Санкт-Петербургский Государственный Университет Математико-механический факультет

Кафедра системного программирования

Трехмерная модель робота в QReal:Robots

Курсовая работа студента 361 группы

Павлова Сергея Николаевича

Научный руководитель **ст. пр. Брыксин Т.А.**

Санкт-Петербург

2012

Оглавление

Введение

Сейчас активно развивается робототехника, а в связи с этим растет и необходимость обучать людей программированию для роботов. На базе проекта QReal[1] создано средство для обучения основам программирования и кибернетики QReal:Robots. В него уже встроена двумерная модель, которая позволяет эмулировать поведение робота на компьютере. Однако она не может отобразить взаимодействие робота с окружающим миром полностью, поэтому было решено добавить и трехмерную модель. В связи со сложностью реализации будет использоваться сторонняя технология отображения трехмерного мира и его составляющих.

Постановка задачи

На основе одного из средств отображения трехмерной графики реализовать трехмерную модель робота и интегрировать ее с QReal:Robots. Для осуществления этой цели необходимо решить следующие задачи: выбрать более подходящее средство для отображения трехмерной графики, исследовать его возможности, реализовать программу для управления моделью робота, интегрировать ее с QReal:Robots.

Существующие решения

На данный момент существует достаточно большое количество средств, позволяющих создавать трехмерный мир и наполнять его различными моделями. Среди них есть как коммерческие продукты, некоторые из которых бесплатны, так и с открытым исходным кодом. Далее приведен список известных мне 3D-симуляторов[2].

Коммерческие:

- Microsoft Robotics Developer Studio
- Virtual Robot Experimentation Platform
- Visual Components
- SimplyCube
- Actin
- Workspace 5
- Webots
- WorkCellSimulator
- RoboLogix

С открытым исходным кодом:

- Gazebo
- Blender for Robotics and Morse projects

- Breve
- Simbad 3D Robot Simulator
- LpzRobots
- OpenSim
- Moby
- SimRobot
- OpenHPR3

Для проекта QReal:Robots и моей курсовой интересны только те симуляторы, которые предоставляются с открытым исходным кодом, чтобы в дальнейшем можно было внедрить нужные технологии в сам проект, а не использовать в качестве трете-стороннего продукта. Кроме того, в связи с тем, что проект QReal:Robots реализован на c++, подходящих симуляторов осталось немного. Среди них оказался Gazebo, который было решено исследовать первым.

Gazebo

Это симулятор, поддерживаемый Unix-подобными системами и позволяющий создавать популяцию роботов, сенсоры и другие объекты в трехмерном мире. Он предоставляет следующие возможности:

- симуляция стандартных сенсоров, включая сонар, лазерный дальномер, GPS, IMU, а также моно и стерео камеры;
- использование стандартных общеизвестных моделей, таких как Pioneer2DX, Pioneer2AT, SegwayRMP;
- удобный графический интерфейс, который позволяет управлять миром и объектами в нем;
- создание своих собственных моделей роботов и сенсоров, загрузка их в мир во время работы;
- использование “скинов” из 3D-редакторов.

В настоящее время сопровождением проекта Player/Stage/Gazebo занимается Robot Operating System(ROS) сообщество[3], которое предоставляет различные инструменты и библиотеки для создания приложений для роботов. Поэтому проект продолжает активно развиваться: создаются различные плагины, сенсоры, а также сборки для других платформ. После изучения возможностей симулятора было решено, что их достаточно много и их хватит для решения поставленных целей. В связи с этим его взяли в качестве основного средства отображения трехмерной графики, а другие симуляторы не стали рассматривать ввиду ограничения по времени.

Реализация

Взаимодействие с Gazebo

Для взаимодействия с миром используются команды и сервисы, которые присутствуют в сборке от ROS сообщества(далее ros-сервисы и ros-команды). Эти ros-сервисы и команды выполняются через стандартные потоки ввода-вывода, поэтому их использование ничем не затруднено. Эти ros-сервисы были использованы для реализации программы управления моделью робота,

позволяющей осуществить простейшее движение модели, а также работу сенсоров. Каждая команда выполняется при помощи отдельного процесса, который создается в своем потоке.

Запуск мира, который является сервером, получающим дальнейшие запросы к модели, используется команда **roslaunch gazebo_worlds empty_world.launch**. Первый аргумент указывает путь к папке, содержащей launch-файлы миров, а второй аргумент – сам мир. Я запускаю пустой мир, хотя помимо него уже есть и другие готовые, которые тоже можно будет использовать при необходимости. Для добавления модели в мир есть **rosrun gazebo spawn_model -file ~/GFA/model.urdf -urdf -model robo**. Первый параметр gazebo указывает путь к имеющимся командам, таким как spawn_model, которая идет следующим аргументом. После чего указывается -file, который говорит, что далее идет путь к файлу модели, далее указываем -urdf или -gazebo, если файл имеет соответственно расширение .urdf или .xml. Последним параметром в этом примере указывается имя модели, которое будет использоваться в мире, для него и установлен параметр -model. Кроме этого если в конец добавить -x 5 -y 3 -z 1, то наша моделька появится не в стандартной позиции (0,0,0), а в точке (5,3,1). Теперь, что касается управления моделью: для этого используется **rosservice call gazebo/apply_joint_effort '{joint_name: aw1, effort: -20, start_time: 10000000000, duration: 1000000000}'**. Движение осуществляется за счет вращения, так сказать, осей, соединяющих колесо с телом робота. Параметр effort означает силу вращения, то есть скорость движения модели. Этот сервис позволяет запустить движение вперед, для остановки же необходимо использовать его еще раз, но уже для параметра effort указать величину с обратным знаком, то есть в этом примере будет effort: 20. Помимо этого сервиса для движения можно также использовать и **rosservice call gazebo/apply_body_wrench '{ body_name: "robo::wheel1", wrench: { force: { x: 0, y: 0, z: 0 } , torque: { x: -20, y: 0, z: 0 } }, start_time: 10000000000, duration: 10000000000 }'**. Здесь также указывается скорость вращения, но уже не оси, а самого, в данном случае, колеса. За это отвечает параметр torque. С помощью этой команды можно вращать по разным осям, что полезно, если в качестве колеса используется шар. Кроме этого можно указать еще внешнюю действующую силу, используя не нулевые значения параметра force, где x: -20 будет двигать колесо по оси x, а так как оно связано со всей моделью, то и всю модель. Однако с этим параметром я полностью разобраться не смог и при его использовании обнулять его не получалось, то есть всегда, хоть и маленькая, но сила действовала, что приводило модель в движение. Также есть еще один недостаток этого сервиса: при попытке остановить модель, используя снова эту команду только уже с параметром torque: x: 20, она не останавливалась мгновенно, а двигалась по инерции в прежнем направлении, уменьшая свою скорость, до полной остановки. Что касается параметров start_time и duration в обоих сервисах, так я использовал значения, взятые из tutorials от ROS-сообщества, так как не смог найти зависимости от них при разных больших значениях, но при маленьких числах с моделью ничего не происходило. Кроме того, что сервис apply_body_wrench может использоваться для движения при помощи вращения колеса, также его можно использовать для поворота, при помощи того же torque только уже по оси z. Именно так и

осуществляется поворот влево и вправо в моей программе, но поворачивается не само колесо, а специальный элемент соединенный с колесом, вращение которого влияет и на вращение колеса. Последние три команды: spawn_model, apply_joint_effort и apply_body_wrench - выполняются и завершаются, то есть не требуют для себя удерживать поток ввода\вывода долгое время, как это делается, например, для запуска мира, который получает информацию от этих трех команд и выдает ошибку, если пошло что-то не так, и для запуска сенсоров, о которых пойдет речь далее. Перед тем как запускать команду, для получения информации от сенсора, этот сенсор надо добавить к модели, что делается прописыванием специальных параметров в файле для модели. Выделяется нужный элемент, который будет использоваться в качестве сенсора, а если необходимо, то создается новый, и указывается тип этого сенсора. После этого выполняется команда **rostopic echo -c /bbody_bumper/state**, которая отправляет в стандартный поток вывода данные о текущем состоянии указанного сенсора, параметр -c очищает поток после каждого вывода информации. Обновление осуществляется каждую миллисекунду. Чтобы корректно фильтровать получаемые данные и чтобы они не терялись, использовался метод waitForReadyRead(), который запускался после каждого обновления.

Интеграция с QReal:Robots

Перед внедрением Gazebo и программы по управлению моделью в проект надо его сначала подготовить. Сначала опишу архитектуру плагина с роботами. Он содержит в себе несколько подпроектов, мне же предстоит дополнять и изменять один - robotsInterpreter. Он в свою очередь содержит следующие составляющие:

- blocks - классы для блоков, соответствующих элементам диаграммы выполнения (то есть включение мотора, таймер, ожидание цвета, энкодера, сонара, финальный блок и другие);
- d2RobotModel - содержит в себе все необходимые классы для работы с двумерной моделью, сюда также входит и отрисовка 2D-мира, робота и остальные возможности, которые имеются при работе с этой моделью;
- robotCommunication - содержит классы, отвечающие за установку соединения с реальным роботом по usb, по bluetooth, а также классы для осуществления этого соединения в Windows, Linux и Mac системах.
- robotImplementation - классы, описывающие разные реализации модели робота, то есть класс для реального робота, для двумерной модели и nullRobotModelImplementation, а также здесь и абстрактная модель, от которой все остальные наследуются. Здесь же находятся и различные реализации сенсоров и моторов;
- robotParts - классы для сенсоров и моторов;
- interpreter - класс, хранящий контекст вычислений и отвечающий за работу потоков.

Общая структура плагина изображена на рис. 1, где RobotModel зависит от вышеописанной robotImplementation. Теперь более конкретно о структуре, изображенной на этом рисунке. Как я уже сказал, интерпретатор(Interpreter)

выполняет работу с потоками(Thread) и хранит данные о логической модели робота(RobotModel), кроме этого у него есть также таблица блоков(BlocksTable). Каждый блок(Block) запускается в отдельном потоке, в момент выполнения на диаграмме он подсвечивается. В зависимости от типа блока, он обращается к модели робота, которая в свою очередь отправляет запрос соответствующей части робота(RobotPart). Отправка запроса к реальному роботу осуществляется через специальный интерфейс(RobotCommunicationInterface).

Моя же задача заключается в реализации класса D3RobotModel по аналогии с уже имеющимся для двумерной модели D2RobotModel. Я уже упоминал, что двумерная модель содержит также класс и для отрисовки мира, но в связи с тем, что для этих целей используется Gazebo, нет необходимости создавать аналогичный класс для трехмерной модели. Далее я реализовывал UnrealD3RobotImplementation по примеру уже имеющегося класса для двумерной модели. Кроме этого добавлено несколько методов для запуска самого симулятора Gazebo, а для выполнения ros-сервисов и команд был реализован класс GazeboThread, наследуемый от Thread.

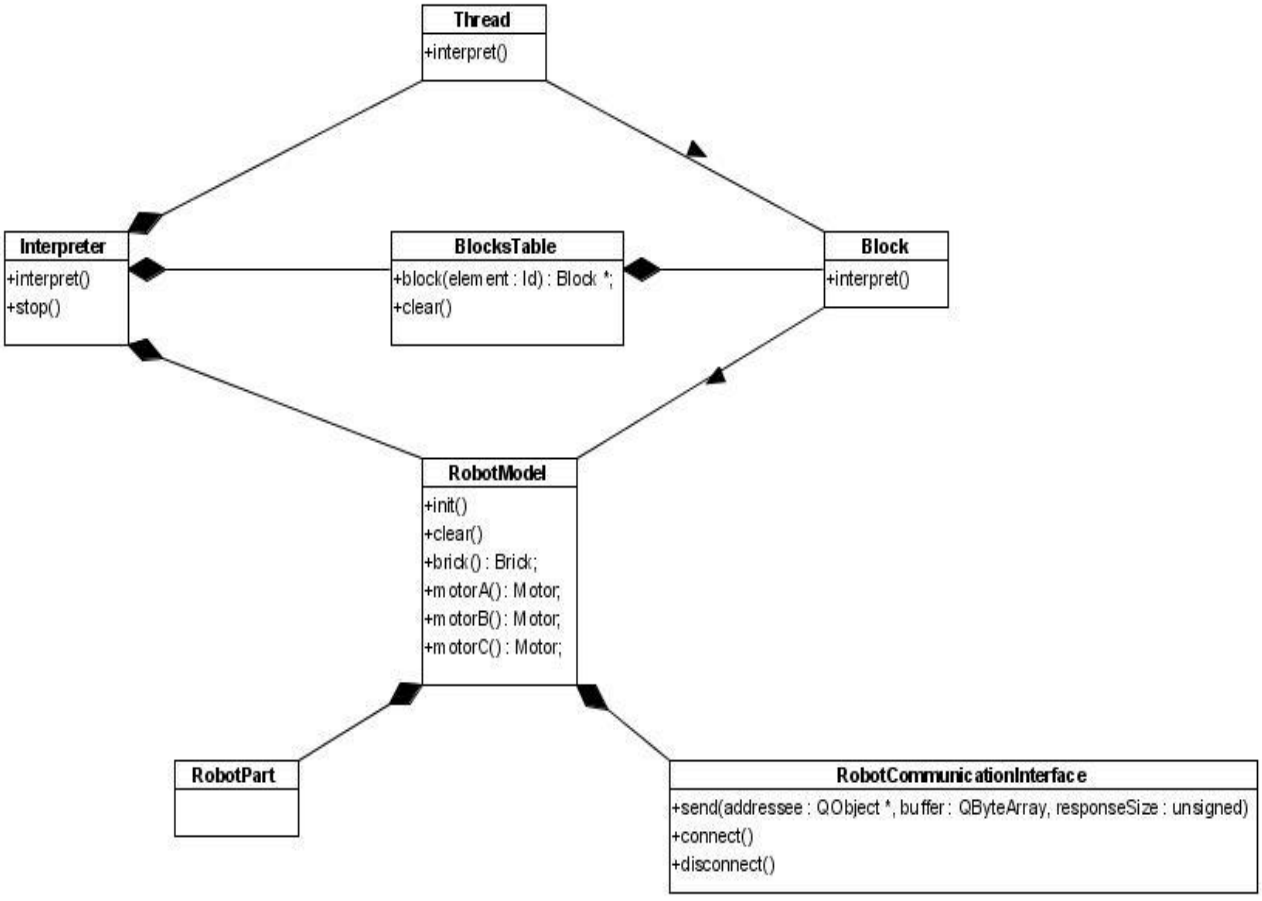

Рис.1 Структура плагина Robots

При реализации была соблюдена основная последовательность выполнения, используемая ранее, которая изображена на рис.2. Сначала метод interpret() вызывается у самого интерпретатора, после чего он вызывается у соответствующего потока. Далее в зависимости от блока идет запрос к нужной

части робота, которая находится либо у реального робота, тогда запрос идет через интерфейс коммуникации, либо у модели, которая выполняет нужное действие и возвращает ответ, который также ступенчато возвращается к блоку. Единственным отличием является то, что в моем случае запрос от конкретной RobotPart отправляется трехмерной модели, а не через bluetooth к реальному роботу. Естественно и ответ идет от трехмерной модели выше.

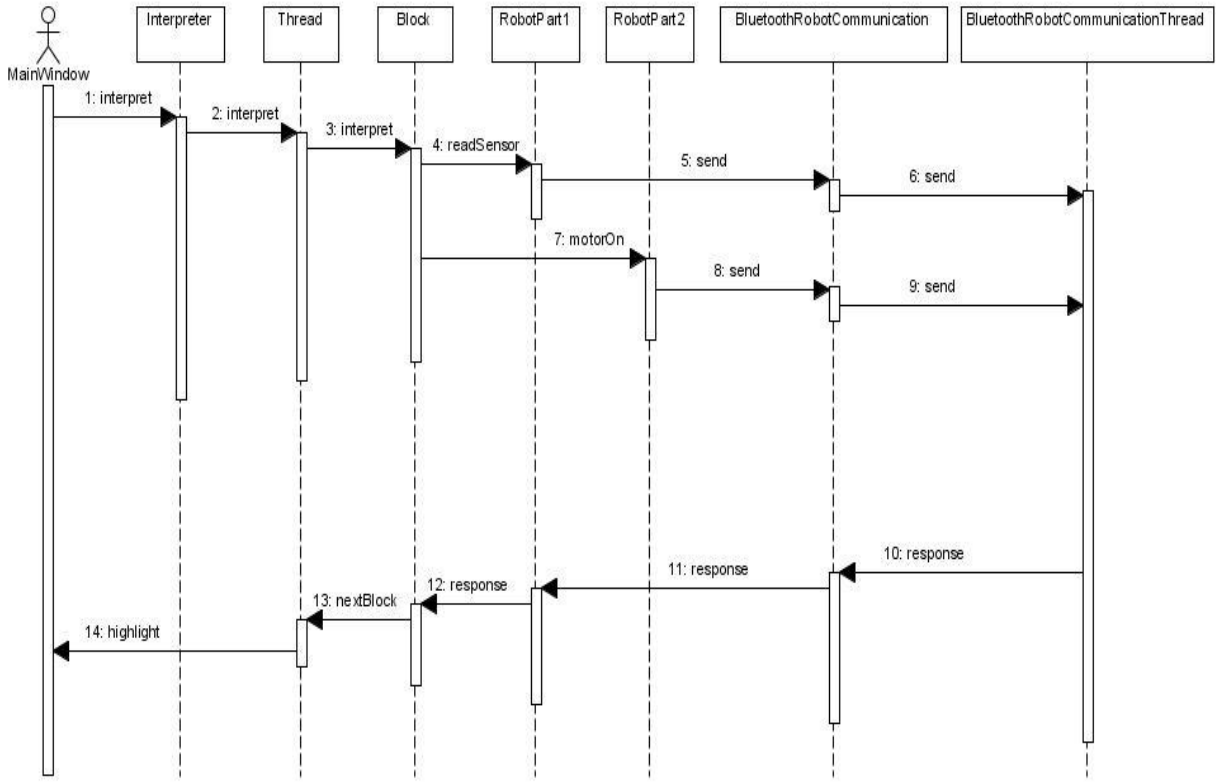

Рис.2 Последовательность выполнения

Апробация

Для демонстрирования корректной работоспособности трехмерной работы в QReal:Robots используется стандартный пример с движением робота, касанием о стену и остановкой. Остановка осуществляется после получения информации от сенсора касания.

Выводы

В результате моей работы был исследован 3D-симулятор Gazebo, который является средством для отображения трехмерного мира. Также была разработана программа для управления роботом, использующая специальные команды и сервисы. И в конце концов это все было внедрено в QReal:Robots. Gazebo используется в качестве трете-стороннего продукта для отображения трехмерной мира и действий в нем, а для выполнения этих действий используются специальные ros-сервисы и ros-команды, которые являются частью сборки Gazebo от ROS-сообщества. На рис.3 изображена диаграмма классов, которая получалась в результате моей работы. Добавлен класс UnrealD3RobotImplementation, D3RobotModel и классы UnrealD3MotorImplementation и

UnrealD3SensorImplementation.

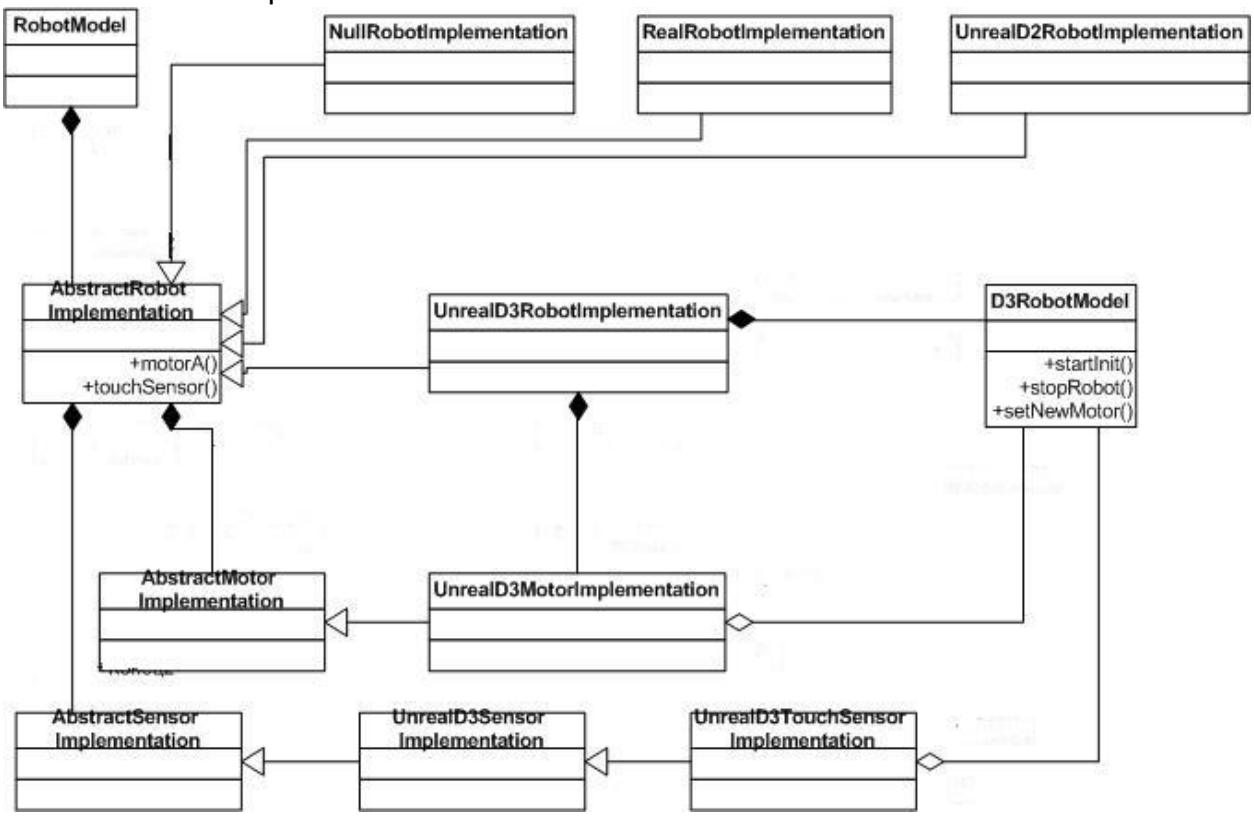

Рис.3 Результат

Перспективы развития

В дальнейшем планируется внедрить трехмерный мир, который сейчас запускается трете-сторонним средством, непосредственно в сам проект QReal:Robots, как это сделано с двумерной моделью. Также лучше использовать библиотеку libgazebo от ROS-сообщества для более конструктивной, а скорее всего и более оптимальной, реализации управления трехмерной моделью. Реализовать работу оставшихся сенсоров.

Материалы

[1] Сайт проекта QReal: http://se.math.spbu.ru/SE/qreal, http://qreal.ru/

[2] 3D-симуляторы: http://en.wikipedia.org/wiki/Robotics_simulator#Open_source_simulators

[3] Сайт Robot Operating System сообщества: http://www.ros.org/wiki/

[4] Общая структура и детали реализации плагина Robots: https://github.com/qreal/qreal/wiki/%D0%9F%D0%BE%D0%B4%D0%B4%D0%B5%D1%80%D0%B6%D0%BA%D0%B0-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%BE%D0%B2-Lego-Mindstorms

[5] Поддержка плагинов в QReal: https://github.com/qreal/qreal/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD%D1%8B

[6] Курсовая работа студентки 244 группы Дерипаска Анны Олеговны “Эмуляция поведения робота в проекте QReal:Robots” от 2011 года: https://github.com/qreal/qreal/wiki/Deripaska_report_244.docx