходов — длина маршрута между точками рабочего пространства, соответствующими вершинам сети.

В других случаях решаются задачи маршрутизации с учетом иных, в том числе множественных критериев. Например, в работе [28] ставится задача маршрутизации объекта, в которой предпочтение отдается поступательному движению и минимизируется вращательная составляющая. В работе [29] рассматривается задача поиска безопасных маршрутов, наиболее удаленных от препятствий окружения.

Для унификации способов задания критериев поиска в среде предусмотрены соответствующие классы для представления и исчисления стоимостей. Предполагается, что стоимости представимы абстрактным классом ***ICost***, а абстрактный класс ***ICostSpace*** определяет сигнатуры операций над ними:

- ***null()->ICost***
  получить нулевое значение стоимости
- ***equal(ICost a, ICost b)->bool***
  оператор равенства
- ***less(ICost a, ICost b)->bool***
  оператор сравнения
- ***add(ICost a, ICost b)->ICost***
  оператор сложения

Средой допускается задание условий многокритериального поиска, поэтому абстрактные классы ***ICost*** и ***ICostSpace*** уточняются соответствующими классами для операций над скалярными величинами (***ISimpleCost*** и

ISimpleCostSpace) и векторными величинами (***ICompoundCost*** и ***ICompoundCostSpace***). Первая пара классов реализуется в виде шаблонов ***TSimpleCost<TValue>*** и ***TSimpleCostSpace<TValue>***, параметризуемых конкретным типом скалярной переменной. Вторая пара реализуется в виде шаблонных кортежей ***TCompoundCost<TValues…>*** и ***TCompoundCostSpace<TValues…>***. При этом в классе ***TCompoundCostSpace*** предусмотрена возможность задания бинарного отношения доминирования по Парето или Слейтеру. Данные отношения реализуются соответствующими классами ***ParetoDominanceRelation*** и ***SlaterDominanceRelation*** с общим интерфейсом ***IDominanceRelation***.

4.5 Маршрутизаторы

Алгоритмы поиска путей реализуются в конкретных классах, наследуемых от абстрактного класса ***IRoadmapPlanner*** и получающих доступ к заданной маршрутной сети через его ассоциацию типа ***IRoadmap***. Поиск осуществляется с помощью метода ***findPath(IRoadmapVertex*** *initialVertex****, IRoadmapVertex*** *goalVertex****, IRoadmapPath*** *path* ***)-> bool***.

Для реализации эвристических алгоритмов предназначен абстрактный класс ***IHeuristicRoadmapPlanner***, который является специализацией базового класса ***IRoadmapPlanner*** и дополнительно агрегирует объект типа ***ICostSpace*** для задания эвристической функции приоритизации вершин.

Среда предоставляет реализации нескольких популярных алгоритмов поиска путей, которые представлены конкретными классами ***Dijkstra*** (алгоритм Дейкстры [23]), ***AStar*** (алгоритм A* [23]), ***LPAStar*** (алгоритм LPA* [30]) и ***DStarLight*** (алгоритм D*-light [31]). Тем самым разработчику предоставляется возможность выбора алгоритма маршрутизации, наиболее подходящего для решаемых прикладных задач и применяемых методов планирования движения.

Для планирования в динамических маршрутных сетях класс ***TRoadmap*** реализует механизм оповещения наблюдателей типа ***IRoadmapObserver***. Сам механизм оповещения реализуется в конкретных классах маршрутизаторов, наследуемых от базового ***IRoadmapPlanner***, который в свою очередь наследуется от интерфейса наблюдателя ***IRoadmapObserver***. Данный интерфейс определяет следующие методы реакции на изменения в маршрутной сети:

- ***onAfterVertexAdded(Vertex*** *vertex* ***)***
  добавлена новая вершина
- ***onBeforeVertexRemoved(Vertex*** *vertex* ***)***
  вершина будет удалена
- ***onAfterEdgeAdded(Edge*** *edge* ***)***
  добавлено новое ребро
- ***onBeforeEdgeRemoved(Edge*** *edge* ***)***

ребро будет удалено

- ***onAfterEdgeCostChanged(Edge** edge **)***
  изменился вес ребра
- ***onBeforeClear()***
  структура будет очищена

Данные методы реализуются в конкретных классах маршрутизаторов с учетом характера изменений. Например, в алгоритмах LPA* и D*-light в качестве реакции на изменение веса ребра обновляется приоритетная очередь вершин.

5. Пакет классов StateSpace

Под состоянием или конфигурацией объекта понимается набор значений параметров, однозначно определяющих положение всех точек его геометрической модели в трехмерном пространстве окружения. Обычно используется минимальный набор параметров, соответствующий количеству степеней свободы объекта и определяющий конфигурационное пространство объекта. Задача планирования пути формулируется как задача построения бесконфликтной непрерывной траектории в конфигурационном пространстве объекта, которая соединяет заданную пару точек, соответствующих его начальному и конечному состоянию.

Конфигурационное пространство, допустимое состояние, траектория, генератор конфигураций, планировщик и верификатор траекторий —ключевые математические абстракции, которые положены в основу средств планирования движения в составе объектно-ориентированной среды. Подобные абстракции представлены соответствующими классами и интерфейсами ***IStateSpace***, ***IState***, ***ISampler***, ***IProbabilisticPlanner***, ***TSteeringMethod<TState>***. Обсудим особенности их реализации более подробно.

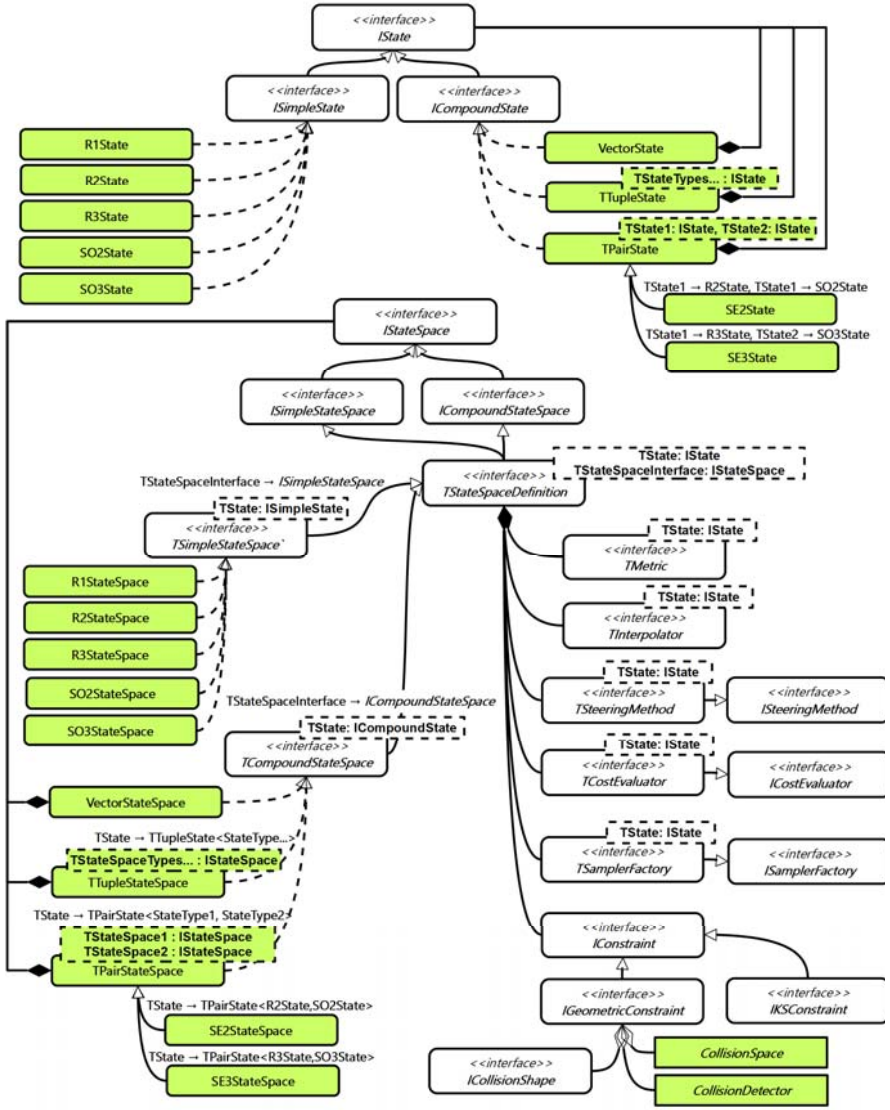


Рис. 7. Диаграмма классов конфигурационного пространства

Fig. 7. Configuration space class diagram.

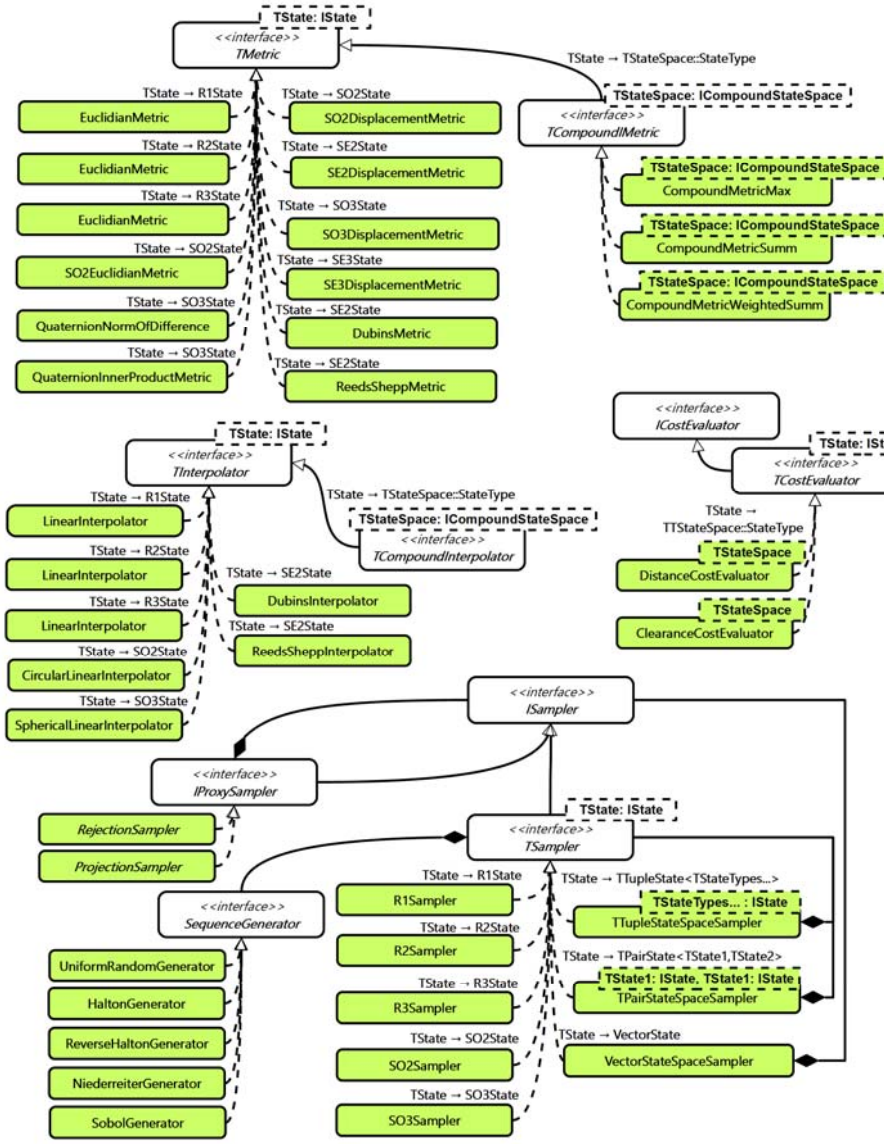


Рис. 8. Диаграмма классов конфигурационного пространства

Рис. 8. Configuration space class diagram.

5.1 Конфигурационные пространства

Выделение абстракций конфигурационного пространства и точки пространства и их представление в среде соответствующими интерфейсами ***IStateSpace*** и ***IState*** обусловлены необходимостью поддержать альтернативные способы задания обобщенных координат объекта. Например, твердое тело, свободно движущееся в трехмерном пространстве, имеет шесть степеней свободы, соответствующих поступательному и вращательному движению. Более сложные объекты могут иметь большее количество степеней свободы. Например, состояние манипуляционного робота может задаваться многомерным вектором, элементы которого определяют углы поворота всех его подвижных шарниров. При этом важно обеспечить возможность обобщенной реализации алгоритмов планирования движения и, в частности популярных сэмплинг алгоритмов, без какой-либо конкретизации явных или неявных способов задания обобщенных координат объекта. Интерфейс ***IStateSpace*** определяет необходимый для этого набор методов:

- ***getSpaceDimension()->integer***
  возвращает размерность конфигурационного пространства
- ***distance(IState first, IState second)->real***
  возвращает значение расстояния между конфигурациями
- ***interpolate(IState start, IState end, real time)->IState***
  возвращает интерполированную точку по заданной начальной и целевой конфигурации и параметру $time \in [0; 1]$
- ***isFree(IState state)->bool***
  верифицирует заданную конфигурацию
- ***isFree(IState state)->(bool,float)***
  верифицирует заданную конфигурацию и возвращает значение расстояния до ближайшего препятствия
- ***getSteeringMethod()->ISteeringMethod***
  предоставляет доступ к верификатору путей
- ***getCostEvaluator()->ICostEvaluator***
  предоставляет доступ к вычислителю стоимости движения
- ***createUniformSampler()->ISampler***
  создает генератор конфигураций, равномерно распределенных в заданной области

Вопросы применения данных методов при реализации сэмплинг алгоритмов подробно обсуждаются в следующих разделах. В данном разделе остановимся на специализациях базового класса ***IStateSpace***, в частности, на абстрактных классах ***ISimpleStateSpace*** и ***ICompoundStateSpace***, служащих для определения простых и составных конфигураций. В отличие от ***ISimpleStateSpace***, класс ***ICompoundStateSpace*** предполагает задание конфигурационного пространства

составного объекта как прямого декартова произведения пространств соответствующих частей. Выделение перечисленных абстракций позволяет унифицировать основные операции анализа конфигураций, а также упростить процедуру их обратного преобразования в рабочее трехмерное пространство.

ICompoundStateSpace конкретизируется классами, определяющими конфигурационное пространство как произведение фиксированного и переменного числа операндов: ***TPairStateSpace<TSpace1,TSpace2>***, ***TTupleStateSpace<TStateSpace...>***, ***VectorStateSpace***.

Например, конфигурационное пространство твердого тела, свободное движение которого в трехмерном пространстве определяется группой преобразований $SE(3) = R^3 \times SO(3)$, может быть задано следующей специализацией шаблона ***SE3Space=TPairStateSpace<R3Space,SO3Space>***. Конфигурационные пространства колесных механизмов ***DubinsSpace*** и ***ReedsSheppSpace*** могут быть определены путем наследования от класса ***SE2Space=TPairStateSpace<R2Space,SO2Space>***, соответствующего группе преобразований $\mathrm{SE}(2) = \mathrm{R}^2 \times \mathrm{SO}(2)$.

Наконец, класс ***VectorStateSpace*** позволяет задавать конфигурационные пространства динамически в ходе выполнения программы, поскольку поддерживает неоднородную коллекцию объектов типа ***IStateSpace*** переменного размера. В частности, данная возможность полезна при работе с кинематическими системами, описание которых хранится в файлах или формируется непосредственно в ходе пользовательской сессии.

Для определения конкретных классов конфигурационных пространств со строгим контролем соответствия типов и гибкой настройкой поведения предназначен шаблонный класс ***TStateSpaceDefinition<TState,TStateSpaceInterface>***. Параметр шаблона ***TState*** соответствует конкретному классу конфигурации типа ***IState***, а ***TStateSpaceInterface*** — интерфейсу простого или составного конфигурационного пространства ***ISimpleStateSpace*** или ***ICompoundStateSpace***.

Данный класс предусматривает агрегацию ключевых алгоритмических компонентов, необходимых для реализации методов интерфейса ***IStateSpace***. Поскольку алгоритмические компоненты одного назначения представлены единой иерархией классов, в классе возможна настройка альтернативных алгоритмов соответствующих типов. В данной проектной схеме делегирования операции определения расстояния между конфигурациями и интерполяции точек выполняются установленными объектами классов ***TMetric<TState>*** и ***TInterpolator<TState>***. Верификация линейных сегментов пути реализуется назначенным объектом класса ***TSteeringMethod<TState>***. Оценка стоимости перехода между конфигурациями делегируется соответствующему объекту класса ***TCostEvaluator<TState>***, а генерация конфигураций с равномерным рапределением — объекту класса ***TSamplerFactory<TState>***.

5.2 Генераторы конфигураций

Генерация конфигураций является ключевым элементом всех сэмплинг методов планирования движения и имеет свои особенности. Например, в работе [32] показано, что наивный метод, основанный на случайном выборе координат, соответствующих эйлеровым углам, не обеспечивает равномерное распределение конфигураций в пространстве SO(3) и предложен метод на основе единичных кватернионов.

Генераторы в среде представлены абстрактным классом ***ISampler***, в котором определен виртуальный метод получения очередной конфигурации ***generateState()->IState***. Метод допускает альтернативные реализации генераторов псевдослучайных чисел, квазислучайных последовательностей и регулярных сеток [1] в наследуемых конкретных классах.

Среда предоставляет набор классов для построения выборок с равномерным распределением для каждого конкретного класса конфигурационного пространства. В частности, доступны генератор псевдослучайных чисел ***UniformRandomGenerator***, генераторы последовательностей Холтона ***HaltonGenerator*** и ***ReverseHaltonGenerator***, генераторы Соболя ***SobolGenerator*** и Нидеррайтера ***NiederretierGenerator***. На этапе конструирования генераторов устанавливаются границы области сэмплирования.

В реализациях генераторов предусмотрены возможности задания эвристик сэмплирования, позволяющих формировать выборки преимущественно из допустимых конфигураций, имеющих перспективу стать точками конструируемых путей и снижающих вычислительные расходы на поиск подобных точек в равномерно распределенных выборках. Для этих целей используется абстрактный класс ***IProxySampler***, который являясь наследником ***ISampler***, предоставляет метод получения генерируемых конфигураций. В процессе отбора перспективных конфигураций используется вспомогательный генератор исходных конфигураций, например, с равномерным распределением, поэтому класс ***IProxySampler*** агрегирует требуемый для этого объект типа ***ISampler***.

Абстрактный класс ***IProxySampler*** специализируется конкретными классами ***RejectionSampler*** и ***ProjectionSampler***, реализующими две основные стратегии эвристического сэмплирования конфигураций: путем отклонения неперспективных точек на основе заданного условия или путем их последующей трансформации.

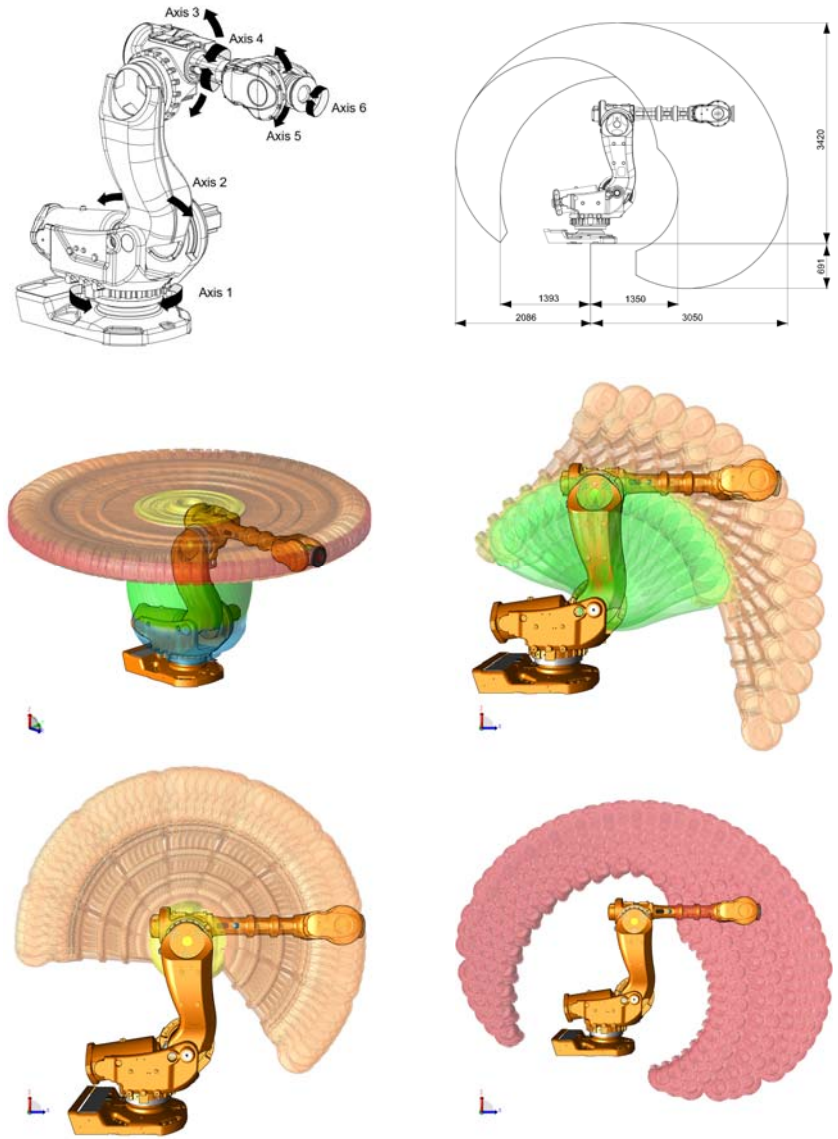


Рис. 9. Результат работы генератора точек в конфигурационном пространстве семизвенного робота
Fig. 9. Example of 6DOF robot configuration space sampling.

Первый подход используется для генерации точек вдоль границ препятствий [33–35] и для отсечения точек по области видимости [36]. Класс

RejectionSampler использует указатель на функцию с сигнатурой ***acceptStateRule(IState** state **)-> bool*** для проверки соответствия конфигурации заданному условию и пороговое значение, определяющее максимальное количество предпринимаемых попыток генерации надлежащей конфигурации.

Аналогичным образом, класс ***ProjectionSampler*** использует указатель на функцию с сигнатурой ***transformStateRule(IState state)->State*** для трансформации конфигурации в соответствии с заданным правилом. Данный класс может использоваться для построения точек, наиболее удаленных от препятствий [37], или для построения проекций на подпространство конфигураций, удовлетворяющих кинематическим ограничениям [38,39].

5.3 Верификаторы путей

Абстрактный класс ***TSteeringMethod<TState>*** определяет единый интерфейс верификаторов путей, предназначенных для проверки возможности бесконфликтного перехода между парой точек конфигурационного пространства и верификации ребер маршрутной сети. Интерфейс содержит следующий набор виртуальных методов:

- ***verifyMotion(IStateSpace** cspace**, IState** start**, IState** end **)-> bool***
  устанавливает факт возможности бесконфликтного перехода из конфигурации *start* в конфигурацию *end*
- ***verifyMotion(IStateSpace** cspace**, IState** start**, IState** end **)–> (bool,IState)***
  возвращает последнюю бесконфликтную точку при переходе из конфигурации *start* в конфигурацию *end*
- ***verifyMotion (IStateSpace** cspace**, IState** start**, IState** end **)-> (bool,IState[])***
  возвращает массив бесконфликтных точек при переходе из конфигурации *start* в конфигурацию *end*

В состав среды включен конкретный класс ***DiscreteSteeringMethod***, который наследует интерфейс ***TSteeringMethod*** и реализует процедуру верификации путем дискретизации отрезка с сопутствующей проверкой промежуточных точек на столкновения (рис. 10). Шаг дискретизации определяется значением погрешности, которое устанавливается в качестве параметра при конструировании экземпляров конфигурационного пространства типа ***IStateSpace***. Значение погрешности соответствует расстоянию между точками, определяемому заданной метрикой конфигурационного пространства с учетом габаритов трехмерного объекта.

Выделение абстрактного класса ***TSteeringMethod*** обеспечивает возможность реализации более эффективных способов верификации путей с учетом особенностей прикладных задач. Например, в случае простых твердотельных объектов повысить эффективность верификации путей можно с помощью техники протяжек ограничивающих выпуклых оболочек или алгоритмов

непрерывного определения столкновений (Continuous Collision Detection) [40,41].

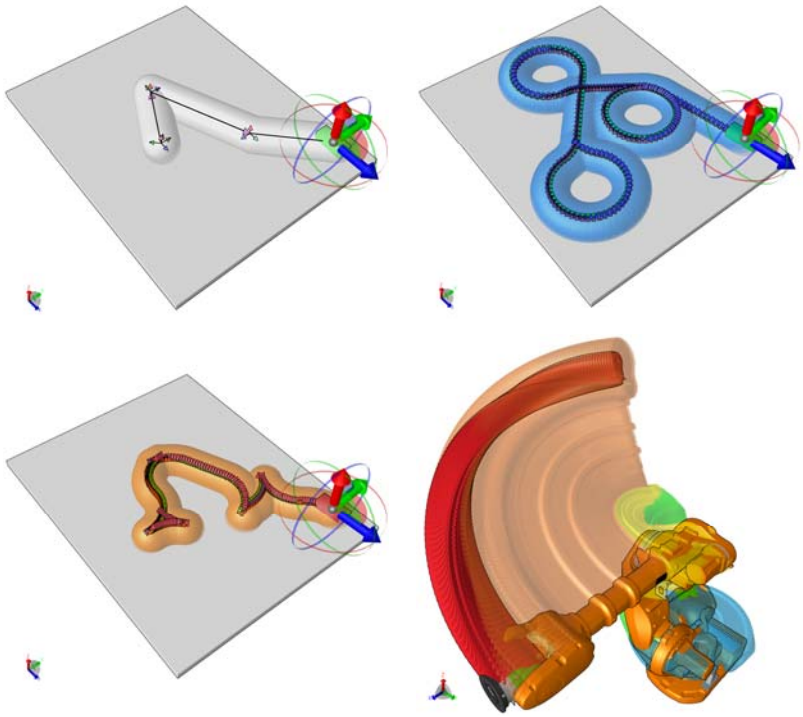


Рис. 10. Верификация пути для различных моделей движения
Fig. 10. Path verification for different types of motion.

5.4 Вычислитель стоимости переходов

Алгоритмы планирования движения, нацеленные на поиск оптимальных путей в процессе их построения, в частности, TRRT [29], RRT* [24] и TRRT* [42], требуют задания функции для оценки качества решений. Для этих целей служит абстрактный класс ***TStateCostEvaluator<TState>***, наследующий интерфейс стоимостного пространства ***ICostSpace*** и дополнительно определяющий виртуальный метод ***evaluate(IState*** *from****, IState*** *to* ***)-> ICost***. Данный метод выступает в роли функции стоимости перехода между заданными конфигурациями и реализуется в конкретных наследуемых классах

вычислителей. Тем самым, средой допускается поддержка альтернативных критериев оптимальности.

Для поиска оптимальных путей по критериям длины и удаленности от препятствий реализованы и включены в состав среды классы ***DistanceStateCostEvaluator*** и ***ClearanceStateCostEvaluator***. Для задания критериев, определяемых пользователем, может быть использован вспомогательный абстрактный шаблонный класс ***TStateCostEvaluator<TStateSpace,TCostSpace>***, специализация которого выполняется в результате задания конкретных классов конфигурационного и стоимостного пространств.

6. Подсистема локального планирования движения

6.1 Запросы планирования движения

Алгоритмы локального планирования движения в конфигурационном пространстве объекта реализуются на основе абстрактного класса ***IProbabilisticPlanner***, который предоставляет внешний интерфейс запросов планирования в виде ***findPath(IState*** *initialState****, IState*** *goalState****, IPath*** *path* ***)-> PlanningResult***. Входными параметрами метода являются начальная и целевая конфигурации объекта, а возвращаемые результаты — ссылка на построенный путь типа ***IPath*** и значение перечислимого типа ***PlanningResult***, отражающее статус выполнения запроса планирования: *SUCCESS* – путь успешно найден, *FAILURE* – бесконфликтный путь не существует, *PROBABLY_FAILURE* – путь не найден, *INVALID_INPUT* – неверно заданы входные данные, *INTERNAL_ERROR* – внутренняя ошибка программы.

Найденные пути представляются шаблонным классом ***TPath<TStateSpace>***, наследующим интерфейс ***IPath*** и параметризуемым конкретным типом конфигурационного пространства, в котором они строятся. Каждый путь представляет собой упорядоченную коллекцию бесконфликтных конфигураций. При этом подразумевается, что сегменты путей между соседними конфигурациями, построенные в соответствии с предопределенной интерполяционной функцией, также неконфликтны.

Поскольку запросы планирования движения объекта могут быть одиночными и множественными, каждый планировщик типа ***IProbabilisticPlanner*** хранит ссылку на конфигурационное пространство объекта, в котором разрешаются подобные запросы и для которого может быть уже развернуты деревья поиска или маршрутная сеть. Данная ссылка реализуется соответствующей ассоциацией класса и устанавливается при конструировании планировщиков.

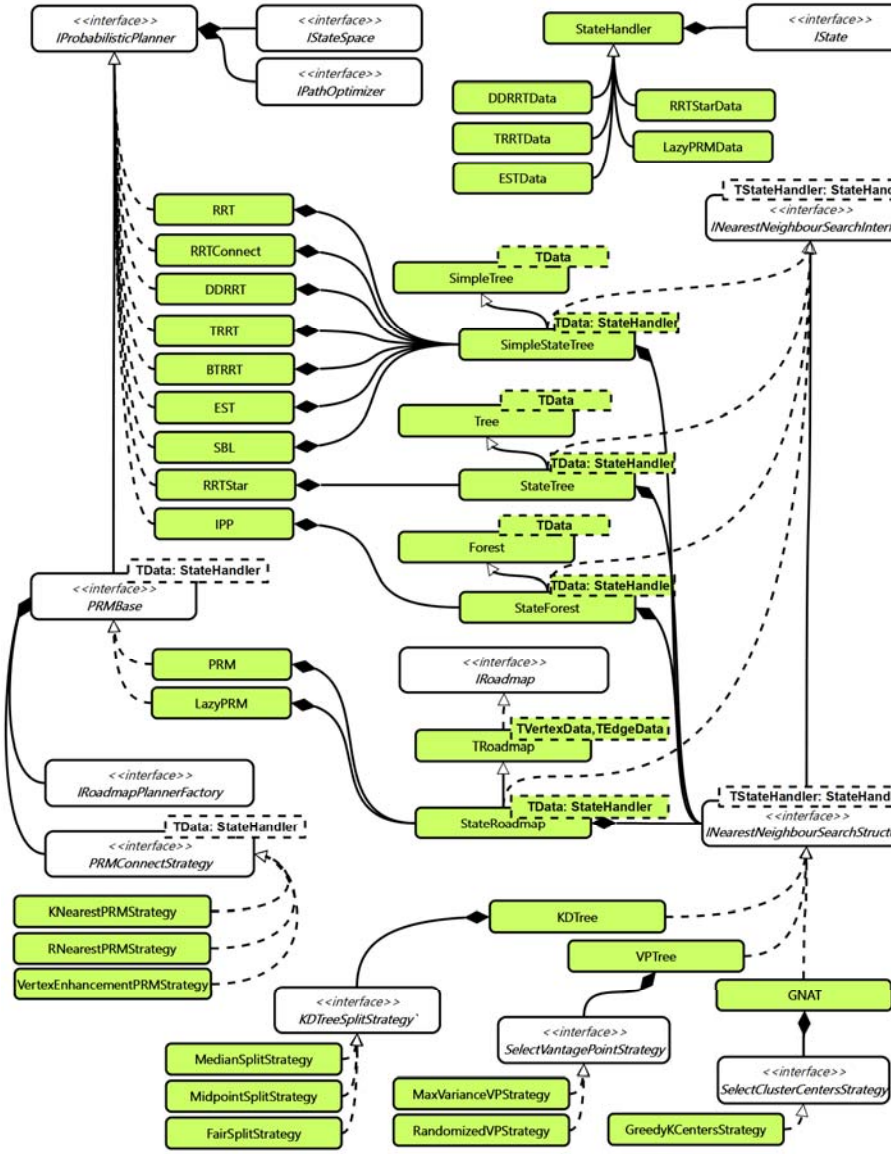


Рис. 11. Диаграмма классов локального планирования движения
Fig. 11. Local planning class diagram.

Общая алгоритмическая схема разрешения запроса планирования реализуется непосредственно в классе ***IProbabilisticPlanner***, в котором проводится контроль входных данных, предобработка, необходимая, например, для реализации алгоритмов на основе вероятностных маршрутных сетей, сам поиск и постобработка найденных путей для их оптимизации. Сами алгоритмы реализуются в наследуемых или ассоциируемых классах. В частности, на основе класса ***IProbabilisticPlanner*** реализуется обсуждаемое ниже семейство сэмплинг алгоритмов с различными эвристическими стратегиями поиска.

6.2 Пространственные индексы

Одной из базовых операций, используемых при реализации сэмплинг алгоритмов, является поиск ближайших соседей в представлении поискового дерева. Для повышения производительности обычно используют пространственные индексы, которые строятся на множестве точек, полученных в результате сэмлирования конфигурационного пространства.

Среда предоставляет несколько готовых к использованию классов, реализующих необходимые индексные структуры, а именно: ***KDTree*** (kD-дерево [43]), ***GNAT*** (Geometric Near-neighbor Access Tree [44,45]) и ***VPTree*** (Vantage Point Tree [46]). Поскольку затраты на построение индексов и исполнение запросов могут существенно варьироваться в зависимости от размерности пространства, характера распределения допустимых конфигураций и алгоритма сэмплирования, организация классов индексов предусматривает возможность подмены альтернативных реализаций. С этой целью конкретные реализации классов индексов унаследованы от абстрактного класса ***INearestNeighbourSearchStructure*** и его базового интерфейса ***INearestNeighbourSearchInterface<TStateHandler>***, определяющего запросы поиска ближайших соседей в известных постановках:

- ***nnSearch(IState state) -> TStateHandler***
  поиск ближайшей точки
- ***kNNSearch(IState state, integer K) -> TStateHandler[]***
  поиск K ближайших точек
- ***rNNSearch(IState state, float R) -> TStateHandler[]***
  поиск точек, находящихся в радиусе R от заданной точки

Поскольку реализация пространственных индексов предполагает задание метрики, базовый класс индексов определяет соответствующую ассоциацию на конфигурационное пространство типа ***IStateSpace*** с необходимой функцией определения расстояния между заданными точками. Параметр шаблона ***StateHandler*** имеет тип вершины поискового дерева или маршрутной сети, связанной с соответствующей точкой конфигурационного пространства.

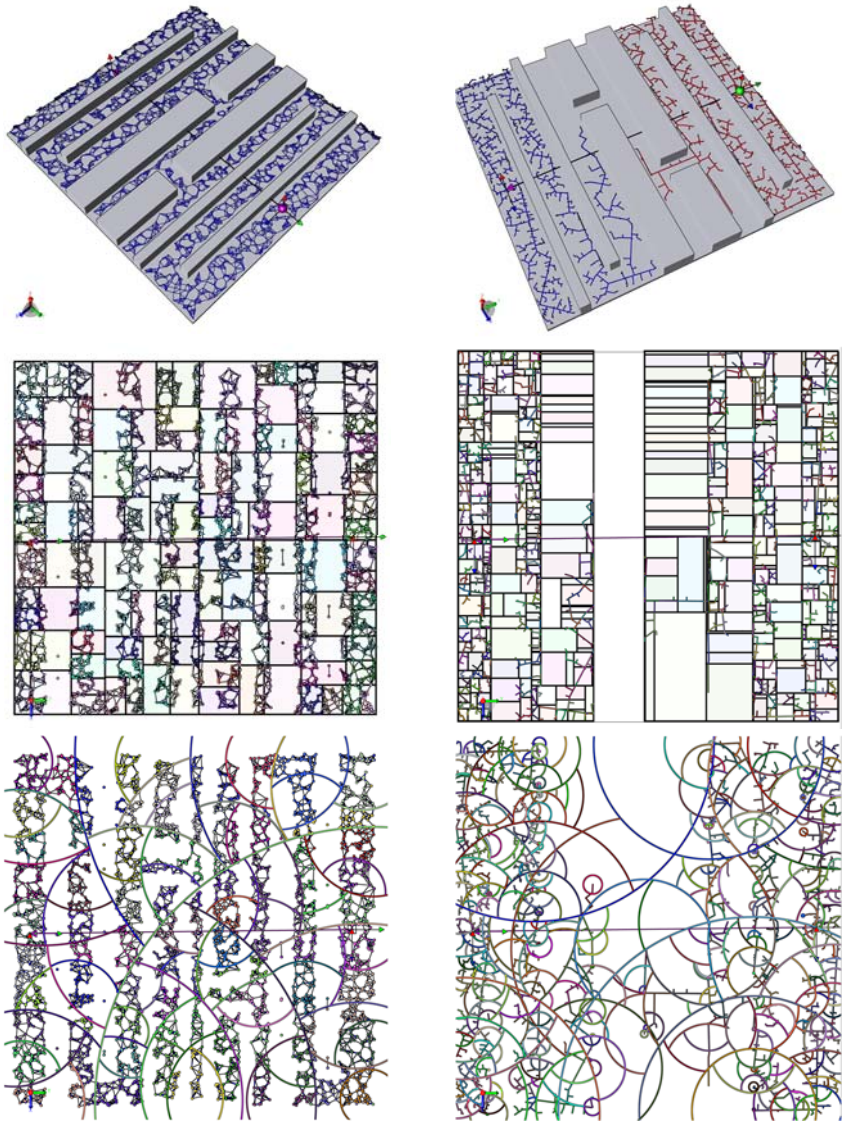


Рис. 12. kD-деревья и vp-деревья,построенные на множестве вершин вероятностной маршрутной сети (PRM) и быстро растущих случайных деревьев (RRT-connect)

Fig. 12. kD-trees and u vp-trees built on PRM and RRT-connect vertex set.

Для удобства реализации различных алгоритмов планирования движения в состав среды включены шаблонные классы ***StateTree<TNodeData>***, ***StateForest<TNodeData>*** и ***StateRoadmap<TVertexData,TEdgeData>***, которые, являясь специализациями соответствующих базовых классов ***Tree***, ***Forest*** и ***Roadmap***, наследуют необходимые операции работы с графами. Помимо этого, данные классы реализуют наследуемый интерфейс ***INearestNeighbourSearchInterface***, делегируя выполнение операций поиска ближайших соседей агрегируемому пространственному индексу типа ***INearestNeighbourSearchStructure***. Индекс обновляется автоматически при добавлении, удалении, модификации элементов поисковых деревьев и маршрутных сетей.

6.3 Семейство локальных планировщиков путей

Рассмотренные выше классы среды служат удобным инструментарием для реализации популярных алгоритмов планирования движения, основанных на поисковых деревьях и вероятностных маршрутных сетях. Обобщенные шаблонные реализации позволяют относительно просто конфигурировать интерфейсы и классы среды для разработки приложений и настройки алгоритмов с учетом плотности покрытия в окрестности вершин [47], количества успешных и неуспешных попыток распространения [48], динамической области сэмлирования [26], суммарной стоимости пути из корня дерева [24] и т.п.

Среда предоставляет развитое семейство локальных планировщиков путей, реализованных в виде соответствующих классов-наследников ***IProbabilisticPlanner***. Классы ***RRT*** и ***EST*** реализуют алгоритмы на основе поисковых деревьев Rapidly Exploring Random Trees и Expansive-Spaces Trees соответственно. Алгоритмы с онлайн-оптимизацией деревьев Transition-based RRT и RRT* реализованы в классах ***TRRT*** и ***RRTStar***. Алгоритмические версии, адаптированные для двунаправленного поиска, представлены классами ***RRTConnect***, ***SBL*** и ***BTRRT***. Класс ***IPP*** реализует диффузионный алгоритм с отложенной проверкой на столкновения (Iterative Diffuse Path Planner). Наконец, алгоритмы на основе вероятностных маршрутных сетей, ориентированные на обработку множественных запросов поиска, представлены классами ***PRM*** и ***LazyPRM***.

6.4 Оптимизация путей

Случайный характер поиска допустимых конфигураций и дискретный способ построения путей сэмплинг алгоритмами крайне негативно влияют на качество получаемых решений. Естественными требованиями, предъявляемыми к найденным путям, являются их минимальная длина, гладкость, наибольшее удаление от препятствий окружения и т.п. В связи с этим постобработка найденных путей является важным этапом улучшения их качества, который предусматривается классом локальных планировщиков ***IProbabilisticPlanner***.

С этой целью на заключительном этапе выполнения запроса планирования вызывается метод ***optimize(IStateSpace*** *cspace*, ***IPath*** *path* ***)-> IPath*** в назначенном оптимизаторе путей типа ***IPathOptimizer***. Данный метод вызывается автоматически при успешном выполнении предыдущих этапов и, в частности, при наличии хотя бы одного найденного бесконфликтного пути. Оптимизация пути выполняется с помощью алгоритмов сглаживания, укорачивания и построения ретракта [28], реализуемых соответствующими классами ***PathSmoothing***, ***PathShortening*** и ***PathRetractor*** — наследниками ***IPathOptimizer***.

7. Подсистема глобального планирования движения

Рассмотренные выше программные средства среды обеспечивают задание условий и решения задач планирования движения в локальной статической постановке. Вместе с тем, на практике возникает необходимость решения более сложных задач, связанных с построением путей в сложном динамическом окружении. С этой целью в состав среды включены соответствующие средства, составляющие подсистему глобального планирования и реализующие общую вычислительную стратегию, предложенную и апробированную авторами ранее [13,49].

Стратегия подразумевает построение единой маршрутной сети в рабочем трехмерном пространстве окружения, которая затем используется для принятия решений о наиболее перспективных маршрутах. Отобранные маршруты верифицируются и при необходимости корректируются локальным планировщиком с учетом геометрии конкретного объекта и наложенных на него кинематических ограничений. Представление сети обновляется синхронно с событиями, происходящими в динамическом окружении. Для построения сети могут применяться методы пространственной декомпозиции, диаграммы Вороного или планы окружения, построенные вручную.

Принципы организации и функционирования подсистемы обеспечивают возможность гибкого конфигурирования глобального планировщика из компонентов, реализующих альтеративные способы построения случайных деревьев и маршрутных сетей, а также осуществляющих их верификацию и коррекцию рассмотренными выше алгоритмами локального планирования.

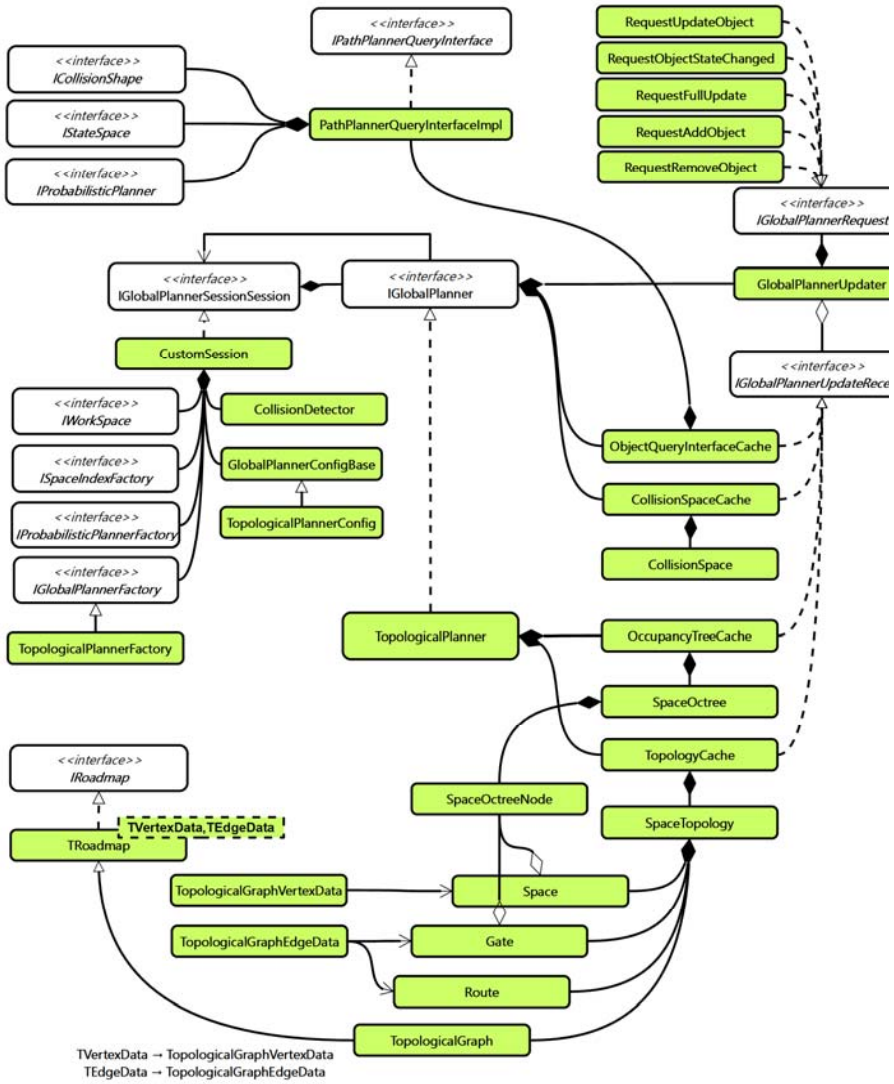


Рис. 13. Диаграмма классов подсистемы глобального планирования

Fig. 13. Global planning subsystem class diagram

7.1 Глобальный планировщик

Глобальный планировщик выполняет две основные функции: формирование единой маршрутной сети на основе пространственного анализа трехмерного окружения и разрешение запросов планирования движения с помощью развернутой сети. Эффективное исполнение множественных запросов предполагает поддержку согласованного представления сети на протяжении всей пользовательской сессии. Это же относится и к связанным с ней вспомогательным данным, в частности, альтернативному геометрическому представлению окружения и пространственным индексам. В случае динамического окружения сеть и вспомогательные данные должны обновляться синхронно с происходящими событиями, причем инкрементальным образом. Поскольку подобные обновления являются вычислительно затратными операциями, они выполняются в фоновом режиме.

За реализацию перечисленных функций в среде отвечает класс планировщиков ***IGlobalPlanner***. Остановимся более подробно на механизме сообщений, с помощью которого происходят обновления в подсистеме глобального планирования.

Сообщения типизируются и представляются следующим набором классов, наследуемых от абстрактного ***IGlobalPlannerRequest***:

- ***RequestAddObject***
  создан новый объект
- ***RequestRemoveObject***
  объект удален
- ***RequestUpdateObject***
  изменилось представление объекта
- ***RequestObjectStateChanged***
  изменилось состояние объекта
- ***RequestFullUpdate***
  требуется полное обновление всех насчитываемых данных

Обновление вспомогательных данных реализуется классами обработчиков как реакция на полученные сообщения. Данные классы наследуют от абстрактного ***IPathPlannerUpdateReceiver*** следующие методы:

- ***getUniqueName()->string***
  возвращает уникальное имя обработчика
- ***dependsOn(string*** *receiverName* ***)-> bool***
  установливает факт зависимости от другого обработчика с заданным именем
- ***fullUpdate()***
  выполняет полное обновление
- ***smartUpdate(IPathPlannerRequest*** *request* ***)***

  выполняет инкрементальное обновление в зависимости от типа сообщения

Обработчик сообщений автоматически регистрируется в диспетчере сообщений при конструировании. Сообщения попадают в очередь диспетчера и затем рассылаются всем зарегистрированным обработчикам.

Для поддержки функций глобального планирования в целевом приложении необходимо разработать класс пользовательской сессии с предопределенным интерфейсом ***IGlobalPlannerSession*** и следующими виртуальными методами конфигурирования подсистемы планирования:

- ***getConfig()->GlobalPlannerConfig***
  возвращает настройки подсистемы планирования
- ***getWorkSpace()->IWorkSpace***
  предоставляет доступ к окружению
- ***getCollisionDetector()->CollisionDetector***
  возвращает анализатор столкновений
- ***getSpaceIndexFactory()->ISpaceIndexStructureFactory***
  предоставляет доступ к фабрике пространственных индексов
- ***getLocalPlannerFactory()->IProbabilisticPlannerFactory***
  предоставляет доступ к фабрике локальных планировщиков
- ***getGlobalPlannerFactory()->IGlobalPlannerFactory***
  предоставляет доступ к фабрике глобальных планировщиков
- ***OnSessionStart()->bool***
  выполняет необходимые действия при старте сессии
- ***OnSessionStop()***
  выполняет необходимые действия при завершении сессии

Целевому приложению при этом доступны следующие методы, определяемые **IGlobalPlannerSession**:

- ***Start()->bool***
  начинает сессию
- ***Stop()***
  завершает сессию
- ***getSessionID()->uid_t***
  возвращает уникальный идентификатор сессии
- ***getPathPlannerQueryInterface(uid_t*** *objectID* ***)-> PathPlannerQueryInterfaceHandler***
  предоставляет интерфейс поиска пути для заданного объекта
- ***processRequest(IGlobalPlannerRequest*** *request* ***)***
  уведомляет о событиях в динамическом окружении

Проследим последовательность вызовов методов в ходе работы подсистемы глобального планирования. Запуск подсистемы инициируется методом ***Start()***, который выполняет верификацию настроек с последующим созданием планировщика типа ***IGlobalPlanner***. Данная операция выполняется путем обращения к соответствующей фабрике, доступной в результате вызова метода сессии ***getGlobalPlannerFactory()***.

При конструировании глобальный планировщик создает диспетчер сообщений, инициирует создание и регистрацию обрабочиков сообщений. В самом планировщике разворачивается кэш геометрических моделей объектов окружения ***CollisionSpaceCache*** и кэш обработчиков запросов поиска пути ***ObjectQueryInterfaceCache***.

На следующем шаге вызывается метод ***fullUpdate()*** диспетчера сообщений, который выполняет принудительное обновление всех насчитываемых вспомогательных данных, после чего инициируется событие ***OnSessionStart()***, уведомляющее приложение об успешном запуске сессии.

Уведомление подсистемы об изменениях в окружении осуществляется путем вызова метода ***processRequest(IGlobalPlannerRequest*** *request* ***)***, принимающее в качестве параметра соответствующее типизированное сообщение. Сообщения помещаются в очередь диспетчера и управление возвращается основному потоку программного приложения. Обработка сообщений происходит в фоновом режиме в соответствии с порядком их поступления. Диспетчер берет из очереди первое поступившее сообщение и уведомляет о нем всех зарегистрированных обработчиков с учетом зависимостей. Для каждого обработчика вызывается метод ***smartUpdate(IPathPlannerRequest*** *request* ***)***, после чего сообщение уничтожается.

Для исполнения запросов планирования движения предназначен класс ***GlobalPlannerObjectQueryInterface***, внешний интерфейс которого представлен единственным методом ***findPath(IState*** *initialState****, IState*** *goalState****, IPath*** *path* ***)-> PlanningResult***. Данный класс агрегирует конфигурационное пространство типа ***IStateSpace*** и локальный планировщик типа ***IProbabilisticPlanner*** для решения соответствующих задач планирования с приписанным объектом окружения. Инстанциирование класса осуществляется в результате вызова метода сессии ***getPathPlannerQueryInterface(uid_t*** *objectID* ***)*** с указанным идентификатором объекта. Инстанциирование влечет за собой выполнение следующих действий. Во-первых, осуществляется поиск объекта по уникальному идентификатору посредством метода ***IWorkSpace::findObject()***. Для найденного объекта вызывается метод ***IWorkSpaceObject::createStateSpace()*** и конструируется экземпляр конфигурационного пространства. Далее по заданной ассоциации на геометрическую модель окружения, находящуюся в кэше глобального планировщика, формируются геометрические ограничения класса ***GeometricConstraint***. Далее, путем вызова метода сессии

getLocalPlannerFactory() создается локальный планировщик, ассоциированный с конфигурационным пространством объекта.

Построение пути осуществляется следующим образом. По окончании обработки всех сообщений очередь диспетчера блокируется. Далее строятся ограничивающие параллелепипеды геометрической модели объекта в начальном и конечном положениях, которые передаются глобальному планировщику для поиска предварительного маршрута.

Верификация и коррекция маршрута осуществляется локальным планировщиком, который выполняет построение результирующего бесконфликтного пути в конфигурационном пространстве объекта. Предварительно отобранный маршрут позволяет ограничить область сэмплирования и генерировать конфигурации с распределением, обеспечивающим более высокую плотность в труднопреодолимых областях окружения. Для этих целей генератор гауссова распределения применяется в окрестностях ключевых точек маршрута. Радиус окрестности и количество испытаний определяются, исходя из оценок расстояния до препятствий.

7.2 Построение маршрутной сети

Упомянутую выше вычислительную стратегию глобального планирования реализует класс ***TopologicalPlanner***, являющийся наследником класса ***IGlobalPlanner*** и использующийся по-умолчанию при отсутствии других альтернативных реализаций. Стратегия основана на извлечении пространственной, метрической и топологической информации из геометрического представления окружения, построении маршрутной сети в нем и согласованном использовании сети при поиске путей [3,13,14].

Реализация стратегии подразумевает формирование структуры пространственной заполненности всего объема окружения в виде октодерева с приписанными октантам статусами наполненности. Данная структура может рассматриваться в качестве упрощенного геометрического представления окружения. Октодерево дополняется метрической информацией путем приписывания свободным октантам значений расстояний от их центров до препятствий. Метрическая информация используется для кластеризации свободных октантов в виде связных областей пространства. Последние идентифицируются как "зоны" и "переходы" таким образом, что зоны занимают основные свободные области и соединяются друг с другом переходами. В результате восстанавливается топология свободных областей и формируется единая маршрутная сеть окружения в виде графа, вершины которого соответствуют зонам, а ребра — переходам между ними. Для принятия решений о наиболее перспективных маршрутах элементы сети снабжаются информацией о координатах центров выделенных областей, расстоянии до ближайшего препятствия и длинах переходов между смежными зонами.

Для программной реализации вычислительной стратегии используется два вспомогательных класса: ***SpaceOcTree***, предназначенный для формирования

структуры пространственной заполненности, и ***SpaceTopology***, используемый для представления описанной специализированной сети.

Класс ***SpaceOcTree*** поддерживает представление октодерева заполненности и инкрементально обновляет его при изменениях, связанных с добавлением, удалением и модификацией объектов окружения. Одновременно пересчитываемые значения расстояний до препятствий позволяют выделить свободные области путем объединения смежных свободных октантов и идентификации их в виде зон и переходов, представимых классами ***Space*** и ***Gate*** соответственно. Маршруты, соединяющие центры инцидентных зон и переходов, выделены в отдельный класс ***Route***.

Зоны, переходы и маршруты агрегируются классом ***SpaceTopology***, который обеспечивает их поддержку в дополнение к графовому представлению маршрутной сети класса ***TopologicalGraph***. Последний наследует рассмотренный выше шаблон ***TRoadmap*** с надлежащей специализацией типов вершин и ребер как классов зон, переходов и маршрутов.

Для согласованного обновления элементов маршрутной сети при изменениях в окружении используются классы обработчиков сообщений ***SpaceOcTreeCache*** и ***SpaceTopologyCache***. Данные классы обеспечивают эффективное инкрементальное обновление соответствующих производных данных в результате обработки типизированных сообщений.

Заключение

Таким образом, рассмотрены принципы организации и функционирования разработанной инструментальной среды для программной реализации моделей, методов и приложений теории планирования движения. Среда предоставляет развитый набор готовых к использованию программных компонентов для автоматического построения бесконфликтных траекторий для робота, перемещаемого в статическом и динамическом трехмерном окружении.

Организация среды в виде объектно-ориентированного каркаса обеспечивает развитие, адаптацию и гибкое конфигурирование разработанных программных компонентов в составе целевых приложений. Благодаря выделенным интерфейсам разного уровня и предусмотренным точкам расширения среда допускает интеграцию со сторонними прикладными системами.

Ожидается, что разработанная инструментальная среда, а также связанный с ней метод построения целевых приложений глобального планирования движения позволят существенно сократить сроки и затраты. В дальнейшем планируется апробировать среду в ходе развития системы визуального моделирования и планирования индустриальных проектов с функциями пространственно-временной верификации.

Список литературы

[1]. LaValle S.M. Planning Algorithms. Cambridge University Press, 2006.

[2]. Казаков К.А., Семенов В.А. Обзор современных методов планирования движения. Труды ИСП РАН том 28, 2016, выпуск 4. стр. 241–292. 10.15514/ISPRAS-2016-28(4)-14

[3]. Semenov V.A., Kazakov K.A., Zolotov V.A. Global path planning in 4D environments using topological mapping. eWork Ebus. Archit. Eng. Constr. 2012. pp. 263–269.

[4]. Semenov V.A., Kazakov K.A., Zolotov V.A. Advanced spatio-temporal validation of construction schedules. ICCCBE. 2012.

[5]. Dubins L.E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. 1957.

[6]. Reeds J.A., Shepp L.A. Optimal paths for a car that goes both forwards and backwards. Pacific J. Math. 1990. Vol. 145, № 2. pp. 367–393.

[7]. Chitsaz H., LaValle S.M., Balkcom D.J., Mason M.T. Minimum wheel-rotation paths for differential-drive mobile robots. Proc. - IEEE Int. Conf. Robot. Autom. 2006. Vol. 2006, № May. pp. 1616–1623.

[8]. Koenig S., Likhachev M. Fast Replanning for Navigation in Unknown Terrain Technology. 2002. Vol. XX, № May. pp. 968–975.

[9]. Naderi K., Rajamaki J., Hamalainen P. RT-RRT*: A Real-Time Path Planning Algorithm Based On RRT*. Proc. 8th ACM SIGGRAPH Conf. Motion Games - SA ’15. 2015. pp. 113–118.

[10]. Gipson I., Gupta K., Greenspan M. MPK: An open extensible motion planning kernel. J. Robot. Syst. 2001. Vol. 18, № 8. pp. 433–443.

[11]. Diankov R. Automated Construction of Robotic Manipulation Programs Architecture. 2010. Vol. Ph.D. pp. 1–263.

[12]. Sucan I., Moll M., Kavraki L.E. The Open Motion Planning Library IEEE Robot. Autom. Mag. 2012. Vol. 19, № 4. pp. 72–82.

[13]. Казаков К.А., Морозов С.В., Семенов В.А., Тарлапан О.А. Применение топологических схем для глобального планирования движения в сложном динамическом окружении. ITSE, 2016.

[14]. Semenov V.A. et al. Global path planning in complex environments using metric and topological schemes. Proc. CIB W78-W102. 2011. pp. 26–28.

[15]. Mamou K., Ghorbel F. A simple and efficient approach for 3D mesh approximate convex decomposition. 16th IEEE Int. Conf. Image Process. 2009. pp. 3501–3504.

[16]. Золотов В.А., Семенов В.А. Перспективные схемы пространственно-временной индексации для визуального моделирования масштабных индустриальных проектов. Труды ИСП РАН том 26, вып. 2, стр. 175–196. 10.15514/ISPRAS-2014-26(2)-8

[17]. Bergen G. Van Den. Efficient Collision Detection of Complex Deformable Models using AABB Trees. J. Graph. Tools. 1997. Vol. 2. pp. 1–13.

[18]. Gottschalk S., Lin M.C., Manocha D. OBB Tree: A Hierarchical Structure for Rapid Interference Detection. Proc. SIGGRAPH 96. 1996. № 8920219. pp. 171–180.

[19]. Li H., Wu Y., Wiu Y. An improved dynamic-octree-based judging method of real-time node in moving geometry. Proceedings of the 2013 International Conference on Intelligent Control and Information Processing, ICICIP 2013. 2013. pp. 333–337.

[20]. Tracy D.J., Buss S.R., Woods B.M. Efficient large-scale sweep and prune methods with AABB insertion and removal. Proc. - IEEE Virtual Real. 2009. pp. 191–198.

[21]. van den Bergen G. Proximity queries and penetration depth computation on 3d game objects. Game Dev. Conf. 2001.

[22]. Gilbert E.G., Johnson D.W., S.S. K. A Fast Procedure for Computing Distance Between Complex Objects in Three-Dimensional Space. 1988.
[23]. Russell S.J., Norvig P. Artificial Intelligence: A Modern Approach. Neurocomputing. 1995. Vol. 9, pp. 215-218.
[24]. Karaman S., Frazzoli E. Sampling-based algorithms for optimal motion planning. Int. J. Robot., vol. 30, № 7, 2011, pp. 846–894.
[25]. Kuffner J.J., LaValle S.M. RRT-connect: An efficient approach to single-query path planning. Proc. IEEE Int. Conf. Robot. Autom. ICRA '00. 2000. Vol. 2, № Icra. pp. 995--1001 vol.2.
[26]. Jaillet L., Yershova A. Lavalle S.M. Simeon T. Adaptive tuning of the sampling domain for dynamic-domain RRTs. 2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS. 2005. pp. 4086–4091.
[27]. Ferre E., Laumond J.-P. An iterative diffusion algorithm for part disassembly. IEEE Int. Conf. Robot. Autom. 2004. Proceedings. ICRA '04. 2004. Vol.3, pp. 3149–3154.
[28]. Geraerts R., Overmars M.H. Creating High-quality Paths for Motion Planning. Int. J. Rob. Res. 2007. Vol. 26, № 8. P. 845–863.
[29]. Jaillet L., Cortes J., Simeon T. Sampling-Based Path Planning on Costmaps Configuration-space. IEEE Trans. Robot. 2010. Vol. 26, № 4. P. 635–646.
[30]. Koenig S., Likhachev M., Furcy D. Lifelong Planning A*. Artif. Intell. 2004. Vol. 155, № 1–2. pp. 93–146.
[31]. Koenig S., Likhachev M. D* Lite. Proc. Eighteenth Natl. Conf. Artif. Intell. 2002. pp. 476–483.
[32]. Kuffner J.J. Effective sampling and distance metrics for 3D rigid body path planning. Proc. - IEEE Int. Conf. Robot. Autom. 2004. Vol. 4, pp. 3993--3998.
[33]. Amato N.M., Bayazit O.B., Dale L.K., Jones C., Vallejo D. OBPRM: An Obstacle-Based PRM for 3D Workspaces. Proc. Third Work. Algorithmic Found. Robot. Algorithmic Perspect. Algorithmic Perspect. 1998. pp. 155–168.
[34]. Yeh H.Y., Thomas S., Eppstein D., Amato N.M. UOBPRM: A uniformly distributed obstacle-based PRM. IEEE Int. Conf. Intell. Robot. Syst. 2012. pp. 2655–2662.
[35]. Hsu D., Jiang T., Reif J., Sun Z. The bridge test for sampling narrow passages with probabilistic roadmap planners. IEEE Int. Conf. Robot. Autom. 2003. Proceedings. ICRA '03. 2003. Vol. 3. pp. 4420–4426.
[36]. Nissoux C., Simeon T., Laumond J.-P. Visibility based probabilistic roadmaps. 1999 IEEE/RSJ Int. Conf. Intell. Robot. Syst. 1999. IROS '99. Proc. 1999. Vol. 3. pp. 1316–1321 vol.3.
[37]. Lien J.-M., Thomas S., Amato N.M. A general framework for sampling on the medial axis of the free space 2003 IEEE Int. Conf. Robot. Autom. (Cat. No.03CH37422). 2003. Vol. 3. pp. 4439–4444.
[38]. Berenson D., Srinivasa S., Ferguson D., Kuffner J.J. Manipulation Planning on Constraint Manifolds. Robotics and Automation, 2009. ICRA'09. 2009.
[39]. Berenson D., Srinivasa S., Kuffner J.J. Task Space Regions: A framework for pose-constrained manipulation planning. Int. J. Rob. Res. 2011. Vol. 30, № 12. pp. 1435–1460.
[40]. Redon S., Kheddar A., Coquillart S. Fast continuous collision detection between rigid bodies. Comput. Graph. Forum. 2002. Vol. 21, № 3. pp. 279–287.
[41]. Redon S., Lin M.C., Manocha D., Kim Y.J. Fast Continuous Collision Detection for Articulated Models. J. Comput. Inf. Sci. Eng. 2005. Vol. 5, № 2. pp. 126.

[42]. Devaurs D., Simeon T., Cortes J. Optimal Path Planning in Complex Cost Spaces with Sampling-Based Algorithms. IEEE Trans. Autom. Sci. Eng. 2016. Vol. 13, № 2. pp. 415–424.
[43]. Yershova A., LaValle S.M. Improving Motion Planning Alorithms by Efficient Nearest-Neighbor Searching. IEEE Trans. Robot. 2006. pp. 1–8.
[44]. Gipson B., Moll M., Kavraki L.E. Resolution Independent Density Estimation for motion planning in high-dimensional spaces. Proc. IEEE Int. Conf. Robot. Autom. 2013. pp. 2437–2443.
[45]. Fredriksson K. Geometric Near-neighbor Access Tree (GNAT) revisited. 2016.
[46]. Yianilos P.N. Data structures and algorithms for nearest neighbor search in general metric spaces. Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms. 1993. pp. 311–321.
[47]. Sanchez G., Latombe J.C. A single-query bi-directional probabilistic roadmap planner with lazy collision checking Robotics Research. 2003. pp. 403–417.
[48]. Kavraki L.E., Svestka P., Latombe J.C., Overmars M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Autom. 1996. Vol. 12, № 4. pp. 566–580.
[49]. Kazakov K.A., Semenov V.A., Zolotov V.A. Topological Mapping Complex 3D Environments Using Occupancy Octrees. 21st Int. Conf. Comput. Graph. Vision, Sept. 26-30, 2011, Moscow, Russ., 2011, pp. 111–114.

Object-oriented framework for motion planning in complex dynamic environments

[1] *K.A. Kazakov <kazakov@ispras.ru>*
[1,2] *V.A. Semenov <sem@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*

Abstract. In this paper, we discuss principles of the organization and functioning of the software framework intended for development of models, methods and applications of motion planning theory. Developed within an object-oriented paradigm the framework includes a wide variety of ready-to-use components that provide the functionality required for automatic search for collision-free trajectories for robots moving in both static and dynamic complex 3D environments. The proposed software design provides extensibility, adaptation and flexible configuration of the developed program components as a part of target applications. The developed architecture provides ability to integrate with third-party systems via interfaces of different level and extension points.

Keywords: motion planning; path planning; collision detection; software engineering; object-oriented programming.

DOI: 10.15514/ISPRAS-2017-29(5)-11

For citation: Kazakov K.A., Semenov V.A. Object-oriented framework for motion planning in complex dynamic environments. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017. pp. 185-238 (in Russian). DOI: 10.15514/ISPRAS-2017-29(5)-11

References

[1]. LaValle S.M. Planning Algorithms. Cambridge University Press, 2006.
[2]. Kazakov K.A., Semenov V.A. An overview of modern methods for motion planning. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 4, 2016, pp. 241-292 (in Russian). DOI: 10.15514/ISPRAS-2016-28(4)-14.
[3]. Semenov V.A., Kazakov K.A., Zolotov V.A. Global path planning in 4D environments using topological mapping. eWork Ebus. Archit. Eng. Constr. 2012. pp. 263–269.
[4]. Semenov V.A., Kazakov K.A., Zolotov V.A. Advanced spatio-temporal validation of construction schedules. ICCCBE. 2012.
[5]. Dubins L.E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. 1957.
[6]. Reeds J.A., Shepp L.A. Optimal paths for a car that goes both forwards and backwards. Pacific J. Math. 1990. Vol. 145, № 2. pp. 367–393.
[7]. Chitsaz H., LaValle S.M., Balkcom D.J., Mason M.T. Minimum wheel-rotation paths for differential-drive mobile robots. Proc. - IEEE Int. Conf. Robot. Autom. 2006. Vol. 2006, № May. pp. 1616–1623.
[8]. Koenig S., Likhachev M. Fast Replanning for Navigation in Unknown Terrain Technology. 2002. Vol. XX, № May. pp. 968–975.
[9]. Naderi K., Rajamaki J., Hamalainen P. RT-RRT*: A Real-Time Path Planning Algorithm Based On RRT*. Proc. 8th ACM SIGGRAPH Conf. Motion Games - SA ’15. 2015. pp. 113–118.
[10]. Gipson I., Gupta K., Greenspan M. MPK: An open extensible motion planning kernel. J. Robot. Syst. 2001. Vol. 18, № 8. pp. 433–443.
[11]. Diankov R. Automated Construction of Robotic Manipulation Programs Architecture. 2010. Vol. Ph.D. pp. 1–263.
[12]. Sucan I., Moll M., Kavraki L.E. The Open Motion Planning Library IEEE Robot. Autom. Mag. 2012. Vol. 19, № 4. pp. 72–82.
[13]. Kazakov K.A., Morozov S.V., Semenov V.A., Tarlapan O.A. Application of topological schemes for global movement planning in a complex dynamic environment. ITSE, 2016 (in Russian).
[14]. Semenov V.A. et al. Global path planning in complex environments using metric and topological schemes. Proc. CIB W78-W102. 2011. pp. 26–28.
[15]. Mamou K., Ghorbel F. A simple and efficient approach for 3D mesh approximate convex decomposition. 16th IEEE Int. Conf. Image Process. 2009. pp. 3501–3504.
[16]. Zolotov V.A., Semenov V.A. [Effectie spatio-temporal indexing methods for visual modeling of large industrial projects]. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 2, 2014, pp. 175-196 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-9.
[17]. Bergen G. Van Den. Efficient Collision Detection of Complex Deformable Models using AABB Trees. J. Graph. Tools. 1997. Vol. 2. pp. 1–13.
[18]. Gottschalk S., Lin M.C., Manocha D. OBB Tree: A Hierarchical Structure for Rapid Interference Detection. Proc. SIGGRAPH 96. 1996. № 8920219. pp. 171–180.

[19]. Li H., Wu Y., Wiu Y. An improved dynamic-octree-based judging method of real-time node in moving geometry. Proceedings of the 2013 International Conference on Intelligent Control and Information Processing, ICICIP 2013. 2013. pp. 333–337.
[20]. Tracy D.J., Buss S.R., Woods B.M. Efficient large-scale sweep and prune methods with AABB insertion and removal. Proc. - IEEE Virtual Real. 2009. pp. 191–198.
[21]. van den Bergen G. Proximity queries and penetration depth computation on 3d game objects. Game Dev. Conf. 2001.
[22]. Gilbert E.G., Johnson D.W., S.S. K. A Fast Procedure for Computing Distance Between Complex Objects in Three-Dimensional Space. 1988.
[23]. Russell S.J., Norvig P. Artificial Intelligence: A Modern Approach. Neurocomputing. 1995. Vol. 9, pp. 215-218.
[24]. Karaman S., Frazzoli E. Sampling-based algorithms for optimal motion planning. Int. J. Robot., vol. 30, № 7, 2011, pp. 846–894.
[25]. Kuffner J.J., LaValle S.M. RRT-connect: An efficient approach to single-query path planning. Proc. IEEE Int. Conf. Robot. Autom. ICRA ’00. 2000. Vol. 2, № Icra. pp. 995--1001 vol.2.
[26]. Jaillet L., Yershova A. Lavalle S.M. Simeon T. Adaptive tuning of the sampling domain for dynamic-domain RRTs. 2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS. 2005. pp. 4086–4091.
[27]. Ferre E., Laumond J.-P. An iterative diffusion algorithm for part disassembly. IEEE Int. Conf. Robot. Autom. 2004. Proceedings. ICRA ’04. 2004. Vol.3, pp. 3149–3154.
[28]. Geraerts R., Overmars M.H. Creating High-quality Paths for Motion Planning. Int. J. Rob. Res. 2007. Vol. 26, № 8. P. 845–863.
[29]. Jaillet L., Cortes J., Simeon T. Sampling-Based Path Planning on Costmaps Configuration-space. IEEE Trans. Robot. 2010. Vol. 26, № 4. P. 635–646.
[30]. Koenig S., Likhachev M., Furcy D. Lifelong Planning A*. Artif. Intell. 2004. Vol. 155, № 1–2. pp. 93–146.
[31]. Koenig S., Likhachev M. D* Lite. Proc. Eighteenth Natl. Conf. Artif. Intell. 2002. pp. 476–483.
[32]. Kuffner J.J. Effective sampling and distance metrics for 3D rigid body path planning. Proc. - IEEE Int. Conf. Robot. Autom. 2004. Vol. 4, pp. 3993--3998.
[33]. Amato N.M., Bayazit O.B., Dale L.K., Jones C., Vallejo D. OBPRM: An Obstacle-Based PRM for 3D Workspaces. Proc. Third Work. Algorithmic Found. Robot. Algorithmic Perspect. Algorithmic Perspect. 1998. pp. 155–168.
[34]. Yeh H.Y., Thomas S., Eppstein D., Amato N.M. UOBPRM: A uniformly distributed obstacle-based PRM. IEEE Int. Conf. Intell. Robot. Syst. 2012. pp. 2655–2662.
[35]. Hsu D., Jiang T., Reif J., Sun Z. The bridge test for sampling narrow passages with probabilistic roadmap planners. IEEE Int. Conf. Robot. Autom. 2003. Proceedings. ICRA ’03. 2003. Vol. 3. pp. 4420–4426.
[36]. Nissoux C., Simeon T., Laumond J.-P. Visibility based probabilistic roadmaps. 1999 IEEE/RSJ Int. Conf. Intell. Robot. Syst. 1999. IROS ’99. Proc. 1999. Vol. 3. pp. 1316–1321 vol.3.
[37]. Lien J.-M., Thomas S., Amato N.M. A general framework for sampling on the medial axis of the free space 2003 IEEE Int. Conf. Robot. Autom. (Cat. No.03CH37422). 2003. Vol. 3. pp. 4439–4444.
[38]. Berenson D., Srinivasa S., Ferguson D., Kuffner J.J. Manipulation Planning on Constraint Manifolds. Robotics and Automation, 2009. ICRA’09. 2009.

[39]. Berenson D., Srinivasa S., Kuffner J.J. Task Space Regions: A framework for pose-constrained manipulation planning. Int. J. Rob. Res. 2011. Vol. 30, № 12. pp. 1435–1460.
[40]. Redon S., Kheddar A., Coquillart S. Fast continuous collision detection between rigid bodies. Comput. Graph. Forum. 2002. Vol. 21, № 3. pp. 279–287.
[41]. Redon S., Lin M.C., Manocha D., Kim Y.J. Fast Continuous Collision Detection for Articulated Models. J. Comput. Inf. Sci. Eng. 2005. Vol. 5, № 2. pp. 126.
[42]. Devaurs D., Simeon T., Cortes J. Optimal Path Planning in Complex Cost Spaces with Sampling-Based Algorithms. IEEE Trans. Autom. Sci. Eng. 2016. Vol. 13, № 2. pp. 415–424.
[43]. Yershova A., LaValle S.M. Improving Motion Planning Alorithms by Efficient Nearest-Neighbor Searching. IEEE Trans. Robot. 2006. pp. 1–8.
[44]. Gipson B., Moll M., Kavraki L.E. Resolution Independent Density Estimation for motion planning in high-dimensional spaces. Proc. IEEE Int. Conf. Robot. Autom. 2013. pp. 2437–2443.
[45]. Fredriksson K. Geometric Near-neighbor Access Tree (GNAT) revisited. 2016.
[46]. Yianilos P.N. Data structures and algorithms for nearest neighbor search in general metric spaces. Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms. 1993. pp. 311–321.
[47]. Sanchez G., Latombe J.C. A single-query bi-directional probabilistic roadmap planner with lazy collision checking Robotics Research. 2003. pp. 403–417.
[48]. Kavraki L.E., Svestka P., Latombe J.C., Overmars M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Autom. 1996. Vol. 12, № 4. pp. 566–580.
[49]. Kazakov K.A., Semenov V.A., Zolotov V.A. Topological Mapping Complex 3D Environments Using Occupancy Octrees. 21st Int. Conf. Comput. Graph. Vision, Sept. 26-30, 2011, Moscow, Russ., 2011, pp. 111–114.