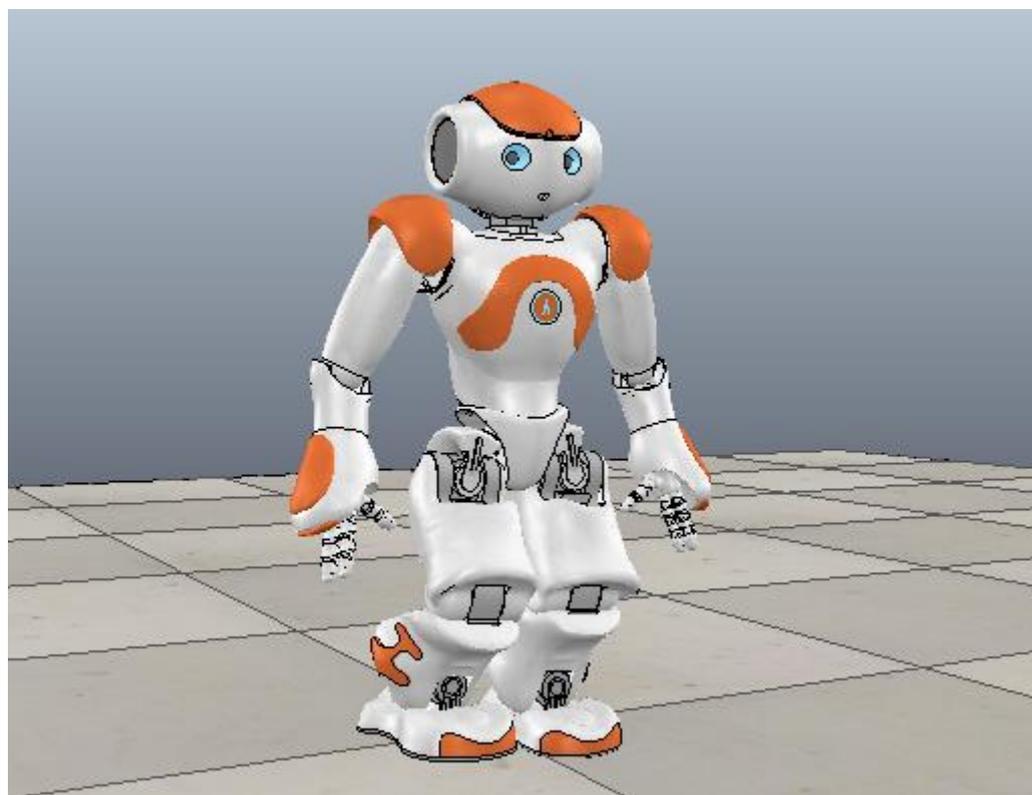


И.А. Бжихатлов
МОДЕЛИРОВАНИЕ РОБОТОТЕХНИЧЕСКИХ
СИСТЕМ В ПРОГРАММЕ V-REP



Санкт-Петербург
2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

И.А. Бжихатлов

**МОДЕЛИРОВАНИЕ РОБОТОТЕХНИЧЕСКИХ
СИСТЕМ В ПРОГРАММЕ V-REP**

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО

по направлению подготовки 15.03.06

**в качестве учебно-методического пособия для реализации основных
профессиональных образовательных программ высшего образования бакалавриата.**

 **УНИВЕРСИТЕТ ИТМО**

Санкт-Петербург

2018

Бжихатлов И.А. Моделирование робототехнических систем в программе V-REP. Учебно-Методическое пособие. – СПб: Университет ИТМО, 2018. – 59с.

Рецензенты:

Абрамчук Михаил Владимирович, к.т.н., преподаватель кафедры мехатроники Университета ИТМО.

Данное учебно-методическое пособие раскрывает вопросы моделирования робототехнических систем в программе V-REP и дает навыки моделирования процесса работы робототехнических и мехатронных комплексов с учетом законов физики. Данное пособие затрагивает в большей или меньшей степени такие области науки, как механика, управление и программирование. В пособии приведены примеры реализации наиболее часто встречающихся робототехнических систем. Методическое пособие предназначено для студентов, обучающихся на факультете систем управления и робототехники по направлению 15.03.06 Мехатроника и робототехника, уже прошедших курсы по механике, основам программирования и теории управления в технических системах.



Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

© Университет ИТМО, 2018

© Бжихатлов И.А., 2018

Оглавление

Введение	4
1. Структура и ключевые особенности V-REP	6
2. Основы работы в программе V-REP	9
2.1 Интерфейс программы	9
2.2 Обзор инструментов V-REP	14
2.3 Написание скриптов в программе V-REP	17
2.3.1 Язык Lua	17
2.3.2 Основные конструкции	18
2.4 Типы объектов программы V-REP	22
3. Лабораторный практикум	26
Лабораторная работа №1	26
Лабораторная работа №2	38
Лабораторная работа №3	52
Заключение	57
Список рекомендованной литературы	58
Приложение 1	59

Введение

В настоящее время робототехника является одним из самых востребованных направлений в развитии автоматизированных технологических систем и активно используется в таких областях, как медицина, телекоммуникация, военное и промышленное дело, а также образование.

Моделирование давно уже стало неотъемлемой частью разработки какой бы то ни было технологии или продукта. Оно позволяет сберечь время и деньги, а в некоторых случаях даже позволяет не подвергать сам продукт, людей и окружающую среду различного рода опасностям. В сфере робототехники актуальность моделирования преобладает еще в большей степени ввиду высокого уровня сложности технических систем. Будь то исследовательский проект или новый продукт для серийного производства, грамотное моделирование с использованием математического аппарата и вычислительных возможностей современных компьютеров сокращает как технологические риски в процессе создания, так и вероятность выхода робототехнической системы из строя при дальнейшей эксплуатации.

Развитие робототехники и технологий моделирования привело к появлению целого класса программного обеспечения – робототехнических симуляторов. Особенную значимость симуляторов необходимо выделить при обучении робототехнике [1]. Виртуальные симуляторы существенно отличаются от сред моделирования общего назначения. В них применяется подход, при котором в модели имеются компоненты, реализующие такие же функции, как и аналогичные компоненты в реальных роботах. Таким образом, еще при работе с виртуальной моделью студенты учатся работать с электрическими приводами, механическими элементами и системой управления, что позволяет снизить риски без вреда для процесса обучения.

Тестирование является наиболее распространенной [2], но далеко не единственной целью разработки экспериментальных платформ. Например, большинство современных промышленных роботов имеют собственные системы оффлайн – программирования, в которую входит и система экспериментального моделирования. Подобные системы выполняют все необходимые расчеты для моделирования и визуализируют процесс. На рис. 1 представлен интерфейс специализированного программного обеспечения для оффлайн – программирования и тестирования исполняемой программы промышленных роботов [3] производства компании «ABB Robotics».

Цель моделирования зависит от поставленных задач, а при создании новых робототехнических решений, главным образом – от стадии разработки. Это может быть и проверка гипотезы, и оптимизация конструкции, и тестирование программного обеспечения, реализующего новые алгоритмы обработки сенсорной информации и управления поведением, и на более поздних этапах – отладка исполняемого кода до его запуска на контроллере рабочей станции манипулятора.

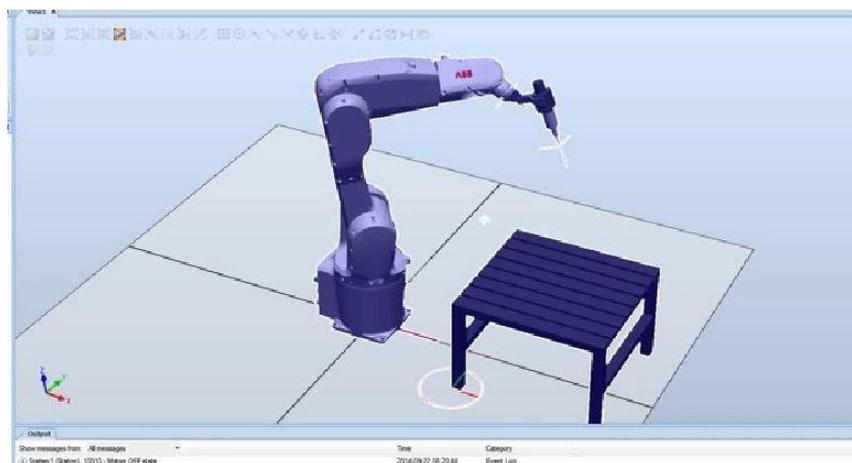


Рисунок 1 – Интерфейс программы ABB Robot Studio

Процесс разработки – нелинейный, и хорошая среда компьютерного моделирования позволяет избежать многих проблем еще на ранней стадии, позволяя тестировать как отдельные узлы робота, так и упрощенные модели, реализующие отдельные функции.

Существует ряд программных решений для робототехнических систем, которые позволяют создавать симуляции с высокой точностью. Среди них можно выделить Gazebo [4] и V-REP [5], как наиболее популярные и имеющие большой функционал. Эти платформы обладают широкими возможностями по моделированию роботов различного типа, начиная от плавающих и заканчивая летающими. Отличительной особенностью платформ также является возможность легкого масштабирования. Однако интерфейс разработчика в программе V-REP существенно лучше интерфейса в Gazebo, где интерфейс для разработки отсутствует как таковой. Программный пакет Gazebo распространяется абсолютно бесплатно, однако вся разработка в нем ведется на языке разметки xml и является крайне трудоемким процессом. Кроме того, программа V-REP регулярно получает новые обновления, появляются новые функции и возможности, в то время как Gazebo слабо поддерживается, и существенных улучшений не наблюдается за последние 3 года. Ввиду этих фактов, V-REP на сегодня является оптимальным решением для тех, кто только начинает изучать возможности моделирования процессов работы робототехнических систем и особенности создания симуляций для роботов, чтобы начать их применять на практике.

1. Структура и ключевые особенности V-REP

Платформа V-REP имеет ряд особенностей, которые предоставляют разработчику широкие возможности для создания симуляций. Как основной компонент V-REP можно выделить технологию встроенных скриптов, которые выполняют функции контроллеров в симуляциях. При этом, наличие возможности привязки отдельных скриптов к компонентам робота позволяет реализовать четкую иерархию, обеспечивая портативность и масштабируемость.

Любая симуляция в V-REP по умолчанию имеет основной скрипт, который крайне не рекомендуется менять, т.к. данный скрипт решает общие задачи обеспечения корректности данных при выполнении симуляции. Например, он вызывает разные подсистемы для моделирования кинематики и динамики механических элементов системы. Из основного скрипта также выполняется вызов дочерних скриптов каскадным способом, но эту функцию можно отключить.

Дочерние скрипты, в отличие от основного, прикрепляются к отдельным компонентам модели. Они являются неотъемлемой частью сценария симуляции и выполняются при каждой итерации моделирования, как и основной скрипт. Следует также помнить, что скрипты являются исполняемыми файлами и не требуют предварительной компиляции. Кроме того, дочерние скрипты могут быть потоковыми или непотоковыми. Разработчики V-REP рекомендуют по возможности использовать непотоковые скрипты, однако в некоторых задачах потоковые скрипты лучше решают поставленные задачи. Следует также заострить внимание на том, что симуляция – это итеративный процесс, т.е. перерасчет параметров моделируемой системы осуществляется через постоянный промежуток времени (шаг моделирования) итеративно. В V-REP по умолчанию используется 50 миллисекунд.

Потоковые скрипты – скрипты, выполнение которых будет продолжаться при следующей итерации. Такие скрипты выполняются один раз от начала до конца, но также можно отключить эту функцию, чтобы не было прерывания после первой итерации.

Непотоковые скрипты – скрипты, выполнение которых начинается с начала при каждой последующей итерации, при этом осуществляется полная передача управления симуляцией в эти скрипты. Если по каким-то причинам скрипт не завершился, и управление не передано обратно на основной скрипт, то симуляция прерывается. Такие скрипты вызываются основным скриптом 2 раза за каждый шаг симуляции: при активации и получении сенсорной информации.

Таким образом, для лучшего понимания процесса исполнения скрипта следует рассмотреть более подробно структуру не потокового скрипта, что будет сделано в главе 3.

Доступность. Распространяется платформа V-REP условно бесплатно. Для научных исследований и ведения образовательной деятельности данная программа имеет отдельные версии. В случае же использования в

коммерческих проектах необходимо приобрести лицензию. Программа V-REP является полностью кроссплатформенной (не зависимой от используемой операционной системы).

V-REP поддерживает модель взаимодействия Клиент-Сервер. Платформа V-REP выступает сервером, на котором запущена симуляция, а управление осуществляется с клиентской стороны. При такой схеме взаимодействия в качестве клиента может выступать как другое программное обеспечение, так и любое устройство с подключением к серверу (V-REP), например, клиентом может быть робот.

API функции. Предлагается широкий набор готовых API функций на языке Lua, использование которых делает процесс создания сценария симуляции быстрым и легким. Широкие возможности также предоставляют API функции для создания сценариев симуляции на других языках программирования (Remote API), а именно: C/C++, Python, Java, Matlab, Octave и Lua.

В V-REP имеется 4 разных физических движка моделирования: ODE, Bullet, Vortex и Newton. Каждый из них имеет более высокую точность моделирования в определенных задачах, но все они хорошо решают общие задачи моделирования роботов.

Встроенные модули для специализированных задач. Имеется мощный и гибкий модуль вычисления обратной и прямой кинематики роботов. Данный пакет хорошо подходит для узкоспециализированных задач при работе с манипуляторами. Встроенный модуль для быстрой проверки столкновений объектов позволяет решать ряд задач, связанных с безопасностью, максимально эффективно. Также имеется схожий модуль, который быстро и точно рассчитывает минимальное расстояние между объектами.

Имеются различные типы датчиков, в том числе наиболее часто применяемые датчики приближения (аналог ультразвуковых датчиков). Также имеются фото-видео датчики, которые анализируют видимые свойства различных компонентов симуляции.

Планирование движения. В программе V-REP планирование движения выполняется с использованием открытой библиотеки планирования движения (Open Motion Planning Library), сокращенно OMPL [6].

Запись и визуализация данных. Возможности для визуализации данных являлись одними из ключевых для разработчиков, и начиная с самой первой версии в V-REP имеется весь необходимый инструментарий для записи и визуализации любых типов данных. Более подробно данная тема раскрыта в главе 2.

Встроенный редактор сетки. Имеется редактор сетки, который позволяет вносить изменения в сетку модели, на основе которой рассчитываются физические параметры механических элементов симуляции. Также поддерживается несколько специальных режимов редактирования сетки объекта.

Импорт и экспорт данных. Загрузка и выгрузка данных из V-REP выполняется через меню: для этих задач имеется специальный инструмент в основном меню. Также программа может считывать данные через соединение с другим устройством.

Полнофункциональная иерархия моделей в сцене. Этот инструмент позволяет платформе V-REP реализовать взаимосвязи компонентов модели крайне удобно и с широкими возможностями масштабирования.

Это далеко не все, чем отличается V-REP от других программ моделирования, однако понимание этих особенностей позволит в дальнейшем работать с платформой и дополнять общую информацию более детальными и специализированными возможностями программы.

Вопросы для самоконтроля:

1. Какие языки программирования поддерживает V-REP для реализации систем управления (написания скриптов)?
2. Какое главное отличие потокового скрипта от не потокового?
3. Какие условия нужны V-REP для обмена данными с реальным роботом?
4. Какие способы ввода и вывода информации имеются в V-REP?

2. Основы работы в программе V-REP

В данном разделе содержится ознакомительная информация, главной целью которой является формирование представления о базовых элементах, с которыми необходимо взаимодействовать при создании симуляций в программе V-REP.

2.1 Интерфейс программы

Интерфейс программы V-REP разделен на несколько частей в зависимости от назначения и реализован в окне с графическим интерфейсом: графическое окно программы используется для управления всеми встроенными инструментами.

Также следует упомянуть **окно консоли**, которое можно наблюдать во время запуска приложения, но по умолчанию данное окно скрывается сразу же. При необходимости можно настроить V-REP на постоянное отображение консоли, вызвав «User settings» («Пользовательские настройки») с помощью первой кнопки в вертикальной панели инструментов. В консольном окне отображаются загружаемые плагины и их процедуры, которые нужны только при работе с плагинами. В данной учебной литературе не затрагивается работа с плагинами, однако при необходимости можно обратиться к документации по работе в V-REP на сайте разработчиков [5].

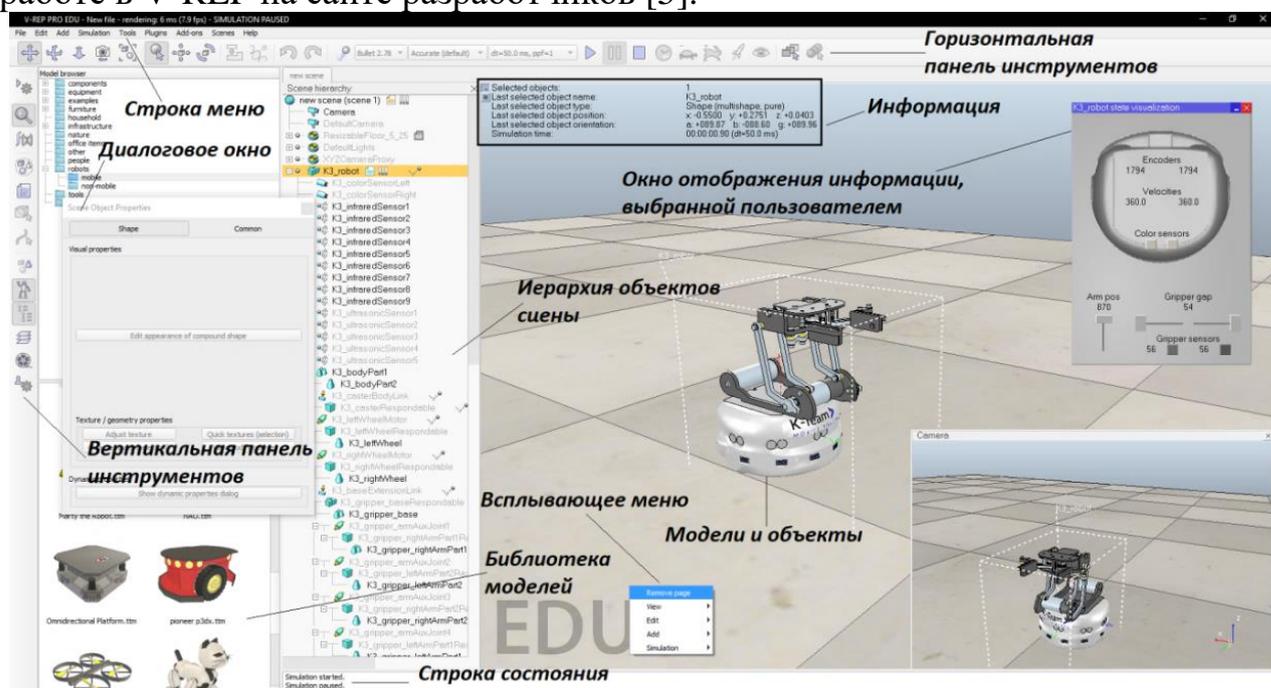


Рисунок 2.1 – Окно приложения V-REP

На рис. 2.1 приведен интерфейс программы V-REP с довольно большим набором активных инструментов, однако надо понимать, что одновременная активация всех инструментов приведет к их взаимному перекрытию.

Далее приведены основные составляющие графического интерфейса программы V-REP с кратким описанием.

Основное меню программы. Содержит различные разделы, в которых имеются инструменты для редактирования имеющихся и добавления новых

объектов в сценарий симуляции. Часть инструментов из главного меню также продублирована во всплывающем контекстном меню V-REP.

Панели инструментов. Наиболее часто используемые инструменты вынесены сюда для быстрого доступа. На рис. 2.2 и 3 приведены кнопки, расположенные на панелях инструментов с их названиями. Каждый инструмент из этих панелей может находиться в активном и пассивном режиме. При этом активация производится простым нажатием на пиктограмму инструмента. Более подробная информация по инструментам приведена в разделе 2. 2.

Библиотека моделей. По умолчанию данный инструмент находится в активном режиме, но может быть деактивирован нажатием на пиктограмму. В активном режиме данный инструмент отображает структуру папок и моделей, содержащихся в текущей папке. Модели из библиотеки можно легко добавить в сценарий симуляции путем перетаскивания в рабочее окно.

Иерархия объектов сценария. Древоподобная иерархия отображает структуру объектов и их компонентов. Их можно изменять, либо добавлять новые элементы. Иерархия объектов является ключевым элементом, который задает основные связи между объектами сценария. Также свойства любого объекта можно открыть двойным нажатием левой кнопки мыши на пиктограмму слева от названия объекта в иерархии сцены. Также можно выбрать сначала необходимый элемент с помощью одного нажатия на элемент и затем активировать инструмент «Свойства объектов» («Scene Object Properties»). Для изменения названий компонентов достаточно навести курсор на текущее название и дважды нажать на левую кнопку мыши, после чего ввести новое название и завершить нажатием кнопки ввода («Enter»). Следует учесть, что допускается использование только букв латинского алфавита. Одной из ключевых функций, реализуемых посредством иерархии объектов, является взаимосвязь компонентов системы. Для данной функции доступно 2 способа. Перетаскивая один объект на другой, можно установить отношения связи между ними (сделать одного из них «родителем» по отношению к другому). Аналогичное действие можно выполнить через всплывающее меню, для этого нужно выбрать сначала «дочерний» элемент, зажать клавишу «Shift» и затем выбрать «родительский», правой кнопкой мыши открыть всплывающее контекстное меню и выбрать «Edit» и «Make last selected object Parent».

Основная страница (основное окно). Каждый сценарий симуляции может содержать до 8 страниц, которые, в свою очередь, могут включать неограниченное количество областей отображения. Подобное разделение позволяет максимально эффективно использовать графический интерфейс для вывода различных типов данных одновременно.

Области отображения. Области отображения представляют собой области интерфейса, предназначенные для разделенного вывода данных сценария и визуализации, полученных с помощью виртуальных видеокамер, сенсоров различного типа и другой заданной пользователем информации.

Информация. Блок с информацией выводится для отдельных моделей, компонентов или сценария в зависимости от выбора пользователя и показывает различные параметры.

Глобальная система координат. Обозначение ориентации всегда присутствует в правом нижнем углу. Также у каждого объекта в V-REP имеется локальная система координат.

Окно отображения информации, выбранной пользователем – настраиваемое пользователем окно, отражающее необходимую информацию или диалог с пользователем.



Рисунок 2.2 – Горизонтальная панель инструментов V-REP

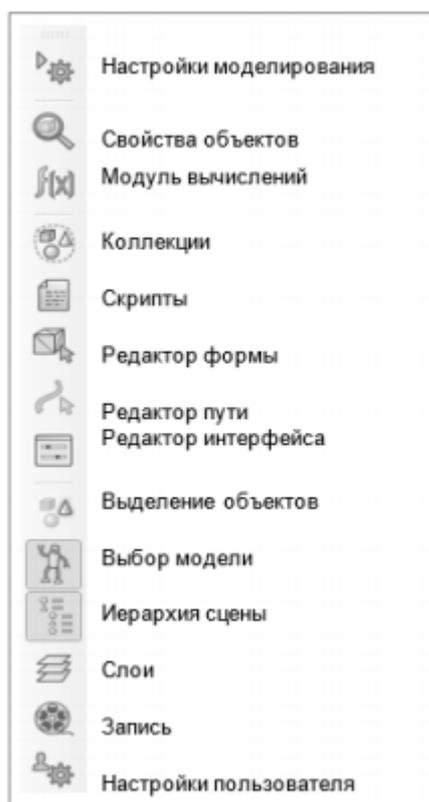


Рисунок 3 – Вертикальная панель инструментов V-REP

Всплывающее меню. Появляется при нажатии правой кнопкой мыши, имеет широкие возможности и дублирует часть инструментов и функций из панелей инструментов.

Диалоговое окно. Отображает свойства объектов с возможностью их редактирования, может каскадно расширяться в некоторых случаях. Окно открывается при активации отдельных инструментов. Параметры, отображаемые в этом окне, зависят от типа выбранного элемента и назначения инструмента.

Строка состояния. Область интерфейса используется для вывода текстовой информации во время выполнения симуляции.

Объекты пустой симуляции. При создании нового сценария симуляции в программе V-REP уже имеются несколько объектов, которые добавляются по умолчанию: источник света, пол с изменяемым размером и камеры. Конечно, их можно удалить, но почти всегда они необходимы для создания симуляции, также иногда возникает необходимость в их настройке. Для изменения размера поверхности пола достаточно выделить его в иерархии сцены и в появившемся пользовательском интерфейсе перемещать ползунок до тех пор, пока не получится требуемый результат.

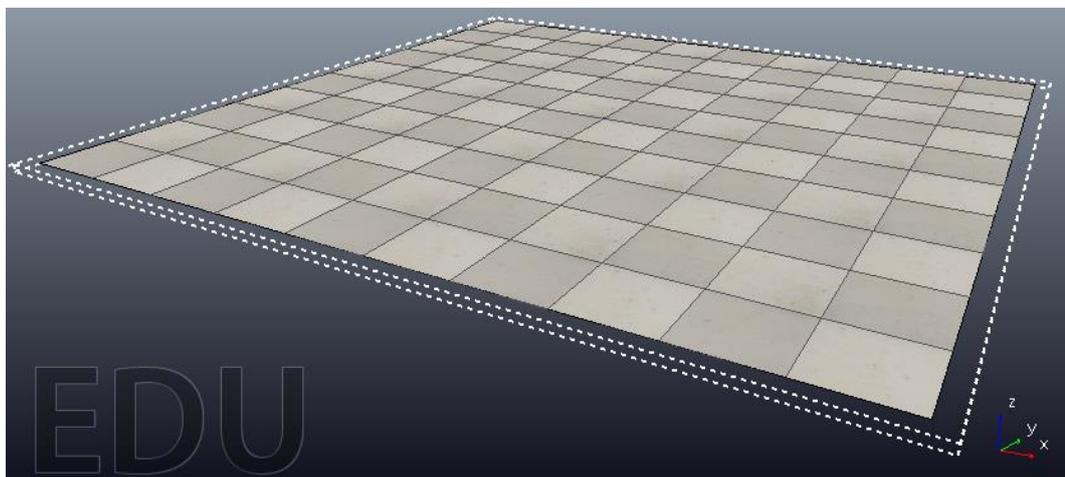


Рисунок 3.2 – Поверхность пола с изменяемым размером

Вопросы для самоконтроля:

1. Как можно активировать инструмент из панели?
2. Какие задачи выполняет иерархия объектов в V-REP?
3. Каким образом можно открыть свойства отдельных объектов для редактирования?
4. Каким образом можно задать взаимосвязь двух элементов системы, которые контактируют друг с другом?

2.2 Обзор инструментов V-REP

В данном разделе приводится описание основных инструментов, без знания и понимания которых становится невозможным создание симуляций робототехнических систем. Следует отметить, что большая часть инструментов V-REP дублируется в разных меню, но при этом их назначение и функциональность остаются абсолютно идентичными. Например, некоторые блоки инструментов из главного меню также можно найти в всплывающем меню, сделано это исключительно для повышения удобства использования.

Изменение положения, ориентации и масштабирование. Это одни из основных инструментов, необходимых для ввода начальных условий симуляции. С помощью инструмента «Object/Item Shift» (рис. 3.2, слева) можно задавать положение объектов в сцене и масштаб (уменьшать или увеличивать объект). Инструмент «Object/Item Rotation» (рис. 3.2 - справа) позволяет задавать ориентацию объекта в пространстве.



Рисунок 3.2 – Инструменты «положение» и «ориентация»

Изменение свойств объекта. Инструмент «Свойства» является не менее часто используемым при создании симуляций в программе V-REP и позволяет задавать различные свойства объектов и компонентов симуляции. После активации инструмента выводится контекстное меню, которое имеет две вкладки. В первой вкладке приводится перечень свойств, который является специализированным и зависит от типа выбранного объекта. Во второй вкладке контекстного меню инструмента приводятся общие свойства, которые у большинства объектов схожи.

Запуск и остановка симуляции. Для этих задач в горизонтальной панели имеется набор кнопок на панели инструментов (рис. 3.3), которые позволяют запускать симуляцию (показан на рис. 3.3, слева), останавливать (рис. 3.3 посередине) и завершать симуляцию (рис. 3.3, справа).



Рисунок 3.3 – Инструменты запуска, остановки и завершения симуляции

Также на рис. 3.3 слева от инструмента запуска симуляции показаны настройки решателя (библиотеки для моделирования законов физики), где можно изменить шаг моделирования, режим моделирования и используемое ядро физического движка.

Управление визуальными свойствами. В программе V-REP реализован механизм послойного разделения всех визуальных объектов. Данный

функционал является необходимым, т.к. в программе нужно использовать совмещение компонентов различных типов, и в таких случаях становится невозможной дальнейшая работа с объектами. Например, наличие одинаковых компонентов, которые имеют одинаковое положение, но моделируют разные свойства, приводит к тому, что они становятся визуально неразличимыми. Поэтому необходимо пользоваться инструментом «Слои». Рекомендованный принцип разнесения на слои: компоненты одного типа должны быть на отдельном слое (например, все шарниры). Всего в V-REP доступно 10 слоев, в свойствах каждой компоненты имеется возможность выбора слоя, на который будет вынесен элемент. После того, как элементы разнесены по слоям, можно активировать инструмент «layers» («слои») из левой панели инструментов и переключаться между слоями.

Инструмент управления скриптами. Добавление и удаление скриптов, а также изменение привязки к отдельным компонентам легко выполняется через иерархию сценария симуляции. Однако V-REP имеет и дублирующий инструмент «Scripts», который тоже позволяет выполнять аналогичные операции и зачастую делает это быстрее. Но для начинающих работу в V-REP этот инструмент может представлять трудности, т.к. принцип его работы не является столь интуитивно понятным, как аналогичная операция через иерархии компонентов.

Редактор форм. Данный инструмент предназначен для редактирования сетки механических элементов моделируемой системы. Установка более мелкой сетки позволяет получить более высокую точность симуляции, но если измельчить сетку во всех компонентах, то это приводит к увеличению потребляемой вычислительной мощности. Поэтому необходимо задавать размеры сетки переменной концентрации посредством редактирования сетки в ручном режиме. Чаще всего этот инструмент необходим при создании симуляции на основе трехмерных моделей, которые были импортированы из CAD-системы. В редакторе форм доступно 3 режима работы: режим редактирования треугольников, режим редактирования вершин и режим редактирования кромки.

Режим редактирования треугольников. В этом режиме отдельные треугольники, составляющие форму, могут быть выделены, после чего их можно удалять, закрывать оставшиеся пустоты и разбивать на более мелкие треугольники. С помощью кнопки «Subdivide largest triangles» можно разбить все треугольники на более мелкие, т.е. каждое нажатие уменьшает размер треугольников в 2 раза.

Режим редактирования вершин. В этом режиме отдельные вершины, составляющие фигуру, могут быть выделены, а затем удалены. Также имеется возможность создания новых треугольников, для этого необходимо выбрать 3 или более вершин и нажать «Insert triangles».

Режим редактирования кромки. В этом режиме доступны функции, аналогичные описанным выше. Отличие состоит лишь в том, что для выделения и редактирования доступны отдельные кромки формы.

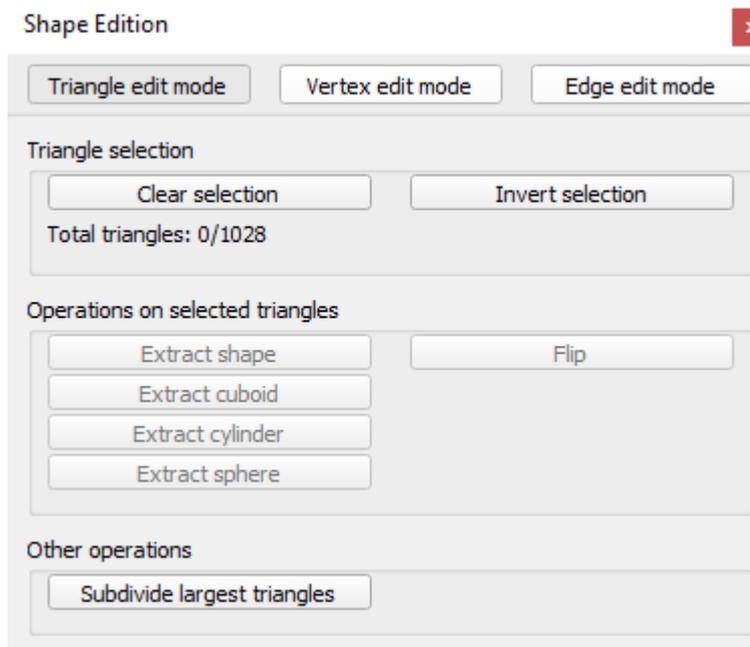


Рисунок 3.4 – Инструмент редактирования форм

Следует отметить, что составные формы нельзя редактировать напрямую с помощью данного инструмента, их необходимо заранее разгруппировать и вносить изменения по отдельности.

Вопросы для самоконтроля:

1. Можно ли менять шаг моделирования в V-REP при создании сценария симуляции?
2. Как можно открыть свойства объектов?
3. Каким образом можно переместить элементы сцены в V-REP?
4. Какие визуальные свойства влияют на поведение объекта при выполнении симуляции?

2.3 Написание скриптов в программе V-REP

Как было отмечено выше, платформа V-REP поддерживает работу с разными языками программирования, в том числе и с наиболее распространенными, такими как C++ и Python. Более подробное знакомство с возможностями написания скриптов будем выполнять с использованием языка Lua, так как этот язык является встроенным языком V-REP, и начать написание скриптов можно сразу после запуска программы.

2.3.1 Язык Lua

Отличительной особенностью Lua является простота: во многом синтаксис схож с популярным языком C, что значительно упрощает работу для тех, кто уже знаком с языком. Скрипты в V-REP открывают большие возможности для реализации управления как отдельными объектами сцены, так и платформой.

Для начала необходимо вспомнить, что скрипты в V-REP бывают двух типов: **поточковые** и **непоточковые** (более подробно смотрите в главе 3). Рассмотрим структуру не потокового скрипта, как рекомендованного разработчиками и наиболее часто используемого. Такой скрипт всегда состоит из нескольких блоков (см. рис. 4.1).

```
1  -- DO NOT WRITE CODE OUTSIDE OF THE if-then-end SECTIONS BELOW!! (unle
2
3  if (sim_call_type==sim_childscriptcall_initialization) then
4
5      -- Put some initialization code here
6
7
8  end
9
10
11 if (sim_call_type==sim_childscriptcall_actuation) then
12
13     -- Put your main ACTUATION code here
14
15     -- For example:
16     --
17     -- local position=simGetObjectPosition(handle,-1)
18     -- position[1]=position[1]+0.001
19     -- simSetObjectPosition(handle,-1,position)
20
21 end
22
23
24 if (sim_call_type==sim_childscriptcall_sensing) then
25
26     -- Put your main SENSING code here
27
28 end
29
30
31 if (sim_call_type==sim_childscriptcall_cleanup) then
32
33     -- Put some restoration code here
34
35 end
```

1. Блок инициализации

2. Блок управляющего кода объектов

3. Блок управляющего кода сенсеров

4. Блок завершения симуляции

Рисунок 4.1 – Окно редактирования потокового скрипта в программе V-REP

Блок инициализации. Содержимое данного блока выполняется только один раз при запуске симуляции. В данном блоке производятся такие операции, как объявление необходимых переменных и присвоение им исходных значений. Также в этой части скрипта задаются обработчики для управления объектами сцены.

Блок активации. Содержимое данного блока выполняется итеративно через равные промежутки времени (шаг моделирования). Часть скрипта, написанная

в этом блоке, будет повторяться вплоть до остановки симуляции или возникновения критической ошибки (при которой тоже симуляция будет остановлена). Здесь описывается основной алгоритм управления.

Блок управляющего кода сенсоров. Содержимое данного блока выполняется столько же раз, сколько и блок активации. Однако основной скрипт V-REP обращается к этому блоку только после завершения выполнения скрипта из блока активации. Данный блок разработан для получения данных из сенсоров.

Блок завершения симуляции. Данный блок позволяет выборочно стереть данные, полученные в ходе симуляции. Скрипт из этого блока начинает выполняться один раз перед завершением симуляции или удалением скрипта. Этот блок в большинстве случаев остается пустым.

Скрипты не потокового типа имеют более простую структуру. Весь скрипт может указываться в файле без разделения на блоки, однако ввиду итеративности моделируемых процессов почти всегда необходимо наличие основного цикла. На рис. 4.2 представлена классическая структура не потокового скрипта в программе V-REP, однако основной цикл там может и отсутствовать при решении задач определенного класса.

```
1 -- Put some initialization code here
2
3 1. Блок инициализации
4 -- Put your main loop here, e.g.:
5 while simGetSimulationState() ~= sim_simulation_advancing_abouttostop do
6
7 2. Основной цикл управляющего кода
8
9 end
10
11
12 -- Put some clean-up code here
13 3. Блок завершения скрипта (очистка)
14
```

Рисунок 4.2 – Окно редактирования не потокового скрипта в программе V-REP

2.3.2 Основные конструкции при написании скриптов

Комментарии на языке Lua должны начинаться с двойного дефиса (рис. 5, строка 4). Рекомендуется не удалять фрагменты скрипта во время разработки алгоритмов управления, а обозначать их как комментарий, чтобы потом не пришлось заново писать, если они будут нужны.

Как правило, ни один скрипт не обходится без использования переменных. В Lua допускается объявление новых переменных в любой момент времени, также при объявлении можно задать их начальное значение (например, рис. 5, строка 9). Также нет необходимости указывать тип переменной. Lua определит его в зависимости от значения, которое присваивается переменной в первый раз. Переменные с числовым значением воспринимаются как число с плавающей точкой. Также допускается использование переменных с символьными значениями и переменных логического типа (возможные значения: «истина» или «ложь»). Имеется и функция уничтожения (освобождения) переменных. Все переменные в Lua по умолчанию являются глобальными, то есть доступны из любой области.

```

3 if (sim_call_type==sim_childscriptcall_initialization) then
4   -- Put some initialization code here
5   base=simGetObjectHandle('verh_dyn')
6   jointhandle=simGetObjectHandle('joint_1')
7   jointcurrentpos=simGetJointPosition(jointhandle)
8   simAddStatusBarMessage('position='..jointcurrentpos)
9   nextposition=1

```

Рисунок 5 – Пример скрипта инициализации

Ветвление и циклы реализованы в Lua также, как и в большинстве С-подобных языков программирования с некоторыми оговорками.

Условный оператор «If» (Если) требует обязательного использования конструкции «then/end». При этом после ключевого слова «If» необходимо указать логическое выражение или переменную. После логического выражения следует ключевое слово «then», за которым начинается «тело» условия – часть скрипта, которая будет выполняться при истинности указанного в скрипте условия. В условном операторе имеется необязательная составляющая – дополнительное условие «иначе» (elseif), «тело» этого условия – часть скрипта, которая выполняется только если указанное в скрипте условие окажется ложным. На рис. 6 приведен пример использования условного оператора.

```

11 k=10
12 if k ~= 10 and k<0 then
13   print('negative')
14   simAddStatusBarMessage('negative')
15 else
16   print('positive')
17   simAddStatusBarMessage('positive and k='..k)
18 end

```

Рисунок 6 – Пример использования условного оператора и функций вывода информации в консольное окно и окно состояния

Условный оператор также допускает комбинирование условий с помощью операторов «and» (логическое «и») и «or» (логическое «или»). Строки 13 и 16 на рис. 6 выводят ключевые слова базовыми функциями Lua в консольное окно, которое по умолчанию скрыто. Логичнее и удобнее выводить данные не в консольное окно, а в строку состояния V-REP, для этого необходимо воспользоваться регулярной функцией V-REP (пример приведен на рис. 6, строки 14 и 17). Также на рис. 6 в строке 17 выводится не только ключевое слово, но и значение переменной.

Как и в случае с условными операторами, система управления редко обходится без использования хотя бы одного цикла. Циклы в Lua задаются с помощью ключевых слов, обозначающих тип цикла совместно с ключевыми словами «do» и «end». При этом порядок такой: сначала указывается тип, затем условие выполнения цикла, далее слово «do», далее следует «тело цикла» - фрагмент скрипта, который будет выполняться циклично, и заканчивается цикл словом «end». Наиболее распространенными являются циклы типа «while» и «for». На

рис. 7 приведен пример самого простого цикла `while`, который увеличивает на единицу значение переменной «`k`», пока условие «`k < 50`» не станет ложным.

```
10 k=1
11 while k < 50 do
12   k = k + 1
13 end
```

Рисунок 7 – Пример использования цикла «`while`»

Циклы в Lua, как и условное ветвление, должны завершаться ключевым словом «`end`». Особое внимание следует уделить условию, которое используется в цикле. Особенно это важно, когда используется значение переменной вместо логического условия. У неопределенных переменных значение по умолчанию равно «`nil`». При этом в условии цикла только переменные со значением «`nil`» и «`false`» (логический тип переменной «ложь») возвращают `false`, в то время как значение переменной «`0`» и `' '` возвращают `true`.

Второй тип циклов, который наиболее часто применяется, – «`for`». Данный тип хорошо подходит для задач, когда необходимо выполнить цикл со счетчиком, но следует упомянуть, что в большинстве случаев «`for`» можно заменить на цикл «`while`», задействовав несколько дополнительных переменных. Пример использования цикла «`for`» приведен на рис. 8, данный фрагмент скрипта выполняет 100 итераций начиная от 1 (включая 1 и 100). При этом можно поменять условия местами, вместо 1 поставить 100 и 100 вместо 1, тогда цикл будет выполняться с изменением значения переменной «`i`» от 100 до 1. Также в цикле «`for`» имеется необязательный параметр шага, по умолчанию шаг равен единице, и нет необходимости его указывать. Однако, для случаев, когда требуется использование шага со значением отличным от 1, то нужно указать его через запятую после конечного значения (если необходим шаг 2, то для примера на рис. 8 это будет условие «`i = 1, 100, 2`»).

```
10 Sum = 0
11 for i = 1, 100 do -- 100 iterations from 1.
12   Sum = Sum + i
13 end
```

Рисунок 8 – Пример использования цикла «`for`».

Таблицы в Lua являются единственным структурным элементом, они сочетают в себе свойства массива, хэш-таблицы («ключ» - «значение»), структуры и объекта. Чаще всего таблицы используются в качестве словарей, а ключ при этом по умолчанию имеет строковый (символьный) тип. Пример приведен на рис. 9. Строка 10 объявляет переменную типа «таблица» и задает 2 ключа и соответствующие им значения. Доступ к значениям можно получить указанием названия таблицы и ключа через точку (пример на рис. 9).

```

11 | t = {key1 = 'value1', key2 = false}
12 | print(t.key1) -- 'value1'
13 | t.newKey = {} --add new key
14 | t.key2 = nil -- delete key2

```

Рисунок 9 – Пример использования таблиц на языке Lua

Как видно из примера, допускается использование в качестве ключа не только строк, но также и всех остальных типов переменных, которые доступны в Lua. Другой и крайне важной составляющей языка программирования являются функции. Функции можно создавать свои собственные, также можно использовать уже готовые, которые можно найти в справочнике по Lua[7]. Пример заданной разработчиком функции приведен на рис. 10. Как видно из примера, функция может принимать несколько значений. Также функции могут возвращать несколько значений.

```

6 | function bar(a, b, c)
7 |     sum=a+b+c
8 |     r=a-b-c
9 |     return sum,r
10| end
11| k,p=bar(2,2,3)
12| simAddStatusBarMessage('k='..k)
13| simAddStatusBarMessage('p='..p)

```

Рисунок 10 – Пример использования пользовательской функции на Lua

Особое внимание стоит уделить регулярным функциям V-REP. Эти функции уже интегрированы в Lua при написании скриптов V-REP и начинаются на «sim». Именно эти функции и используются для взаимодействия с симуляцией: управления, считывания данных, отладки и решения многих других задач. Полный список функций с подробным описанием принимаемых ими переменных и возвращаемых данных доступен в официальной документации V-REP [8].

Вопросы для самоконтроля:

1. Какой тип скриптов является наиболее предпочтительным?
2. В какой части скрипта допускается объявление новой переменной?
3. Какие циклы используются в V-REP?
4. Сколько переменных может принимать функция?
5. Как можно отличить функции V-REP от функций языка Lua?
6. Можно ли создавать собственные функции?

2.4 Типы объектов программе V-REP

В программе V-REP доступно большое количество объектов различного назначения. Некоторые из них применяются крайне редко, а некоторые нужны при моделировании каждой мехатронной и робототехнической системы. В данном разделе изложены основные типы объектов, их свойства и назначение.

Типы физических объектов (формы). Формы в V-REP представляют собой жесткие сетчатые объекты, состоящие из треугольных граней. Они могут быть импортированы, экспортированы и отредактированы. Они входят в три разных подтипа: простая случайная форма, составная случайная форма, простая выпуклая форма, составная выпуклая форма, чистая простая форма, чистая составная форма. На рис. 11 представлены пиктограммы всех типов в том же порядке слева направо, как указано выше.



Рисунок 11 – Пиктограммы различных типов форм в V-REP

Пиктограммы типа формы выводятся в главной иерархии сцены слева от названия каждого компонента механической составляющей робототехнической системы. Как было указано, все они делятся на три типа, и каждый тип - на простой (единый) и составной (состоящий из нескольких простых).

Отличительные особенности случайных форм заключаются в том, что они могут быть любой формы, но для моделирования динамики их не рекомендуется использовать, т.к. это потребует значительных вычислительных ресурсов. Поэтому они применяются часто для получения хороших визуализаций, зачастую моделируют только визуальные свойства и являются повторяющимися движение выпуклых и чистых форм.

Выпуклые и чистые формы содержат оптимизированную сетку и хорошо подходят для моделирования динамики, однако визуальные свойства выпуклых форм зачастую выглядят пугающе. Поэтому рекомендуется использовать выпуклые формы вместе со случайными формами, где выпуклая форма участвует в расчетах физических свойств (при этом визуальные свойства скрыты), а случайная - в расчетах визуальных свойств (физические свойства отключены).

V-REP также позволяет создавать оптимизированные для моделирования выпуклые формы на основе случайных форм, что более подробно рассмотрено в лабораторных работах.

Добавление форм в среде V-REP может быть выполнено двумя способами: через встроенный инструмент «Primitive Shape» и через меню [File→import Mash from...]. В последнем случае необходимо заранее создать трехмерную модель в любой CAD-системе и сохранить модель в формате STL.

Соединение элементов в V-REP. Шарнир (Joint) - это объект, который имеет хотя бы одну внутреннюю степень свободы. Соединения используются для создания механизмов и задания перемещений других объектов. В V-REP доступно 3 типа шарнирных соединений: вращательный шарнир, призматический и сферический.

Шарниры добавляются в сценарий симуляции через главное меню: [Add → Joint]. Новые шарниры имеют нулевые координаты в глобальной системе координат сцены. Поэтому необходимо после добавления шарнира корректно задать необходимую ориентацию, воспользовавшись инструментами «Object/Item Shift» и «Object/Item Rotation». После того, как у шарнира будет установлено необходимое положение и ориентация, можно соединять шарнир с другими объектами путем создания древовидной иерархии. Данный процесс более подробно раскрыт в лабораторных работах.

Соединения с нулевой степенью свободы в программе V-REP выполняются без использования элементов типа шарнир. Два динамических элемента могут быть соединены через элемент типа «Force sensor». Существует и альтернативный вариант - воспользоваться контекстным меню («Edit» → «Grouping/Merging» → «Group selected shapes»), предварительно выделив необходимые для соединения компоненты. Через инструмент группировки допускается соединение неограниченного количества элементов.

Объекты типа «шарнир» имеют большое количество настроек. Они, в частности, позволяют моделировать работу встроенных двигателей. Также визуальные свойства шарниров могут быть отредактированы через свойства объекта. Визуальные свойства никак не влияют на симуляцию и используются исключительно для удобства работы при создании сценария симуляции.

Графики. Вывод данных на графике является одним из самых распространенных и удобных способов представления. В программе V-REP имеется отдельный класс объектов «Graph» (график), с помощью которого можно легко визуализировать как данные с отдельных сенсоров, так и пользовательские данные (независимо от способа получения). Чтобы добавить графики в сценарий симуляции, достаточно в главном меню выполнить [Add→Graph]. Далее необходимо задать свойства. Для этого нужно выделить объект в иерархии сценария и активировать инструмент «Object/Item Properties». В появившемся контекстном меню в разделе «Data stream recording list» необходимо нажать «Add new data stream to record». В появившемся контекстном окне (рис. 11.1) необходимо указать тип данных и источник данных. Если необходимо вывести данные с сенсора на график, то необходимо указать тип данных в выпадающем поле «Data stream type» и название сенсора в поле «Object/Item to record». Зачастую возникает необходимость в предварительной обработке данных с сенсоров перед выводом на графике: сделать это можно через использование скрипта в качестве промежуточного пункта, где и будет выполняться обработка данных. В таком случае необходимо

указать в настройках графика, что выводиться будут пользовательские данные, т.е. установить «Various: user-defined» в поле «Data stream type», а в поле «Object/items to record» выбрать «User data».

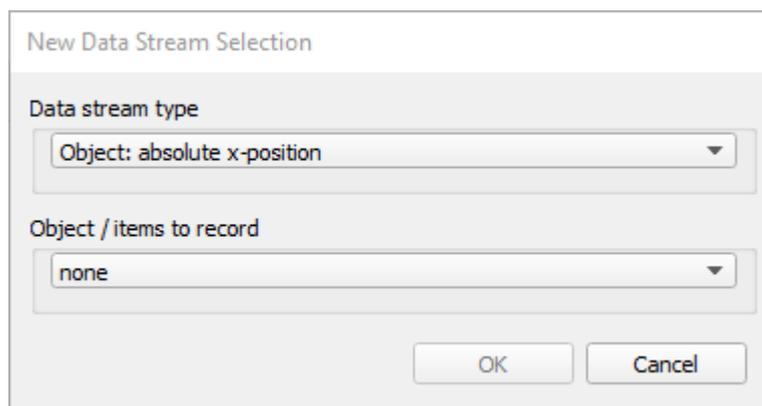


Рисунок 11.1 – Контекстное окно создания потока данных в V-REP

После нажатия «OK» в разделе «Data stream recording list» появится новый поток данных с названием по умолчанию. Двойным нажатием на название потока данных можно перейти в режим редактирования названия. Данное название будет использовано для обращения к графику из скрипта при выводе данных.

```
3 | if (sim_call_type==sim_childscriptcall_initialization) then
4 |     graph=simGetObjectHandle("Graph")
5 | end
6 | if (sim_call_type==sim_childscriptcall_actuation) then
7 |     simSetGraphUserData(graph, 'Red', data)
8 | end
9 |
```

Рисунок 11.2 – Фрагменты скрипта для вывода пользовательских данных на график в программе V-REP

На рис. 11.2 представлен пример фрагмента скрипта, который выводит пользовательские данные из скрипта на график. В примере на строке 4 «Graph» - название объекта, в строке 7 «Red» - название потока данных, «data» - название переменной, значение которой выводится на графике. Как получить данные с сенсора, смотрите в разделе «Сенсоры».

Сенсоры. В программе V-REP доступны различные типы сенсоров: сенсор силы и моментов (force sensor), видео-сенсоры (vision sensor), сенсоры расстояния (proximity sensor). Чтение данных с сенсоров выполняется в скрипте с использованием специальных функций (API-функций). Примеры чтения данных с «vision sensor» и «proximity sensor» приведены на рис. 11.3.

```

2 if (sim_call_type==sim_childscriptcall_initialization) then
3     sensor1=simGetObjectHandle('Sensor')
4     sensor2=simGetObjectHandle("Proximity")
5 end
6 if (sim_call_type==sim_childscriptcall_actuation) then
7     result1,data=simReadVisionSensor(sensor)
8     if (result1>=0) then
9         simAddStatusBarMessage('Data[1]='..data[1])
10    end
11    result2,distance=simReadProximitySensor(sensor2)
12 end

```

Рисунок 11.3 – Фрагменты скрипта для чтения данных из сенсоров в программе V-REP

Вопросы для самоконтроля:

1. Можно ли создать трехмерную модель робота в V-REP без использования дополнительного программного обеспечения?
2. Какие типы соединений доступны в V-REP?
3. Будет ли работать симуляция, если для моделирования динамики использовать «случайные» типы формы?
4. Имеются ли в V-REP объекты типа «мотор», если да, то в каком виде они реализованы?
5. Какие типы данных можно выводить на график?

3. Лабораторный практикум

Лабораторная работа №1

Моделирование работы механического захвата промышленного манипулятора

Цель работы: Создать трехмерную компьютерную модель и создать сценарий симуляции простейшего механического захвата для манипулятора.

Задание.

- 1.) Создать трехмерную модель простого захвата, состоящего из трех элементов, в любой CAD-системе и импортировать в V-REP (либо создать модель механического захвата непосредственно в V-REP).
- 2.) Задать структуру модели (задать взаимосвязи).
- 3.) Добавить в модель несколько сенсоров и написать алгоритм управления, который обеспечивает управление захватом. У захвата должно быть два устойчивых положения: активное и исходное.

В данной лабораторной работе предлагается реализовать симуляцию работы механического захвата для промышленного манипулятора. Перед тем как приступить к выполнению работы, необходимо предварительно ознакомиться с описанием лабораторной работы и ответить на вопросы для самоконтроля.

Вопросы для самоконтроля:

1. Какая строка скрипта на рис. 28 является необязательной?
2. Каких типов бывает шарниры в программе V-REP?
3. Какое условие нужно изменить в скрипте (на рис. 28), чтобы захват активировался не зависимо от расстояния до объекта?
4. Как задаются связи в программе V-REP?
5. Сенсоры каких типов доступны в программе V-REP?
6. Какая функция позволяет получить данные сенсора?
7. Почему в данной работе был использован непотоковый тип скрипта?
8. Можно ли реализовать алгоритм работы захвата с использованием потокового скрипта?
9. Чем отличаются созданные в V-REP трехмерные модели от моделей, загруженных из CAD-системы?
10. Разрешается ли использовать вложенные условия (ветвления)?
11. Почему управляющий скрипт прикреплен именно к базовому элементу механического захвата?

Описание работы.

Трехмерную модель захвата можно создать в любой CAD-системе и импортировать в V-REP без особого труда, данный метод рассматриваться в рамках лабораторной работы №1 не будет.

Для создания трехмерных моделей механического захвата непосредственно в программе V-REP создадим новую сцену. В новой сцене будем пользоваться инструментами Primitive Shape для создания захвата, который будет состоять из трех прямоугольных параллелепипедов (кубоидов).

Используя команды [Add→Primitive Shape → Cuboid] и вводя следующие параметры (как показано на рис. 12), можно получить трехмерную модель основания захвата.

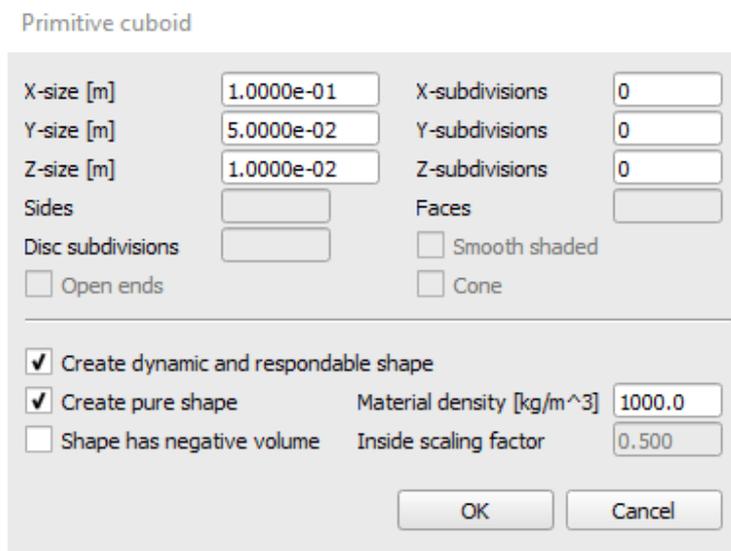


Рисунок 12 – Контекстное меню инструмента «Primitive Shape» (Простая форма) при создании трехмерной модели основного компонента захвата

Далее необходимо создать второй и третий элемент, здесь можно их сделать одинаковыми, следовательно, достаточно создать один элемент и его скопировать.

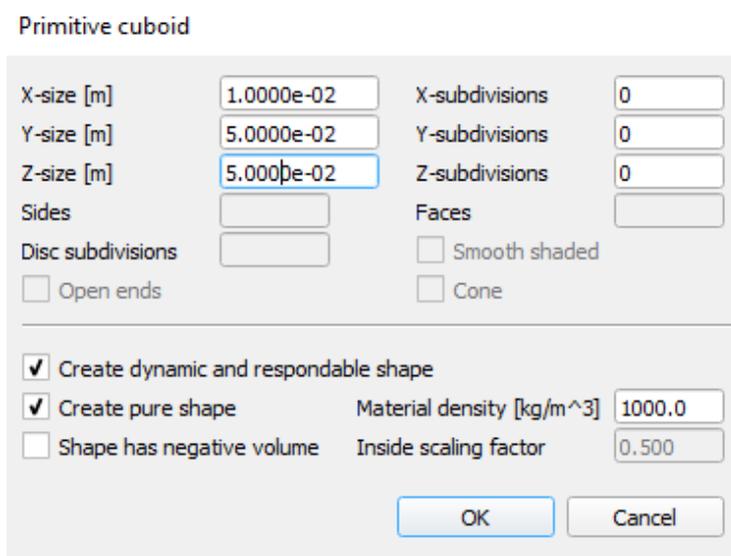


Рис. 13 – Контекстное меню инструмента «Primitive Shape» (Простая форма) для второго компонента

Второй элемент должен иметь параметры, указанные на рис. 13.

Далее необходимо его скопировать и задать всем трем элементам название соответственно их назначению. По умолчанию V-REP задает им имена, соответствующие названию инструмента, с помощью которого они были созданы. Чтобы переименовать, достаточно навести курсор на имя и выполнить двойное нажатие правой кнопки мыши компьютера, задать новое имя и нажать кнопку ввода на клавиатуре (Enter). Обратите внимание, что V-REP не поддерживает кириллические символы. Новые названия должны быть «Gripper_Base», «Right_side» и «Left_side».

Теперь необходимо задать корректное расположение для двух элементов механического захвата относительно основания, соответствующее исходному положению. Для этого выделяем в левом окне дерева модели Left_side и активируем инструмент «Object/Item shift» из верхнего основного меню инструментов. В появившемся окне инструмента выбираем вкладку Position, затем вводим новые значения расположения элемента по оси «x» и по оси «y», т. к. только по этим осям текущие значения не соответствуют требуемым.

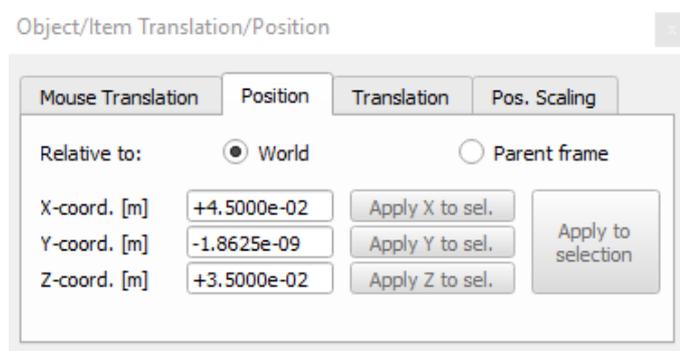


Рис. 14 – Контекстное меню инструмента «Object/Item Position»

Затем необходимо выбрать элемент «Right_side» и задать соответствующие значения с обратным по знаку значением для оси «x».

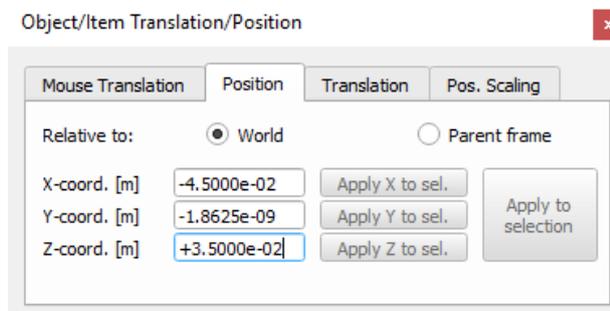


Рисунок 15 – Контекстное меню инструмента «Object/Item Position»

В результате выполнения вышеописанных операций должен получиться результат, представленный на рис. 16.

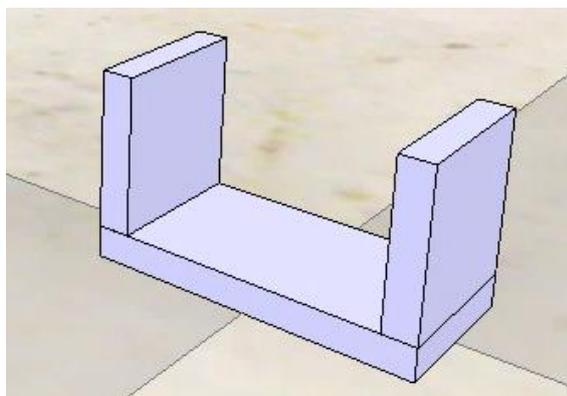


Рисунок 16 – Внешний вид созданных компонентов типа «Random Shape»

Далее необходимо задать механические соединения для имеющихся элементов с помощью элементов «Соединение». В данном случае необходимо добавить два соединения типа «призматический». Добавить их можно через команды «Add» → «Joint» → «Prismatic». Рядом с имеющимися элементами появится новый элемент, который необходимо уменьшить, так как его размер задан по умолчанию и не позволяет нам с достаточной точностью управлять положением элементов нашего механизма. Размеры элементов из категории «Соединение» используются только для удобства работы с ними при создании симуляции и никак не влияют на функциональные параметры соединения.

Поменяем размеры соединения через редактирование свойств, как показано на рис. 17, для этого в дереве модели выбираем необходимый элемент соединения и открываем окно свойств соединения с помощью инструмента «Scene Object Properties» (первый инструмент в левой панели).

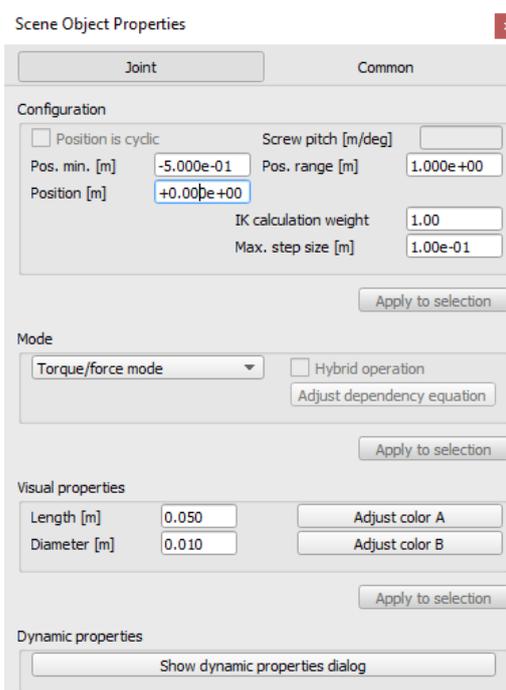


Рисунок 17 – Контекстное меню инструмента «Свойства»

Далее необходимо задать правильное расположение и ориентацию в пространстве для элемента соединения, что и можно сделать, воспользовавшись инструментами для изменения положения и ориентации.

Опишем более подробно основные приемы, которые позволяют выполнить вышеописанные операции.

Чтобы задать положение для соединения, мы можем воспользоваться визуальным интерфейсом и с помощью мышки «перетащить» элемент на требуемое положение, однако такой подход не рекомендуется, т. к. не гарантирует высокой точности. Рекомендуется использовать более быстрый способ – скопировать координаты другого элемента и уже задавать расположение относительно нового положения. Скопировать координаты другого элемента можно, выполнив следующие операции: сначала выбираем в дереве модели объект, в который хотим скопировать координаты, и при зажатой клавише «ctrl» выбираем второй элемент, чьи координаты мы и хотим скопировать. Затем выбираем инструмент «Object/Item shift» и во вкладке «Position» нажимаем на кнопку «Apply to selection» (показано на рис. 18).

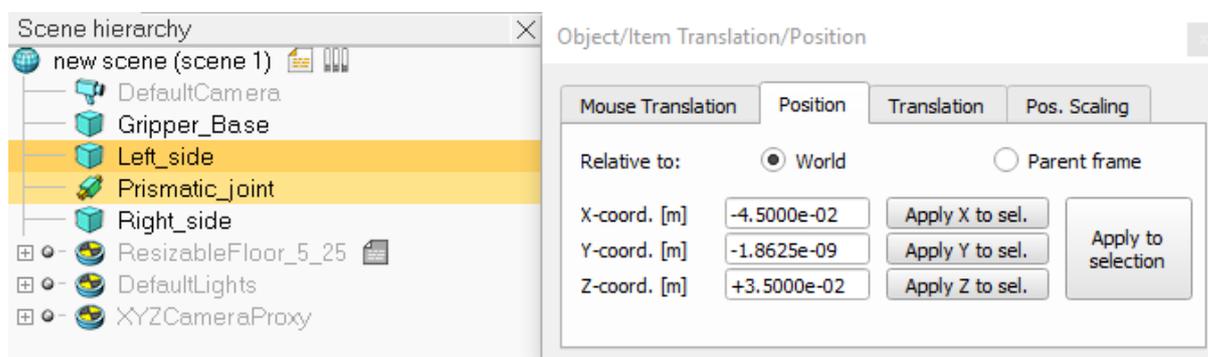


Рисунок 18 – Копирование положения элемента «Left_side» в свойства шарнира

Таким же способом можно скопировать параметры ориентации с одного объекта в другой, воспользовавшись инструментом «Object/Item rotate» вместо «Object/Item shift».

Однако одного копирования данных ориентации и положения бывает недостаточно, как и в нашем случае. В таких случаях необходимо воспользоваться вкладкой «Translation» для инструмента «Object/Item shift», где можно указать значение перемещения относительно текущего положения, и после нажатия кнопки «Translate» будут применены указанные изменения.

Похожая функция имеется и у инструмента «Object/Item rotate», где во вкладке «Rotation» можно указать углы поворота относительно текущей ориентации, выбрав ось вращения, и после нажатия кнопки «Rotate selection» получить результат. При использовании данного инструмента стоит обратить внимание, что необходимо выбрать, относительно какой системы координат выполняется вращение. В большинстве случаев эта операция выполняется относительно

системы координат, связанной с вращаемым элементом, и в таком случае следует в параметре «Relative to» установить на «Own Frame».

Таким образом, воспользовавшись вышеописанными приемами, мы можем получить результат, представленный на рис. 19.

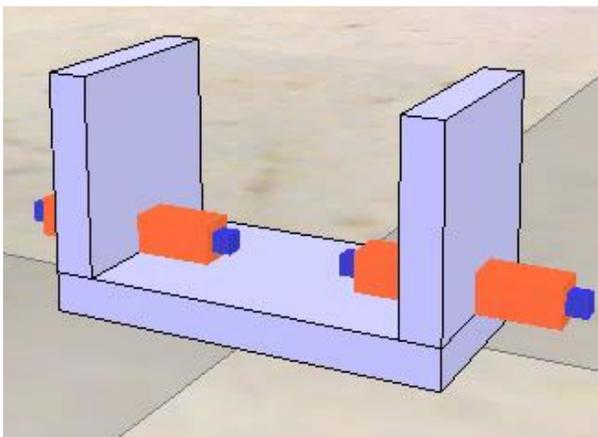


Рисунок 19 – Внешний вид элементов сцены с корректными координатами и ориентацией

После того, как задали требуемое положение и ориентацию для всех элементов, необходимо задать связи этих элементов.

Элемент под названием «Gripper_Base» является основой нашего механизма и, следовательно, все связи должны исходить из него. В V-REP доступны два способа для определения связей. Первый способ наиболее простой и быстрый - выделить и перетащить один элемент на другой в дереве модели. При этом элементы должны образовывать древовидную структуру, где один элемент стоит в основе (базовый элемент), а последующие элементы могут соединяться с основанием или другим элементом одной из ветвей дерева. Второй способ – выделить элемент, который необходимо соединить с родительским элементом, нажать кнопку «ctrl» и выделить элемент, который хотим сделать дочерним, затем нажать правую кнопку мыши и в меню выбрать «Edit», «Make last selected Object parent». Также при определении связей необходимо соблюдать следующее правило: два элемента, моделирующие динамику, нельзя соединять без использования элемента типа «шарнир», т.е. чтобы соединить элементы Gripper_Base и Left_side, необходимо между ними в дереве добавить «Left_joint».

Таким же образом необходимо задать связи между «Gripper_Base» и «Right_side». Итоговая древовидная структура, которую необходимо получить, представлена на рис. 20.

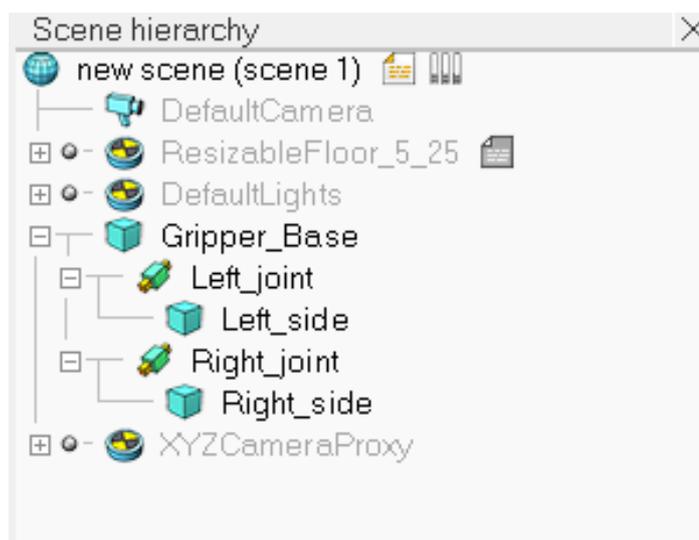


Рисунок 20 – Итоговая иерархия сцены с заданными связями компонентов

Далее необходимо включить встроенные моторы в шарнирах «Left_joint» и «Right_joint», задать необходимые настройки. Для этого нужно выбрать элемент в дереве модели и в левой панели активировать инструмент «Scene object properties», далее в появившемся контекстном меню инструмента необходимо выбрать «Show dynamic properties dialog» и включить мотор, поставив соответствующую отметку напротив «Motor enabled». После этого активируются строки номинальной скорости вращения и максимального момента – «Target velocity» и «Maximum force» соответственно. Ввиду сравнительно малых размеров захвата максимальную силу можно поставить на 5.0e+00 Ньютон и скорость оставить на 0 м. с., т. к. в данной работе его будем задавать через скрипт. Обратите внимание, что необходимо учитывать направление призматического шарнира: в зависимости от его ориентации нужно указать скорость для каждого из захватов с положительным или отрицательным знаком. Определить знак можно экспериментальным способом, проведя симуляцию сценария и принудительно включив моторы и добавив значение к скорости. Если положительное значение не совпадает с требуемым направлением, то необходимо развернуть шарнир на 180°. Пример настройки шарнира приведен на рис. 21.

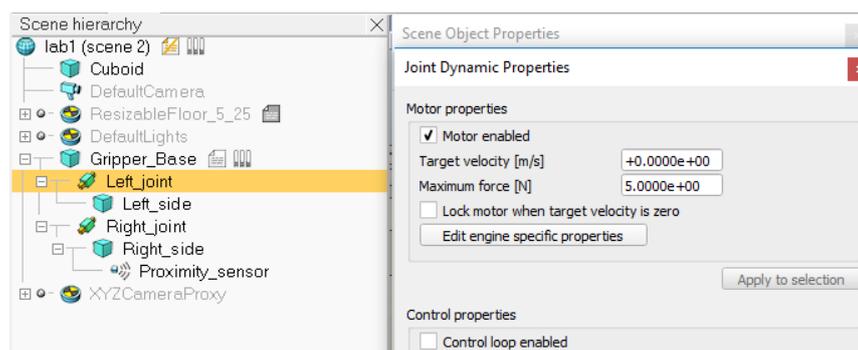


Рисунок 21 – Итоговая иерархия сцены с заданными связями компонентов

Теперь необходимо добавить «Proximity Sensor» (датчик приближения). Добавить его можно через главное меню [Add → Proximity Sensor → Ray Type]. Далее необходимо расположить (задать координаты) сенсор на одном из подвижных элементов захвата так, как показано на рис. 23. После этого можно задать связь между подвижным элементом захвата и сенсором через иерархию сцены. Сделать это проще всего методом «перетащить и отпустить», перетаскиваем сенсор «Proximity_sensor» на название того элемента захвата, с которым совместили сенсор, и отпускаем. Итоговая иерархия сцены, которая должна получиться, представлена на рис. 22.

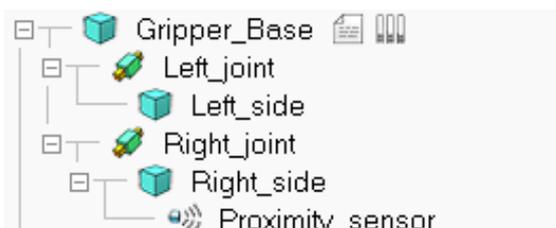


Рисунок 22 – Итоговая иерархия сцены с заданной привязкой «Proximity sensor» к захвату

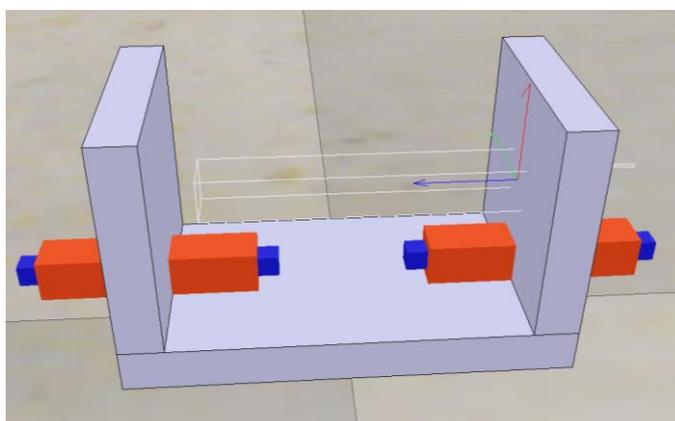


Рисунок 23 – Итоговое расположение всех объектов сцены

В свойствах «Proximity Sensor» необходимо перейти в меню «Show detection parameters» и сделать активным параметр «Don't allow detections if distance smaller than» и установив значение 0. Это связано с погрешностью сенсора и объекта в захвате, который также нужно добавить в сценарий симуляции. Для этого воспользуемся уже знакомым инструментом в меню «Add» → «Primitive shape» → «Cuboid» и зададим размеры, подходящие под масштаб нашего механического захвата.

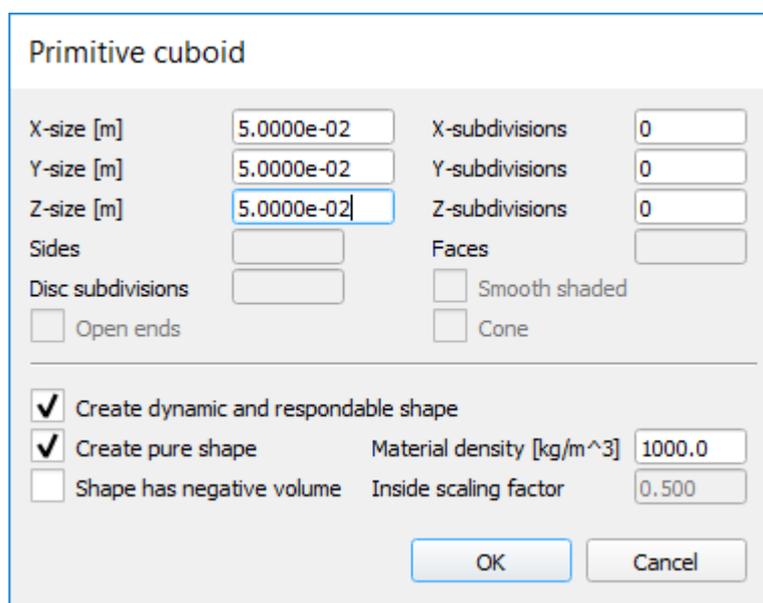


Рисунок 24 – Параметры объекта, который подходит под размеры захвата

Разместить можно сразу в захвате, для этого нужно задать корректное положение. Пример показан на рис. 25.

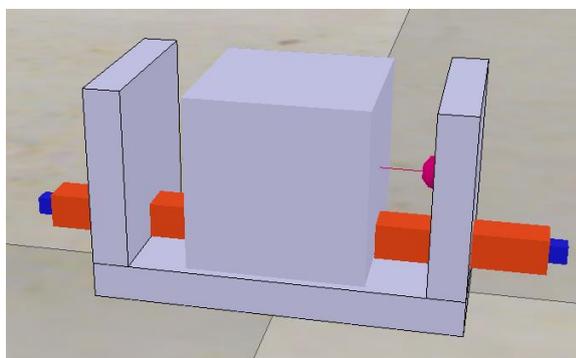


Рисунок 25 – Расположение объекта в захвате

Далее необходимо перейти к реализации системы управления через скрипт. Для этого выбираем элемент «Gripper_Base» и в меню выполняем [Add → Associated Child script → Non-threaded child script]. Теперь в иерархии сцены (часто ее еще называют деревом модели) появилась новая пиктограмма справа от названия элемента «Gripper_Base». Эта пиктограмма означает, что дочерний скрипт привязан к этому элементу. Всегда управляющий скрипт привязывается именно к базовому элементу, т.к. в случае переноса модели удастся избежать проблем с управлением.

Добавить скрипт можно и другим способом, открыв в левой панели инструмент «Scripts», после активации которого можно добавить новый скрипт, нажав на «Insert new script» и в контекстном меню выбрав «Child script (non-threaded)» из выпадающего списка. После этого в списке скриптов появится соответствующая запись «Non-threaded child script (unassociated)», которую еще

нужно привязать: делается это путем выделения скрипта в окне инструмента «Scripts» и выбора в выпадающем списке внизу окна соответствующих объектов привязки (рис. 26).

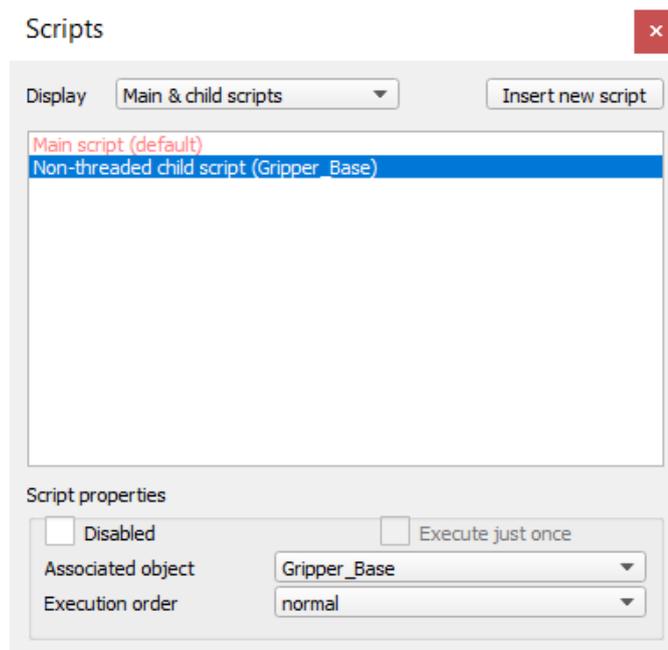


Рисунок 26 – Создание нового скрипта и привязка к базовому элементу через инструмент «Scripts»

После того, как скрипт создан и привязан к базовому элементу, можно приступить к его редактированию. Для этого нужно выполнить двойное нажатие на пиктограмму справа от элемента «Gripper_Base». После перехода в режим редактирования скрипта можно увидеть шаблон скрипта (рис. 27). В данной работе необходимо написать скрипты в двух разделах: блок инициализации и блок управляющего скрипта.

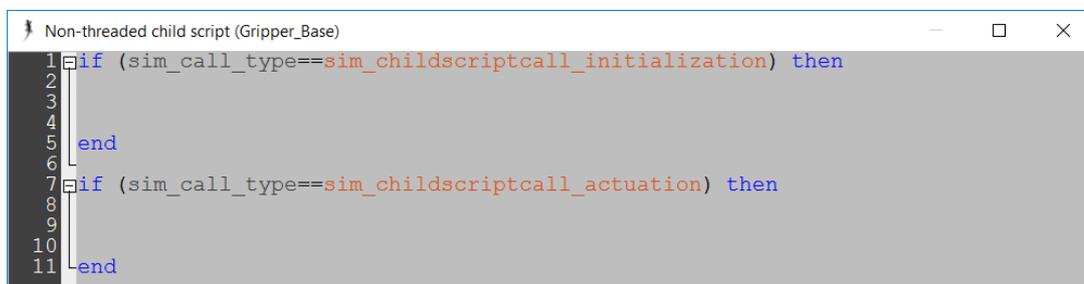


Рисунок 27 – Шаблон скрипта

В блоке инициализации необходимо объявить только названия моторов и сенсора (а точнее, получить ссылки на их обработчики). Перед захватом объекта также необходимо убедиться, что объект находится между подвижными элементами захватного устройства. Чтобы реализовать эту функцию, объявим дополнительную переменную с таким значением, что расстояние до противоположного захвата равно 1. Поэтому значение

переменной «Detection» можно поставить 0.5, что будет равняться определению предмета сенсором до середины захвата.

```
Non-threaded child script (Gripper_Base)
1 if (sim_call_type==sim_childscriptcall_initialization) then
2
3     modelBase=simGetObjectHandle('Gripper_Base')
4     Sensor=simGetObjectHandle('Proximity_Sensor')
5     LeftHandle=simGetObjectHandle('Left_Joint')
6     RightHandle=simGetObjectHandle('Right_Joint')
7     Detection=0.5
8
9 end
```

Рисунок 28 – Пример фрагмента скрипта для блока инициализации

Переходим к основному коду управления: содержимое данного блока напрямую зависит от алгоритма управления. В нашем случае можно использовать следующий алгоритм: если объект находится между подвижными элементами захвата, то активируем захват, иначе ждем. Соответственно в скрипте необходимо реализовать принятие решения захвата, либо ожидания объекта.

В первую очередь необходимо считывать данные с сенсора при каждой итерации. Для считывания данных из сенсора «Proximity sensor» необходимо воспользоваться соответствующей функцией V-REP (обратитесь к справочнику регулярных функций). Эта функция, как видно из документации, на первом месте возвращает состояние сенсора, что в приведенном примере будет записано в переменную «result» (рис. 29, 13 строка). Также из документации видно, что значение этой переменной «1» соответствует рабочему состоянию сенсора, т. е. в рабочей зоне сенсора есть какой-либо объект, «0» - обратному. Вторым значением возвращается расстояние, которое записывается в переменную «distance».

```
12 if (sim_call_type==sim_childscriptcall_actuation) then
13     result, distance=simReadProximitySensor(Sensor)
14     if (result>0) and (distance<Detection) and (distance>0) then
15         simSetJointTargetVelocity(LeftHandle, 0.008)
16         simSetJointTargetVelocity(RightHandle, 0.008)
17     else
18         simSetJointTargetVelocity(LeftHandle, 0)
19         simSetJointTargetVelocity(RightHandle, 0)
20     end
21 end
```

Рисунок 29 – Фрагмент скрипта, отвечающий за реализацию алгоритма управления захватом

Теперь необходимо написать логическую часть кода с помощью ветвления. Для этого воспользуемся условным оператором «if» и входящим в него необязательным условием «else». В 14 строке на рис. 29 ветвление выполняется

тремя условиями – состоянием сенсора, проверкой вхождения в диапазон доступности и условия отключения захвата (нулевое расстояние до объекта).

Если все три условия выполняются, т.е. сенсор обнаружил какой-то объект («result > 0»), расстояние до него меньше значения переменной «Detection» и расстояние до объекта ненулевое, то активируем захваты (задаем скорость движения моторов). В V-REP допускается управление моторами через заданную скорость и через заданное положение. В данной работе будем использовать управление по скорости. Скорость задаем равной 0.008 м/с.

Если же какое-либо условие из указанных в «if» не выполняется изначально или перестанет выполняться, то необходимо деактивировать захватный механизм, т.е. задать нулевую скорость, что уже прописывается в условии «else» (Иначе).

Приведенный в данной работе алгоритм управления является самым простым и имеет ряд недостатков, однако с пониманием всего описанного выше можно с легкостью его улучшить.

Лабораторная работа №2

Моделирование работы промышленного манипулятора

Цель работы: Смоделировать процесс работы промышленного манипулятора, оснащенного сенсором. Манипулятор должен быть запрограммирован на поиск объекта заданного цвета, который лежит в рабочей зоне манипулятора. После обнаружения объекта манипулятор должен остановить поиск и подвести конечное звено для захвата объекта.

Задание.

- 1) Подготовить трехмерную модель промышленного манипулятора в любой CAD-системе и импортировать в V-REP.
- 2) Задать связи компонентов (задать структуру). Разделить визуальную и физическую часть модели на 2 слоя: первая – визуальная, вторая – физическая (используемые для расчетов).
- 1) Реализовать алгоритм для выполнения поиска кубика заданного цвета (см. Таблицу 1) в скрипте.
- 2) Написать алгоритм управления для робота, который должен подвести захват к объекту, зафиксировать его и перенести на заданное расстояние 20 мм от исходного положения в любом направлении.

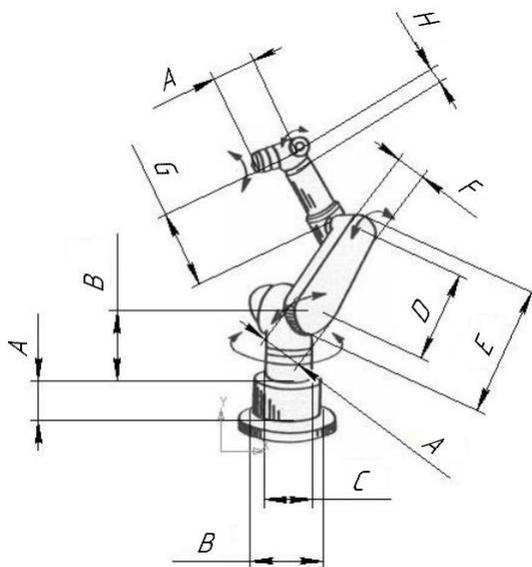


Рисунок 30 – Схематичное изображение манипулятора

Промышленный многозвенный манипулятор является классическим примером динамической системы в мехатронике и робототехнике. Разработка сценария, позволяющего моделировать процесс работы и всю физику взаимодействия компонентов манипулятора, позволяет наиболее полно изучить основные аспекты моделирования большого класса систем.

№	A	B	C	D	E	F	G	H	Цвет
1	80	140	100	180	200	60	160	30	красный
2	80	140	100	180	200	60	160	30	синий
3	80	140	100	180	200	60	160	30	зеленый
4	160	280	200	360	400	120	380	60	красный
5	160	280	200	360	400	120	380	60	синий
6	160	280	200	360	400	120	380	60	зеленый
7	120	180	140	220	240	100	200	70	красный
8	120	180	140	220	240	100	200	70	синий
9	120	180	140	220	240	100	200	70	зеленый
10	100	160	120	200	220	80	180	50	красный
11	100	160	120	200	220	80	180	50	синий
12	100	160	120	200	220	80	180	50	зеленый
13	130	190	150	230	250	110	210	80	красный
14	130	190	150	230	250	110	210	80	синий
15	130	190	150	230	250	110	210	80	зеленый
16	90	150	110	190	210	70	170	40	красный
17	90	150	110	190	210	70	170	40	синий
18	90	150	110	190	210	70	170	40	зеленый
19	40	70	50	90	100	30	80	15	красный
20	40	70	50	90	100	30	80	15	синий
21	40	70	50	90	100	30	80	15	зеленый
22	60	90	70	110	120	50	100	35	красный
23	60	90	70	110	120	50	100	35	синий
24	60	90	70	110	120	50	100	35	зеленый
25	80	110	90	130	140	70	120	55	красный
26	80	110	90	130	140	70	120	55	синий
27	80	110	90	130	140	70	120	55	зеленый
28	100	130	110	150	160	90	150	75	красный
29	100	130	110	150	160	90	150	75	синий
30	100	130	110	150	160	90	150	75	зеленый

Таблица 1 – Варианты с исходными данными

В данной работе рассматривается создание симуляции без углубления в математические аспекты моделирования, что позволяет сконцентрироваться на аспектах практического применения. Основные подходы для моделирования

динамики роботов, а также метод Ньютона-Эйлера, который используется в V-REP для расчета динамики, подробно рассмотрены в учебниках [9] и [10]. Также стоит помнить, что симуляция не дает точности в 100%, допускаются небольшие отклонения на разных этапах выполнения.

Перед тем как приступить к выполнению работы, необходимо предварительно ознакомиться с описанием лабораторной работы и ответить на вопросы для самоконтроля.

Описание работы.

В данной работе подразумевается, что студент уже имеет навыки создания трехмерных моделей в любой CAD-системе. Если таковых навыков нет, то рекомендуем обратиться к учебнику [11], либо другой аналогичной литературе. Сначала необходимо импортировать уже созданную 3D модель манипулятора из CAD-системы в V-REP. Для этого необходимо сохранить сборку манипулятора с расширением *.STL. Каждый элемент сборки сохранится отдельным файлом. Файлы при сохранении в формате STL сохраняют свои координаты, и в дальнейшем будет удобно с ними работать уже в программе V-REP.

Запускаем V-REP и в главном меню выполняем [File → Import → Mesh...], выбираем все сохраненные STL файлы и нажимаем "Открыть".

В появившемся окне выбираем пункт «1 unit represents 1 millimeter», что позволит задать корректный масштаб уже в новой системе координат, также может возникнуть необходимость в изменении ориентации «Mesh orientation». После того, как все настройки корректно заданы, остается нажать "ОК" (рис. 31).

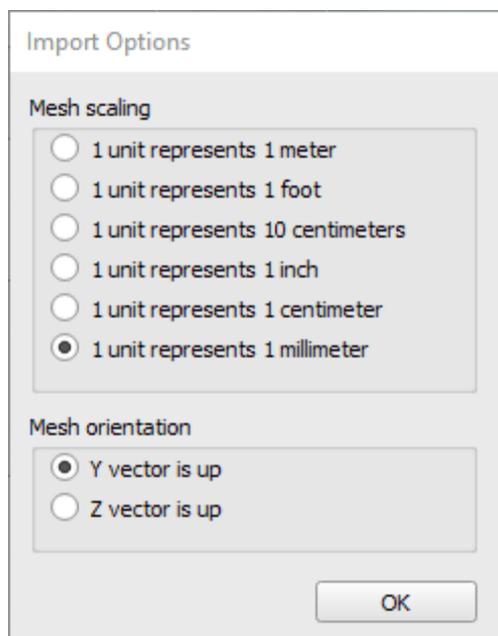


Рисунок 31 – Настройки импорта файлов

После импорта в главном окне должны отобразиться все модели (пример показан на рис. 32). Также можно увидеть, что в иерархии объектов сцены появились новые компоненты, соответствующие каждой части промышленного манипулятора.

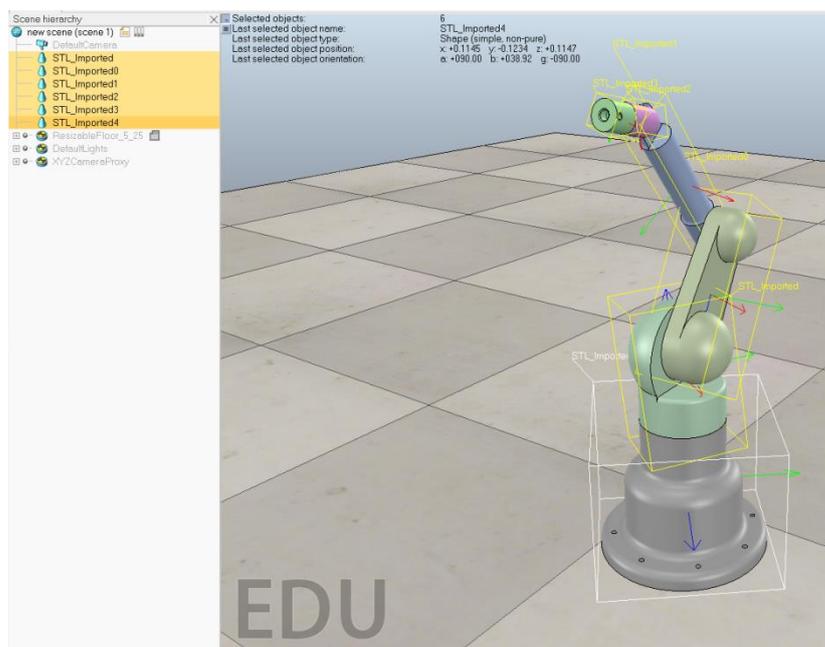


Рисунок 32 – Пример импортированного из CAD-системы манипулятора

Обратим внимание на тип формы каждой компоненты, которую мы импортировали в V-REP. Для идентификации типов удобно пользоваться пиктограммами слева от названия компонента. Сейчас все компоненты имеют тип «Составные случайные формы». Также это можно понять исходя из того, что только этот тип может иметь столь сложную геометрию, как представлено на рис. 32.

В дальнейшем, чтобы было удобнее работать с иерархией сцены, дадим каждой части манипулятора название. Название должно содержать информацию о том, какая это компонента (например, близость к основанию или другая информация для легкой идентификации).

Имеющиеся на данном этапе элементы могут быть использованы для симуляции, однако этого делать не рекомендуется, т.к. элементы данного типа не являются оптимальными для расчета физических свойств. Поэтому создадим модифицированные копии этих элементов, которые будут использоваться для моделирования физики.

Выделяем каждую компоненту, нажимаем правой кнопкой мыши по его названию и во всплывающем меню выбираем [Add → Convex decomposition of selection...].

Стоит обратить внимание, что при изменении параметра «Target nb of triangles of decimated mesh» можно регулировать детализацию сетки создаваемого объекта. Рекомендуемый диапазон значений - от 500 до 1000. Также в контекстном окне инструмента имеется ряд других параметров, которые также позволяют создать оптимизированную сетку.

Выставив необходимые параметры (рис. 33), нажимаем «ОК». Это необходимо повторить для всех компонентов манипулятора. Заметим, что появились новые элементы, и пиктограммы слева от них отличаются от пиктограмм тех элементов, которые у нас были.

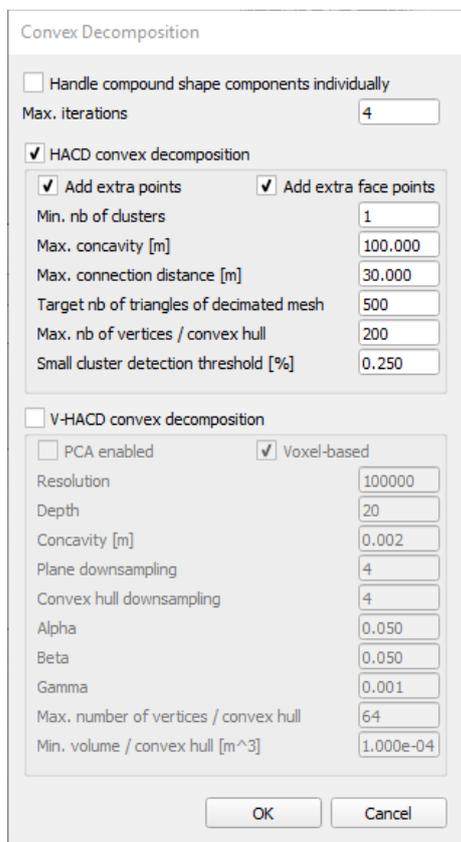


Рисунок 33 – Окно создания копии с оптимизированной сеткой для моделирования динамики

Далее рекомендуется переименовать новые компоненты так, чтобы у них были названия такие же, как у исходных, только с добавлением окончания «_dyn» (помните, что в названии допускаются только латинские буквы).

После этого необходимо связать исходные компоненты с их копиями («_dyn») таким образом, что динамические компоненты являются «родителем» для исходных компонентов. Сделать это можно методом «перетащить и отпустить», выделив исходную компоненту в иерархии сцены (с зажатой левой кнопкой мыши) и перетащив на его копию с динамическими свойствами. Подобную операцию необходимо проделать для всех компонентов

манипулятора. Результат должен получиться схожий с тем, что представлено на рис. 33.

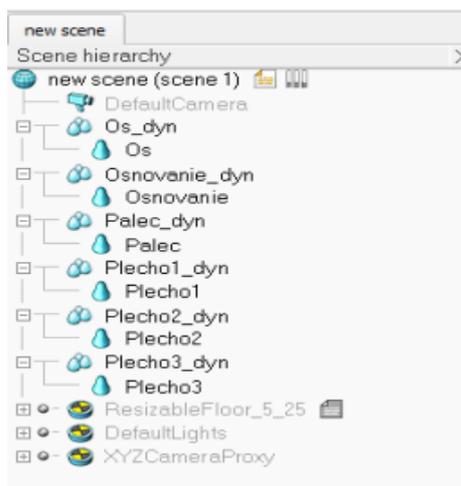


Рисунок 33 – Пример иерархии сцены манипулятора

Теперь в сценарии появились объекты, которые друг друга перекрывают, и необходимо эти объекты разнести на разные слои отображения. Для этого выбираем элемент с оптимизированной сеткой и активируем инструмент «Свойства». Во вкладке «Common» находим пункт «Camera visibility layers» и устанавливаем значения для первого ряда 0000 0000 и 1000 0000 для второго ряда полей. Аналогичную операцию необходимо проделать с другими элементами, у которых оптимизирована сетка.

В итоге получаем визуально изначальный манипулятор, однако с помощью инструмента «Слои» можно переключаться на отображение компонентов другого типа. На рис. 34 представлено отображение только элементов с оптимизированной сеткой.



Рисунок 34 – Пример манипулятора

Теперь осталось настроить элементы таким образом, чтобы оптимизированные под динамику элементы моделировали динамику, а исходные компоненты не участвовали в расчетах, а только повторяли поведение динамических элементов. Итак, открываем «Свойства» динамического элемента и нажимаем «Show dynamic properties dialog». Далее необходимо включить функции «Body is Respondable» (Расчет реакций, связанных с телом) и «Body is dynamic» (Расчет динамических характеристик). В динамических параметрах нажимаем «Compute mass and inertia properties for selected...», чтобы автоматически рассчитать массу и моменты инерции на основе геометрии (рис. 35), после чего надо ввести плотность материала, и V-REP посчитает все динамические показатели. Вышеописанную операцию необходимо повторить для всех частей манипулятора, кроме основания. Основа манипулятора является фиксированной и не должна рассматриваться как динамический элемент, но при этом свойство «Respondable» должно быть включено.

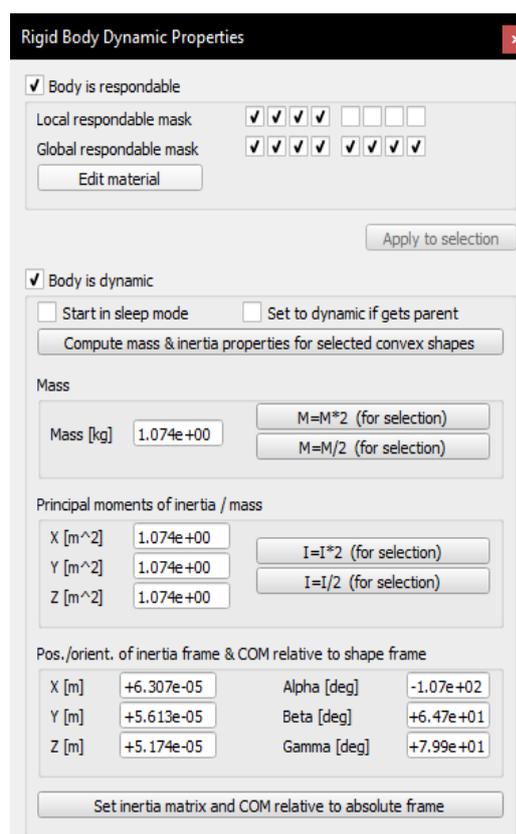


Рисунок 35 – Окно настройки динамических свойств объектов

В программе V-REP есть функция установки маски взаимодействия. С помощью маски можно сообщить программе, взаимодействие между какими элементами нужно игнорировать. Часто бывает, что два твердых тела соединяются с помощью шарнира, и в реальной ситуации они бы не действовали друг на друга. Но в программе V-REP при моделировании мы можем получить грубую сетку, на основе которой и выполняется расчет. Это приводит к тому, что сетки двух рядом расположенных элементов соприкасаются, и для решения этой проблемы потребовалось бы заново

создавать более мелкую сетку, что привело бы к увеличению ресурсов компьютера. Используя маски, можно легко решить подобные проблемы.

Воспользуемся масками объектов для исключения из расчета взаимодействия соседних элементов манипулятора. Для этого открываем свойства основного элемента (основа манипулятора, зафиксированное звено) и устанавливаем маску, как показано на рис. 36.

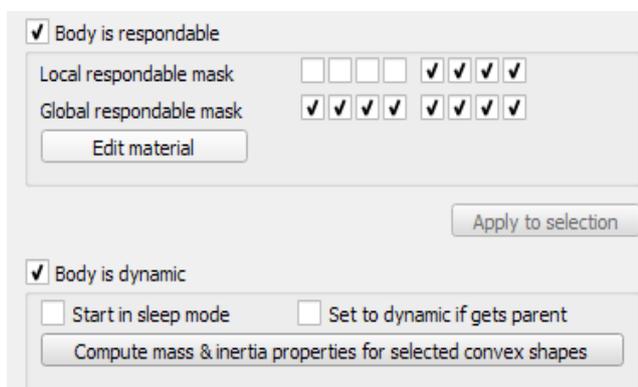


Рисунок 36 – Настройка маски

Далее каждый последующий элемент древовидной структуры должен иметь локальную маску, отличную от маски предыдущего элемента. Например, «Local responsible mask» последующего за основным элементом будет 11110000, а у следующего уже будет такая же маска, как у основания («00001111») и т.д. с чередованием.

Если попробовать запустить получившийся сценарий, то все детали манипулятора должны рассыпаться, так как отсутствуют какие-либо соединения-шарниры.

Для соединения частей компонентов манипулятора нужно добавить в сцену шарниры («Joint») вращательного типа («Revolute»). Сделать это можно через главное меню [Add → Joint → Revolute], после чего в сцену добавляется новый объект. Шарнир представляет собой некую ось, относительно которой будут производиться движения соединенных с шарниром элементов. Исходя из этого необходимо точно задать положение и ориентацию шарниров в точках относительного вращения звеньев манипулятора. Данная задача является одной из сложных, и в этом случае можно прибегнуть к некоторым хитростям. Далее описан один из возможных подходов к этой задаче.

Для позиционирования шарниров, как и в предыдущей работе, воспользуемся координатами уже имеющихся элементов манипулятора. Зажав «Ctrl», выделяем шарнир и основание манипулятора, после этого заходим во вкладку «Position» инструмента «Object/Item Shift» и нажимаем «Apply to selection». Переключаемся на инструмент «Object/Item Rotate» и во вкладке "Orientation" нажимаем "Apply to selection". Но случается так, что шарнир необходимо установить не в центр детали по всем трем координатам, а обязательно

посередине, в нестандартном месте в пазу либо на выступе. В такой ситуации есть два выхода: воспользоваться относительным перемещением или относительным вращением. Для этого выбираем только шарнир (после предыдущей операции может быть выбрано сразу два элемента) и активируем инструмент «Object/Item Shift». Во вкладке «Translation» выбираем относительно «Own frame» (координатная система, связанная с текущим объектом) и вводим, на какое расстояние относительно текущего положения необходимо переместить выделенный объект в соответствующих осях, после чего нажимаем «Translate selection». Аналогично можно вращать шарнир во вкладке «Rotation» инструмента «Object/Item rotate».

Вышеописанный метод решает проблему позиционирования шарниров, но в большинстве случаев ее можно решить еще быстрее, если воспользоваться возможностью разделения элементов с сеткой. Перед тем, как сделать это, лучше воспользоваться дополнительным «черновым» сценарием. Создайте еще один сценарий и скопируйте туда все элементы сцены. Далее в черновой сцене мы должны выполнить позиционирование и после этого скопировать в рабочий сценарий уже корректно спозиционированный шарнир. При копировании объектов из одного сценария в другой их свойства сохраняются. Копирование и вставка выполняются с помощью сочетаний клавиш «Ctrl + C» и «Ctrl + V» соответственно.

В черновом сценарии выбираем элемент типа «составная выпуклая форма» и в контекстном меню правой кнопки мыши выбираем «Edit» → «Grouping/Merging» → «Ungroup selected shapes». Таким образом, можно разделить исходный объект на несколько очень простых форм, и теперь можно воспользоваться координатами любой из полученных форм для позиционирования шарнира. По уже известному методу задаем необходимое положение и ориентацию, после чего копируем шарнир из чернового в рабочий сценарий.

В результате манипуляций получилось задать точное расположение и ориентацию шарнира. Прделав аналогичные действия с другими шарнирами, остается задать читаемые названия, чтобы в дальнейшем было удобно с ними работать.



Рисунок 37 – Пиктограммы различных типов шарнира

Теперь, когда все шарниры на своих местах, можно задавать связи между элементами. Для этого создаем древовидную структуру в иерархии сцены, выставляя одни объекты в качестве дочерних относительно других. Выставление иерархии начинается с динамического объекта основания манипулятора, который является первым родителем, далее идет шарнир, а за шарниром динамический объект, ближайший к основанию, затем шарнир и т.д.

На рис. 38 приведен пример результата, который необходимо получить после выполнения всех вышеописанных действий (обратите внимание на иерархию объектов сценария).

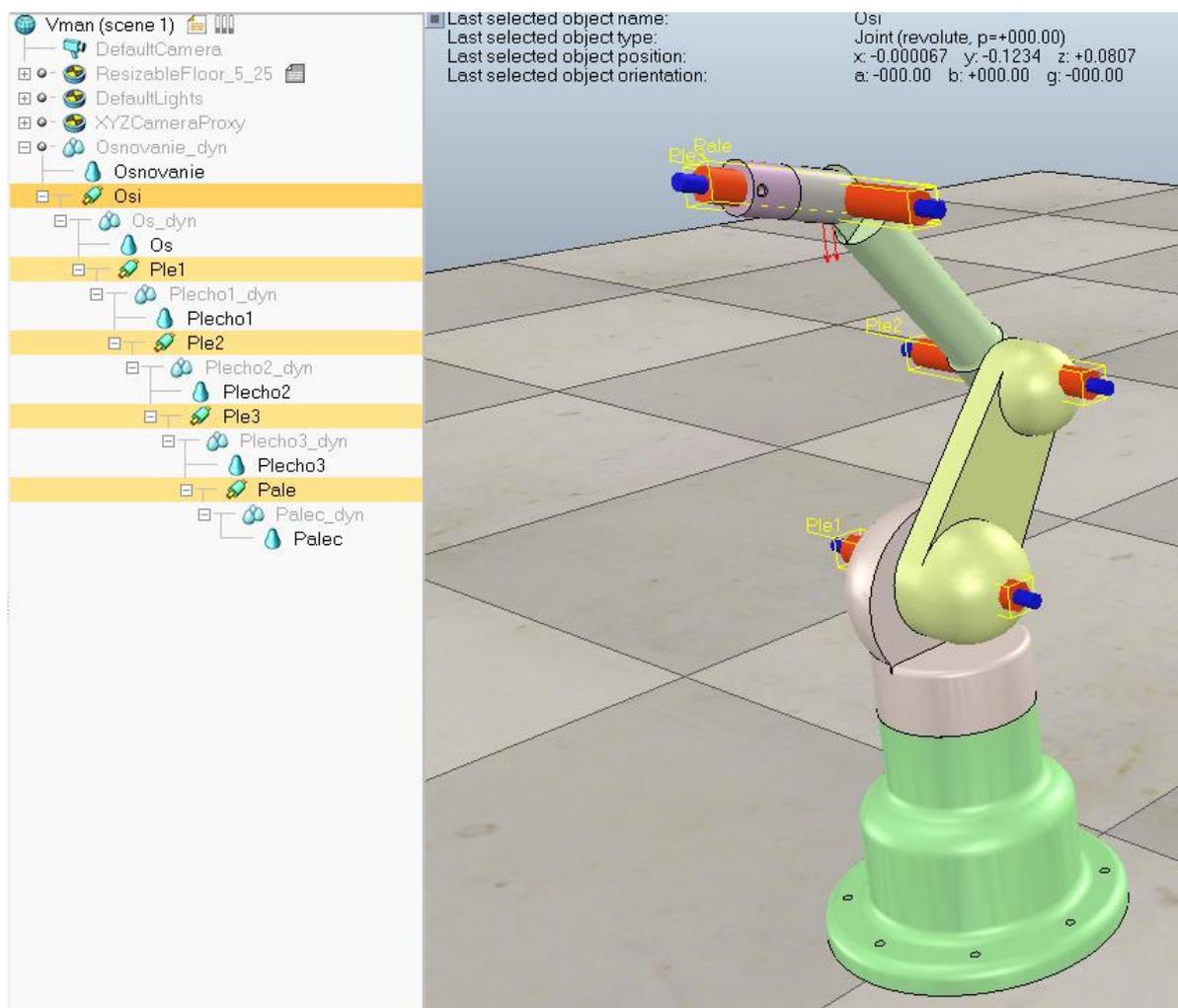


Рисунок 38 – Иерархия объектов и расположение шарниров манипулятора

Для более удобного отображения манипулятора также можно убрать отображение визуальных свойств шарниров.

Приступим к настройке шарниров. Выбираем шарнир в иерархии сцены и выполняем команды [Scene object properties → Show dynamic properties dialog]. В контекстном окне «Joint Dynamic Properties» отметим три важных пункта (рис. 39).

- Motor enabled (включение/выключение мотора).
- Lock motor when target velocity is zero (фиксация мотора при скорости равной нулю).
- Control loop enabled (включение/выключение циклического управления).

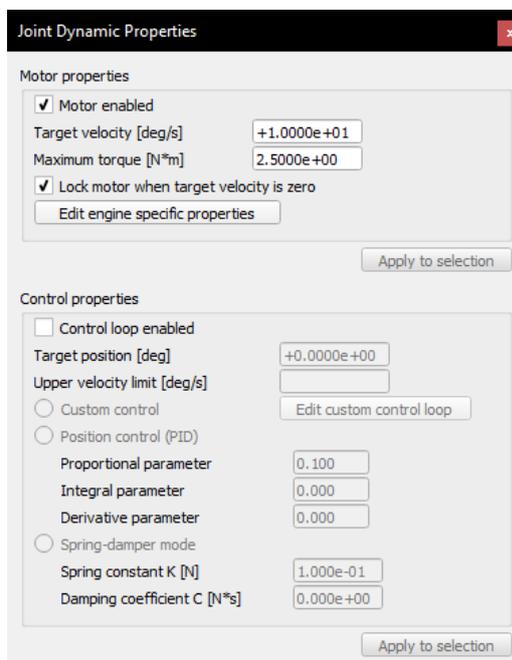


Рисунок 39 – Редактирование динамических свойств шарнира

Ставим галочки напротив "Motor enabled" (включить мотор) и "Control loop enabled" (включить циклическое управление) во всех шарнирах манипулятора. Следующим шагом необходимо установить сенсоры.

Добавляем сенсор через команды [Add → Proximity sensor → Ray type], затем выбираем этот объект и задаем требуемое положение и ориентацию (на конечно звено манипулятора). Устанавливаем привязку к конечному звену манипулятора в иерархии сцены. Этот сенсор будет определять расстояние до объекта. Приступим к его настройке. Выбираем сенсор и переходим [Scene object properties → Show volume parameters]. Параметр "Offset" отвечает за расстояние, с которого начинается отсчет, а параметр "Range" отвечает за дальность работы сенсора.

Добавляем еще один сенсор, который позволит определять цвет объекта. Выполняем команды [Add → Vision sensor → Orthographic type], задаем необходимое положение и ориентацию, затем устанавливаем привязку к конечному звену манипулятора. В данной работе его настраивать не будем, так как настройки по умолчанию отвечают всем задачам.

Теперь добавим куб, который необходимо будет искать манипулятору. В главном меню выбираем [Add → Primitive shape → Cuboid], выбрав необходимые габариты куба, нажимаем "ОК". Выставляем цвет в окне [Scene object properties → Adjust color → Ambient/diffuse component] и устанавливаем его расположение в пределах рабочей зоны манипулятора. Для того, чтобы кубик стал видимым для сенсоров, необходимо поменять его свойства через инструмент [Scene object properties → Common]. Необходимо выставить

свойства "Collidable", "Measurable", "Detectable" и "Renderable" активными, как показано на рис. 40.

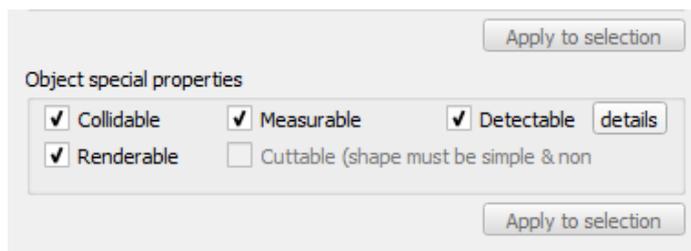


Рисунок 40 – Специальные свойства объектов

Для удобства отображения информации, получаемой с сенсора, добавим график зависимости, который будет отображаться во время моделирования. Создаем график через главное меню [Add → Graph], заходим в его свойства и выбираем «Add new data stream to record». В «Data stream type» выбираем тип «Various: user-defined», а в «Object/items to record» выбираем «User data» и нажимаем "OK".

Приступим к программной части. Выделив основание робота в иерархии сцены, выбираем в главном меню [Add → Associated child script → Non threaded]. Напротив названия появится пиктограмма скрипта (рис. 41), двойное нажатие по которой открывает режим редактирования скрипта.



Рисунок 41 – Пиктограмма дочернего скрипта

Далее приведен скрипт управления манипулятором, который был реализован одним из студентов. Скрипт далек от оптимального как с точки зрения программирования, так и с точки зрения реализации алгоритма управления. Представленный скрипт также является универсальным, т.к. многие параметры зависят от габаритов и особенностей манипулятора. Алгоритм осуществляет поиск и попытку подвести конечное звено к кубу красного цвета. Кубики других цветов манипулятор будет игнорировать.

```

1  if (sim_call_type==sim_childscriptcall_initialization) then
2      glaz=simGetObjectHandle("Vision")
3      Graph=simGetObjectHandle("Graph")
4      rasston=simGetObjectHandle("Proximity")
5      check=0
6      datas=0
7      alt = 0
8      bulev = 1
9      ctrl = 0
10     Osnovanie=simGetObjectHandle('Osi')
11     Zveno1=simGetObjectHandle('Ple1')
12     Zveno2=simGetObjectHandle('Ple2')
13     Zveno3=simGetObjectHandle('Ple3')
14     palec=simGetObjectHandle('Pale')
15     bu = 0
16     nextpositionOsnovanie=0
17     nextpositionOsnovanieMax=7
18     nextpositionPerTarget=1.1
19     nextpositionPer=0.01
20     nextPositionPerMax=0.69
21     nextpositionZveno3=0
22     nextpositionZveno2=0.21
23     nextpositionTik=-2.25
24 end

```

Рисунок 42 – Блок инициализации итогового скрипта

```

25 if (sim_call_type==sim_childscriptcall_actuation) then
26     if (nextpositionPer<nextpositionPerTarget) then
27         nextpositionPer=nextpositionPer+0.1
28         simSetJointTargetPosition(Zveno1,nextpositionPer+1.15)
29         simSetJointTargetPosition(Zveno2,nextpositionPer-0.9)
30         simSetJointTargetPosition(Zveno3,nextpositionTik)
31     end
32     check,datas=simReadProximitySensor(rasston)
33     bu,data=simReadVisionSensor(glaz)
34     simSetGraphUserData(Graph,'Red_stream',data[12])
35     ctrl = data[12]
36     if (nextpositionPer > 1.1) then
37         if (bulev == 1) then
38             if (nextpositionOsn < nextpositionOsnovanieMax) then
39                 if ((ctrl < 0.63) or (ctrl > 0.77)) then
40                     nextpositionOsn = nextpositionOsn + 0.005
41                     simSetJointTargetPosition(Osnovanie,nextpositionOsn)
42                 elseif ((ctrl > 0.63) or (ctrl < 0.77)) then
43                     bulev = 0
44                     simSetJointTargetPosition(Osnovanie,nextpositionOsn + 0.10)
45                 end
46             end
47         end
48         if ((bulev == 0) and (check == 1)) then |
49             if (datas > 0.13) then
50                 nextpositionZveno2 = nextpositionZveno2 - 0.01
51                 nextpositionTik = nextpositionTik + 0.01
52                 simSetJointTargetPosition(Zveno2,nextpositionZveno2)
53                 simSetJointTargetPosition(Zveno3,nextpositionTik)
54             end
55         end
56     end
57 end

```

Рисунок 43 – Блок управляющего кода

Запускаем симуляцию.

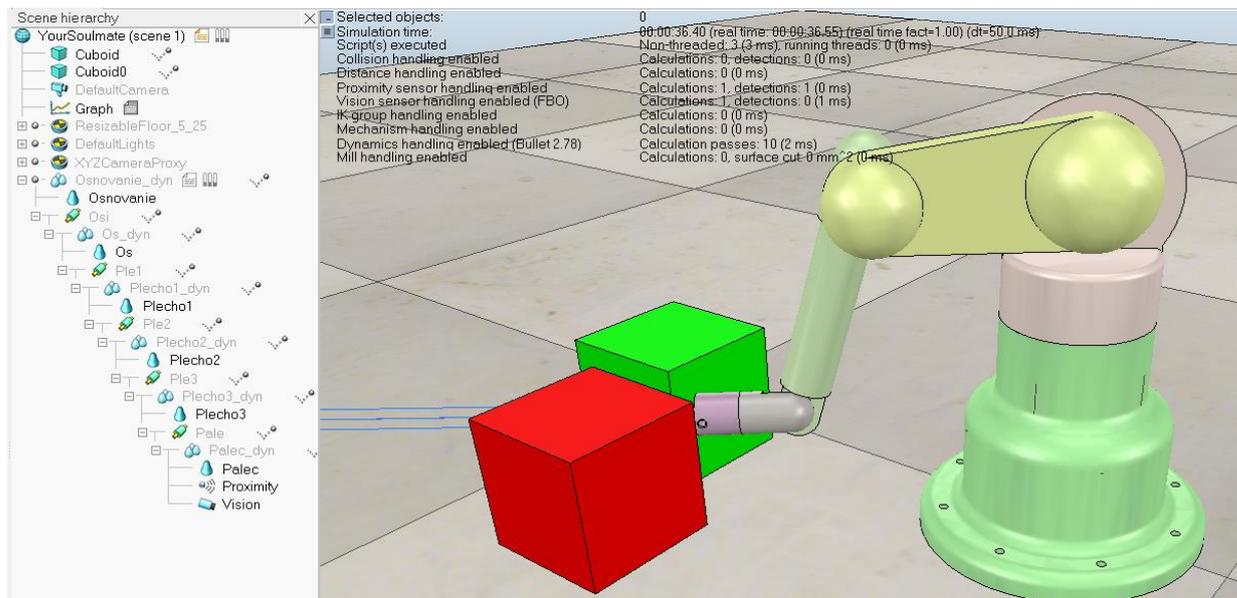


Рисунок 44 – Результат моделирования манипулятора

Чтобы сохранить манипулятор как целую модель, достаточно в свойствах базового объекта (основание манипулятора) дополнительно установить значение «Object is model base» (рис. 42).

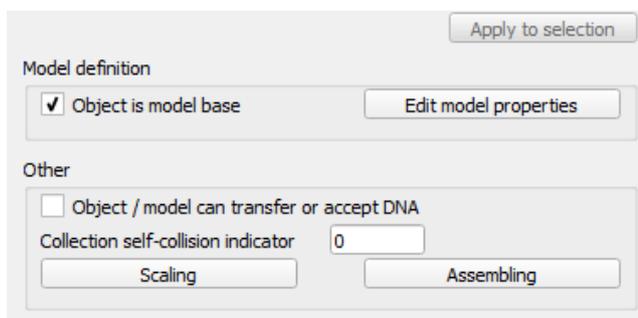


Рисунок 45 – Окно свойств основания манипулятора, раздел «Model definition»

Вопросы для самоконтроля:

1. Можно ли использовать поточный тип скрипта для управления движением колесного робота?
2. Как изменить цвет объекта? В каких случаях цвет может быть проигнорирован фото-сенсорами программы V-REP?
3. Сколько значений возвращает функция считывания данные с фото-сенсора V-REP?
4. Можно ли выполнить предварительную обработку данных из сенсора перед выводом на график?

Лабораторная работа №3

Моделирование работы автономного мобильного робота

Цель работы: Создание симуляции упрощенного мобильного робота и реализация алгоритма управления в автономном режиме. Модель робота должна обладать минимальными характеристиками колесного робота. Алгоритм управления должен обеспечивать движение по замкнутому контуру, основываясь на данных с датчиков.

Задание.

1. Разработать упрощенную модель колесного робота в программе V-REP с использованием инструмента «Primitive Shape» или разработать аналогичную модель в любой CAD-системе и импортировать в V-REP.
2. Задать структуру модели (определить механические соединения), вынести каждый тип элементов на отдельный слой. Задать необходимые свойства механических элементов.
3. Добавить в модель несколько сенсоров и написать алгоритм управления, который позволяет роботу передвигаться по замкнутому контуру согласно ограничениям, которые описаны в пункте 4.
4. Ограничения. Размер робота в исходном положении не может превышать в длину 20 см и в ширину 6 см. Описание трассы: минимальная ширина трассы - 30см, максимальная - 50 см, максимальный угол поворота - 180 градусов. Трасса ограждена стеной, высота которой составляет 20 см.

В данной лабораторной работе предлагается смоделировать процесс работы самого распространенного типа мобильных роботов - колесного. Особенности моделирования роботов данного типа обусловлены характером используемых приводов и алгоритмов, которые отвечают за эффективное выполнение поставленных задач.

В рамках данной работы необходимо не только смоделировать механическую составляющую системы, но и реализовать алгоритм управления, т. к. алгоритм управления является неотъемлемой частью любой современной робототехнической системы.

В ходе работы должна быть использована библиотека моделирования физики «Bullet Physics», также допускается использование библиотеки ODE. Для реализации алгоритма управления необходимо воспользоваться регулярными API функциями, которые представлены в пакете «sim», и описание которых можно найти на сайте производителя программного обеспечения в разделе «Resources».

Вопросы для самоконтроля:

1. Зачем нужен «Control Loop» и почему в симуляции он не используется?

2. Почему выбрано управление через изменение скорости вращения колес? Какие альтернативные способы управления есть?
3. Можно ли использовать «Vision Sensor» вместо «Proximity sensor» в данной симуляции?
4. Можно ли сохранить колесного робота из текущей симуляции как модель и добавить в «Model browser»?

Описание работы.

Рассмотрим простейший вариант реализации автономного робота на четырех колесах. Все четыре колеса независимы, и на каждом из них имеется шарнир.

Опираясь на метод, описанный в лабораторной работе №1 (с помощью инструмента «Primitive Shape»), создаем прямоугольный параллелограмм и четыре цилиндра заданной толщины (см. габаритные размеры в описании к Лабораторной работе). После этого необходимо задать положение и ориентацию для всех компонентов, как показано на рис. 46. Компоненты типа «Простые формы» уже имеют оптимальную сетку для моделирования динамики, поэтому остается только убедиться, что динамические свойства установлены корректно.

Также можно по методике, представленной в лабораторной №2, прибегнуть к помощи любой CAD-системы и импортировать модели в V-REP. В последнем случае необходимо создать копии элементов с оптимизированной сеткой (подробно рассмотрено в предыдущей работе) на основе импортированных CAD моделей. Также не забываем, что в элементах с оптимизированной сеткой необходимо задать все динамические свойства и поставить привязки динамических элементов к исходным элементам.

Далее добавляем в сценарий шарниры с одной вращательной степенью свободы и задаем им расположение (и ориентацию) по центрам колёс, затем устанавливаем древовидную иерархию объектов (рис. 46), тем самым задавая связи элементов модели. Обратите внимание, что в данном случае «основой» робота является корпус, к которому все остальные компоненты привязываются.

Для своевременного поворота колесного робота при движении по замкнутому контуру достаточно воспользоваться двумя сенсорами. Добавляем два сенсора типа «Proximity sensor» в сценарий симуляции и задаем им положение на передней части корпуса колесного робота, также устанавливаем ориентацию под углом 45° в разные стороны от робота и вперед. Выставляем привязки сенсоров в иерархии сцены, тем самым закрепляя сенсоры в текущем положении относительно корпуса.

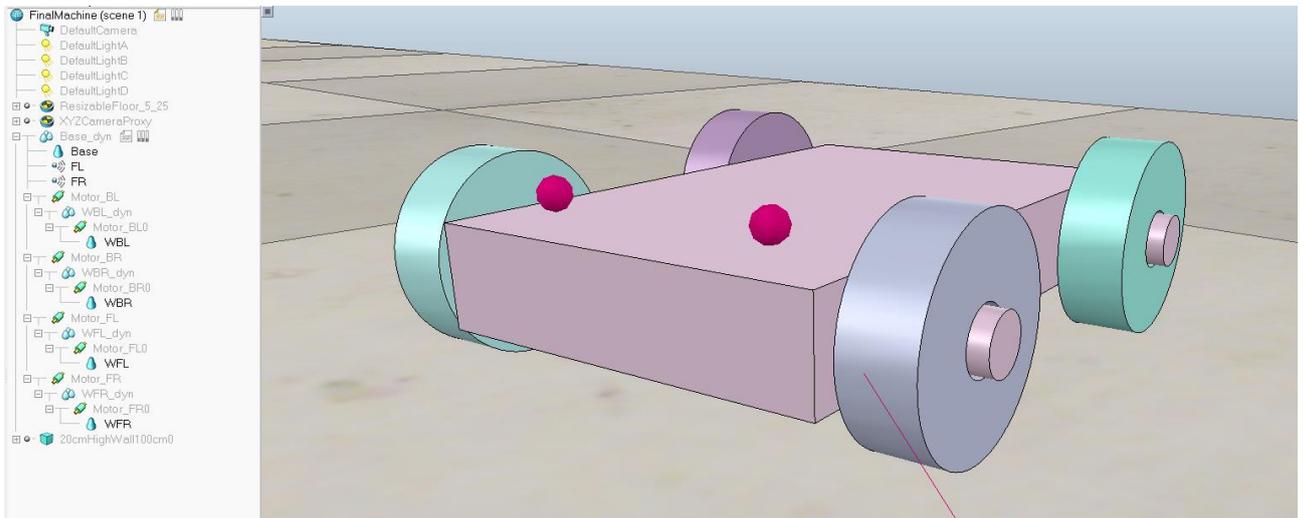


Рисунок 46 – Пример автономного колесного робота

Осталось разработать алгоритм управления и реализовать скрипт. Дочерний скрипт необходимо привязать к основанию робота, т. е. к корпусу колесного робота.

Далее приведен скрипт, разработанный одним из студентов и реализующий простое циклическое движение по замкнутому контуру:

```

1 if (sim_call_type==sim_childscriptcall_initialization) then
2     jointHandle1=simGetObjectHandle('Motor_BL')
3     jointHandle2=simGetObjectHandle('Motor_BR')
4     jointHandle3=simGetObjectHandle('Motor_FL')
5     jointHandle4=simGetObjectHandle('Motor_FR')
6     position1=simGetJointPosition(jointHandle1)
7     position2=simGetJointPosition(jointHandle2)
8     position3=simGetJointPosition(jointHandle3)
9     position4=simGetJointPosition(jointHandle4)
10    simAddStatusBarMessage('position1='..position1)
11    simAddStatusBarMessage('position1='..position2)
12    simAddStatusBarMessage('position1='..position3)
13    simAddStatusBarMessage('position1='..position4)
14    Lsensor=simGetObjectHandle('FL')
15    Rsensor=simGetObjectHandle('FR')
16    Lres=0
17    Rres=0
18    Rdif=0
19    Ldif=0
20    Speed=10|
21 end

```

Рисунок 47 – Скрипт блока инициализации

```

1  if (sim_call_type==sim_childscriptcall_initialization) then
2      jointHandle1=simGetObjectHandle('Motor_BL')
3      jointHandle2=simGetObjectHandle('Motor_BR')
4      jointHandle3=simGetObjectHandle('Motor_FL')
5      jointHandle4=simGetObjectHandle('Motor_FR')
6      position1=simGetJointPosition(jointHandle1)
7      position2=simGetJointPosition(jointHandle2)
8      position3=simGetJointPosition(jointHandle3)
9      position4=simGetJointPosition(jointHandle4)
10     simAddStatusbarMessage('position1='..position1)
11     simAddStatusbarMessage('position2='..position2)
12     simAddStatusbarMessage('position3='..position3)
13     simAddStatusbarMessage('position4='..position4)
14     Lsensor=simGetObjectHandle('FL')
15     Rsensor=simGetObjectHandle('FR')
16     Lres=0
17     Rres=0
18     Rdif=0
19     Ldif=0
20     Speed=10
21 end

```

Рисунок 47 – Скрипт блока инициализации

```

22 if (sim_call_type==sim_childscriptcall_actuation) then
23     Lres,Ldist=simReadProximitySensor(Lsensor)
24     Rres,Rdist=simReadProximitySensor(Rsensor)
25     Rdif=0
26     Ldif=0
27     if (Ldist<Rdist) then
28         Rdif=-5
29         Ldif=5
30     end
31     if (Ldist>Rdist) then
32         Rdif=5
33         Ldif=-5
34     end
35     simSetJointTargetVelocity(jointHandle1, Speed+Ldif)
36     simSetJointTargetVelocity(jointHandle2, Speed+Rdif)
37     simSetJointTargetVelocity(jointHandle3, Speed+Ldif)
38     simSetJointTargetVelocity(jointHandle4, Speed+Rdif)
39 end
40

```

Рисунок 48 – Скрипт основного блока управления

Далее необходимо создать замкнутую трассу для обеспечения условий симуляции. Для этого активируем «Model browser» → «infrastructure» → «walls» и выбираем стены высотой 20 мм. С помощью уже имеющихся моделей стен

можно создать трассу, отвечающую условиям лабораторной работы (На рис. 49 показан один из возможных вариантов).

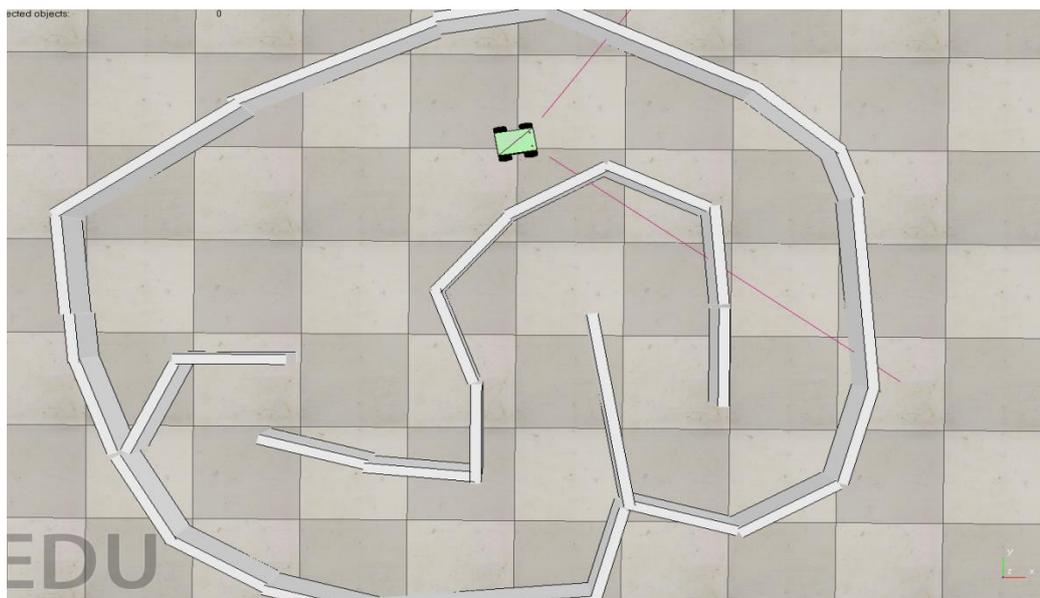


Рисунок 49 – Пример замкнутой трассы для колесного робота

Запускаем симуляцию. Колесный робот должен проезжать заданную трассу, не останавливаясь и не застревая. Возможно, что из-за некорректно выставленной массы и установленной высокой скорости в стартовой позиции робот может не вписываться в повороты или вовсе перевернуться во время начала движения.

Существует довольно распространенный метод движения по заданной траектории, который хорошо описывает некоторые реальные системы. Подобный пример приведен в учебном пособии [12], где полностью раскрываются возможности реализации движущихся систем по заданной траектории.

Заключение

В пособии были подробно рассмотрены основы моделирования робототехнических систем в программе V-REP (Virtual Robotics Experimentation Platform), разработанной компанией Coppelia Robotics и распространяемой свободно в рамках бесплатной лицензии для образовательных целей. Также были рассмотрены базовые подходы для создания симуляций основных типов робототехнических систем. Были подробно описаны процессы планирования и реализации симуляций различными методами. В рамках предлагаемых лабораторных работ студенты знакомятся с особенностями моделирования робототехнических систем. Также студентам предлагаются вопросы для самоконтроля по каждому разделу, которые позволят проверить уровень усвоенного материала.

Список рекомендованной литературы

- 1.) Мониторинг развития образовательной робототехники и IT-образования в городе Москве. Издательский центр АНО «АИР» г. Москва 2017 год.
- 2.) Журнал «Control Engineering». Сергей Колюбин, НИУ ИТМО.
[<http://www.controlengrussia.com/innovatsii/modelnometallicheskaya-obolochka/>]
- 3.) Документация к программному обеспечению ABB Robotic Studio
[<http://developercenter.robotstudio.com/>]
- 4.) Официальный сайт разработчиков Gazebo. [<http://gazebo.org>]
- 5.) Официальный сайт разработчиков V-REP. [<http://coppeliarobotics.com/>]
- 6.) Документация открытой библиотеки планирования движения.
[<http://ompl.kavrakilab.org>]
- 7.) Документация для разработчиков на языке Lua. [<http://www.lua.ru/doc/>]
- 8.) Список регулярных функций V-REP с подробным описанием.
[<http://www.coppeliarobotics.com/helpFiles/en/apiFunctionListCategory.htm>]
- 9.) Динамика робототехнических систем. С.А. Колюбин, Университет ИТМО, 2017.
- 10.) Единый подход к механике Ньютона-Эйлера и Механике Лагранжа. Оливер О'Рейли. Москва, 2011.
- 11.) Журбенко, Гузненков, Бондарева: SolidWorks 2016. Трехмерное моделирование деталей и выполнение электронных чертежей. Учебное пособие.
- 12.) Основы разработки и программирования робототехнических систем. С.В. Сорокин, И.С. Солдатенко, Тверь 2017.

Приложение 1

Терминологический глоссарий

Симуляция – моделирование процесса работы динамической системы за определенный промежуток времени с учетом влияющих на систему факторов с достаточной точностью. Создание симуляций позволяет значительно сократить экономические затраты на создание опытного образца и проведение испытаний. В области тестирования алгоритмов, когда необходимо проверить работу алгоритма с сотнями и даже тысячами различных параметров управления, симуляция приходит на помощь, значительно сокращая временные затраты. В некоторых случаях время на тестирование алгоритмов сокращается с десятков лет на несколько дней, благодаря возможности ускоренной симуляции.

Симулятор – программная платформа, предоставляющая возможность создания различных сценариев с использованием пользовательского интерфейса (без необходимости обращаться к библиотекам моделирования).

Сценарий (сцена) – модель системы с заданным набором параметров, включая все действующие на модель факторы (окружающая среда, посторонние объекты и т.д.).

Решатель физики реального мира (физический движок, библиотека моделирования реального мира) – пакет программных решений (библиотеки), отвечающий за моделирование всех физических законов реального мира (законы всемирного тяготения, свойства и поведение материалов, особенности взаимодействие различных объектов и т.д.).

API – интерфейс программного решения. Посредством API функций описывается взаимодействие физического движка и модели.

Объекты – составляющие части симуляции, которые выполняют задачи моделирования определенных функций и свойств (набора свойств).

Миссия университета – открывать возможности для гармоничного развития конкурентоспособной личности и вдохновлять на решение глобальных задач.

КАФЕДРА МЕХАТРОНИКИ

Развитие мехатроники и робототехники является ключевым звеном в поиске ответов на многие глобальные вызовы. Современные мехатронные системы играют важную роль в промышленности и медицинских приложениях, повышая безопасность и производительность труда, улучшая качество жизни и возвращая людям утраченные возможности здоровья.

Кафедра Мехатроники имеет богатую историю и держит руку на пульсе актуальных научно-технических тенденций. Кафедра стремится реализовывать лучшие мировые практики в своей исследовательской деятельности и подготовке нового поколения квалифицированных и востребованных специалистов.

На кафедре Мехатроники с 2010 года ведет подготовку студентов по направлению «Мехатроника и Робототехника» и «Модульные технологии в биомехатронике».

Редакционно-издательский отдел

Университета ИТМО

197101, Санкт-Петербург, Кронверкский пр., 49