Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Новожилов Евгений Алексеевич

Интеграция робототехнической ОС (ROS) с кибернетическим контроллером ТРИК

Дипломная работа

Допущена к защите.

Зав. кафедрой:

проф., д. ф.-м. н. А.Н. Терехов

Научный руководитель:

ст. преп. Я.А. Кириленко

Рецензент:

доц., к.т.н. К.В. Кринкин

Санкт-Петербург

2015

SAINT PETERSBURG STATE UNIVERSITY

Software Engineering Chair

Evgenii Novozhilov

Integration of Robot Operating System (ROS) with TRIK cybernetic controller

Graduation Thesis

Admitted for defence.
Head of the chair:
Professor Andrey Terekhov

Scientific supervisor:
Senior lecturer Iakov Kirilenko

Reviewer:
Associate Professor, PhD Kirill Krinkin

Saint-Petersburg
2015

Оглавление

1. Введение

В современном мире робототехника является одной из самых быстроразвивающихся отраслей. Роботы находят применение во многих сферах деятельности человека, таких как медицина, военное дело, космические технологии. Роботы участвуют в спасательных операциях, изучают космические тела солнечной системы, выполняют огромное количество рутинной работы на порядок лучше человека. На волне возрастающих вычислительных мощностей вкупе с развитием машинного обучения, робототехника привлекает все больше новых сторонников.

Вместе с распространением интереса к робототехнике происходит снижение порога доступности к ней самой и современным технологиям, которые в ней используются. Большинство людей уже имеет дома роботов-пылесосов, которые находятся в свободной продаже во множестве магазинов и выполняют свою работу во вполне удовлетворительном качестве. Снижение порога доступности можно отметить как в стоимостном факторе, так и в необходимом уровне образования. Еще в 40-х годах 20 века сложно было представить, когда появились первые роботы, похожие на черепах, что уже меньше, чем через столетие, обычный ребенок сможет собирать робота из составных модулей. Такое положение объясняется тем, что для любого процесса, который хоть как-то будет применим в нелокальных масштабах, должен существовать способ максимально его упростить, тем самым дав возможность обучать людей этому процессу с ранних лет, заложив понимание сферы в юные умы.

В связи с развитием робототехники возрастает глобальный спрос на квалифицированных специалистов в этой области. До недавних времен с робототехникой в нашей стране все обстояло не самым лучшим образом. Сказывалась ситуация 90-х годов 20-го века, огромный приток импорта в страну, потеря конкурентоспособности товарами, произведенными внутри страны. Однако в последнее время со стороны государства проявился интерес

к развитию сектора современных технологий в России. Организация Сколковского института науки и технологий в сотрудничестве с Массачусетским технологическим институтом, появление проектов роботов-помощников, таких как Lexy[1], получающих награды во всемирно известных соревнованиях, говорит о реальном развитии робототехники в России. Параллельно с возрастающим спросом на специалистов в области робототехники возрастает интерес к обучению робототехнике в образовательных учреждениях.

Таким образом, сформировался отдельный класс роботов, предназначенных для образования. Если обратиться к списку образовательных роботов на Википедии[2], то можно заметить, что их существует совсем немного. Более того, можно заметить, что вычислительные мощности образовательных роботов сильно ограничены, в то время как в образовательных учреждениях могут обучать вполне нетривиальным алгоритмам.

Для решения этих недостатков был создан контроллер ТРИК[3].

Помимо развития аппаратных платформ происходит развитие ПО в сфере робототехники. Появляются унифицированные наборы инструментов для разработки ПО для роботов, как когда-то появлялись инструменты для разработки кроссплатформенного ПО. Одним из таких инструментов является ROS – Robot Operating System. ROS является довольно популярным средством и используется во множестве сфер, включая науку и образование. Одной из особенностей этого инструмента является его независимость от аппаратных платформ, что позволяет разрабатывать программы на ROS на любой системе для любых поддерживаемых роботов. И для того, чтобы дать возможность людям использовать такой замечательный инструмент в паре с контроллером ТРИК, была поставлена задача реализовать модуль управления контроллером ТРИК для фрэймворка ROS.

2. Постановка задачи

Таким образом, была поставлена цель реализовать модуль управления контроллером ТРИК для фрэймворка ROS с целью поддержки современных средств разработки в обучении робототехники контроллером ТРИК.

В ходе работы были сформулированы следующие задачи.

1. Изучить существующие реализации подобных адаптеров для популярных роботов или сенсоров.
2. Выбрать технологии и архитектуру решения.
3. Разработать модуль управления контроллером ТРИК из ROS.
4. Произвести тестирование модуля на пробных примерах.
5. Внедрить модуль в систему сборки ПО ТРИК.

3. Обзор решений

3.1. Описание фрэймворка ROS

Создание действительно хорошего программного обеспечения для робота самостоятельно является довольно сложной задачей. В 2007 году в Лаборатории Искусственного Интеллекта Стэндфордского Университета был предложен концепт абстракции интерфейса взаимодействия с роботом по примеру того, как операционная система взаимодействует с физическим оборудованием. В дальнейшем этот набор инструментов сменил двух владельцев, привлек к себе множество внимания, в нем появилась поддержка различных алгоритмов машинного зрения, графических утилит управления и т.д.

С точки зрения функциональности весь фрэймворк ROS можно разделить на 2 составляющие части:

1) пакеты, предоставляющие возможности по взаимодействию с роботами, иными словами – robot middleware;
2) пакеты, реализующие различные алгоритмы современной информатики, использующие данные, производимые пакетами, упомянутыми пунктом ранее.

Сформировавшиеся унифицированные интерфейсы для представления различных типов данных, специфичных для робототехники, позволили независимо существовать и развиваться пакетам из обеих категорий.

На данный момент одними из самых известных библиотек, интегрированных с ROS, являются:

1) библиотека алгоритмов компьютерного зрения OpenCV,
2) библиотека 3D-геометрии PCL.

Также во фрэймворке ROS существует множество различных утилит, предназначенных для отладки, визуализации и прочих неотъемлемых частей процесса разработки.

С архитектурной точки зрения ROS представляет собой гетерогенную распределенную вычислительную систему, состоящую из узлов, в терминах ROS называющиеся nodes. Узел представляет собой запущенный процесс на некоторой платформе, подключающийся к главному узлу, сообщая тем самым о своем существовании. Для коммуникации между узлами используется специфичные ROS-интерфейсы «темы» (topics). Узлы могут подписываться на темы и публиковать в них какую-то информацию, представляющую собой сообщения (messages).

С технической точки общение между узлами реализовано с помощью протокола XML-RPC поверх HTTP. Каждый узел, кроме главного, для запуска должен знать URI главного узла. Реализация программных интерфейсов ROS существует для языков Python, C++, Lisp.

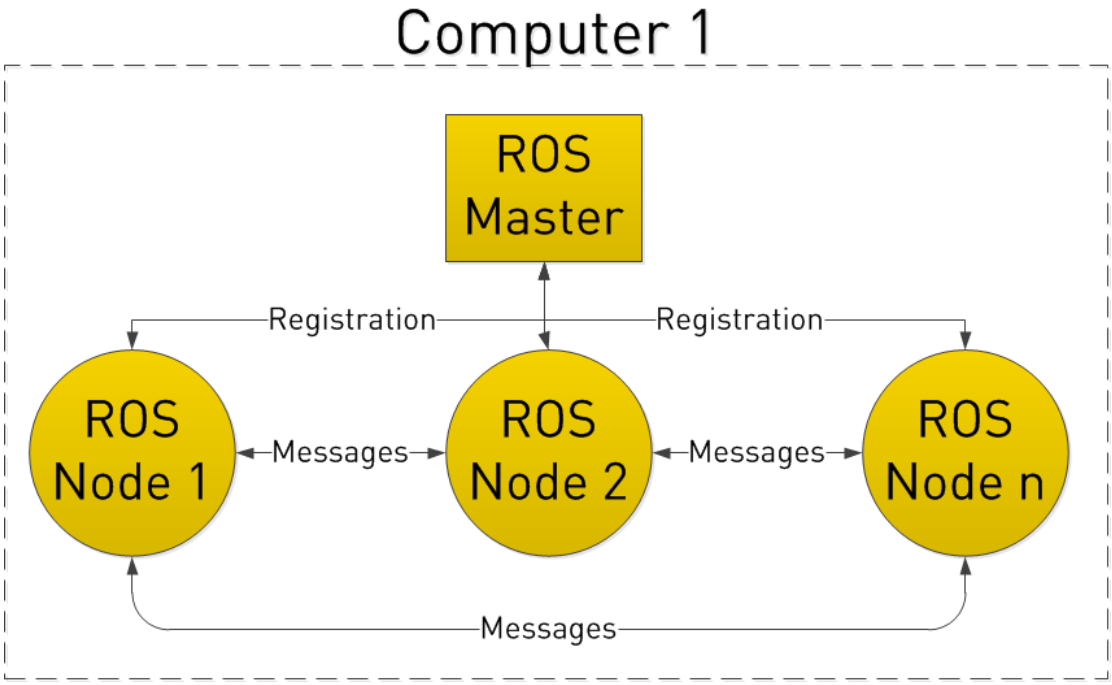

Рис. 1. Пример запущенной сети ROS

3.2. Архитектура решений

3.2.1. Poll и pull реализации

Сенсоры роботов являются неотъемлемой частью робототехники, представляя собой неисчерпаемый источник данных. Большая часть взаимодействия с роботами производится на основе данных, полученных с сенсоров. Существуют 2 основные идеологии получения данных с сенсоров:

1) polling,
2) pull coding.

В первом случае данные с сенсоров опрашиваются с частотой f, и в случае обращения к данным в период $1/f$ с момента последнего опроса сенсора данные берутся из кэша. Данный подход обладает некоторыми недостатками, например:

Постоянный поллинг датчиков несет в себе высокую вычислительную нагрузку;

Существует «мертвая зона» показаний, заключающаяся в том, что в некоторый момент данные могут стать неактуальными и будет принято неверное решение.

Во втором случае обращение к сенсору происходит по желанию стороны, пытающейся получить значение с сенсора. Этот подход позволяет тратить вычислительные ресурсы намного экономнее, обеспечивает потребителя данных всех актуальной информацией. Однако он требует излишней бизнес логики потребителя, которая будет отвечать за опрос pull-источника данных.

Рассматривая эти 2 модели поведения в случае фрэймворка ROS, стоит учитывать тот факт, что в большей своей части ROS рассчитан на применение в производстве, где вычислительные мощности для одного робота представляют собой несколько юнитов серверов. Понимая возможности вычислительных мощностей, и оценивая дублирование бизнес-логики во множествах клиентов, опрашивающих сервер, в роли которого здесь выступает узел, владеющий сенсором, сложилось так, что в большинстве

реализаций сенсоры работают в режиме поллинга. Однако в случае работы с роботами, обладающими слабыми вычислительными ресурсами это ведет к тому, что данные посылаются с сенсоров с довольно низкой частотой.

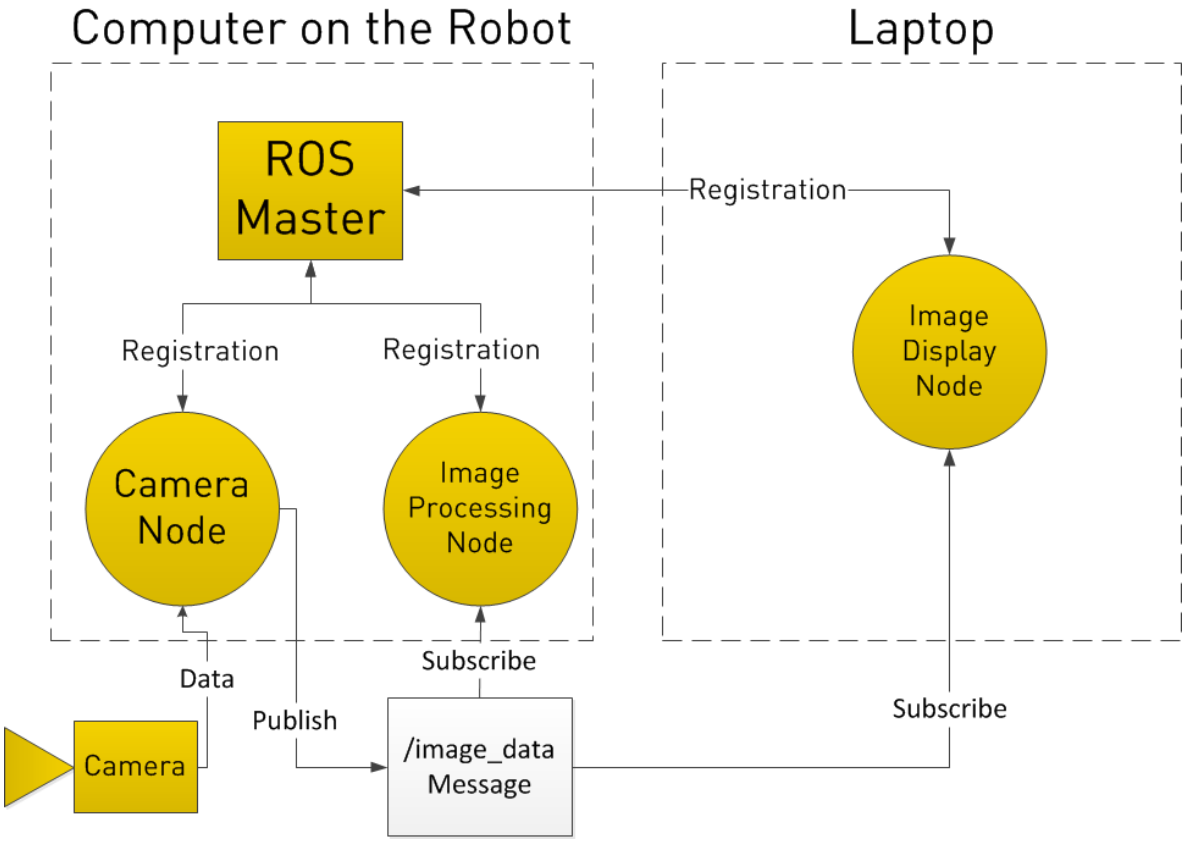

Рис. 2. Пример poll-взаимодействия

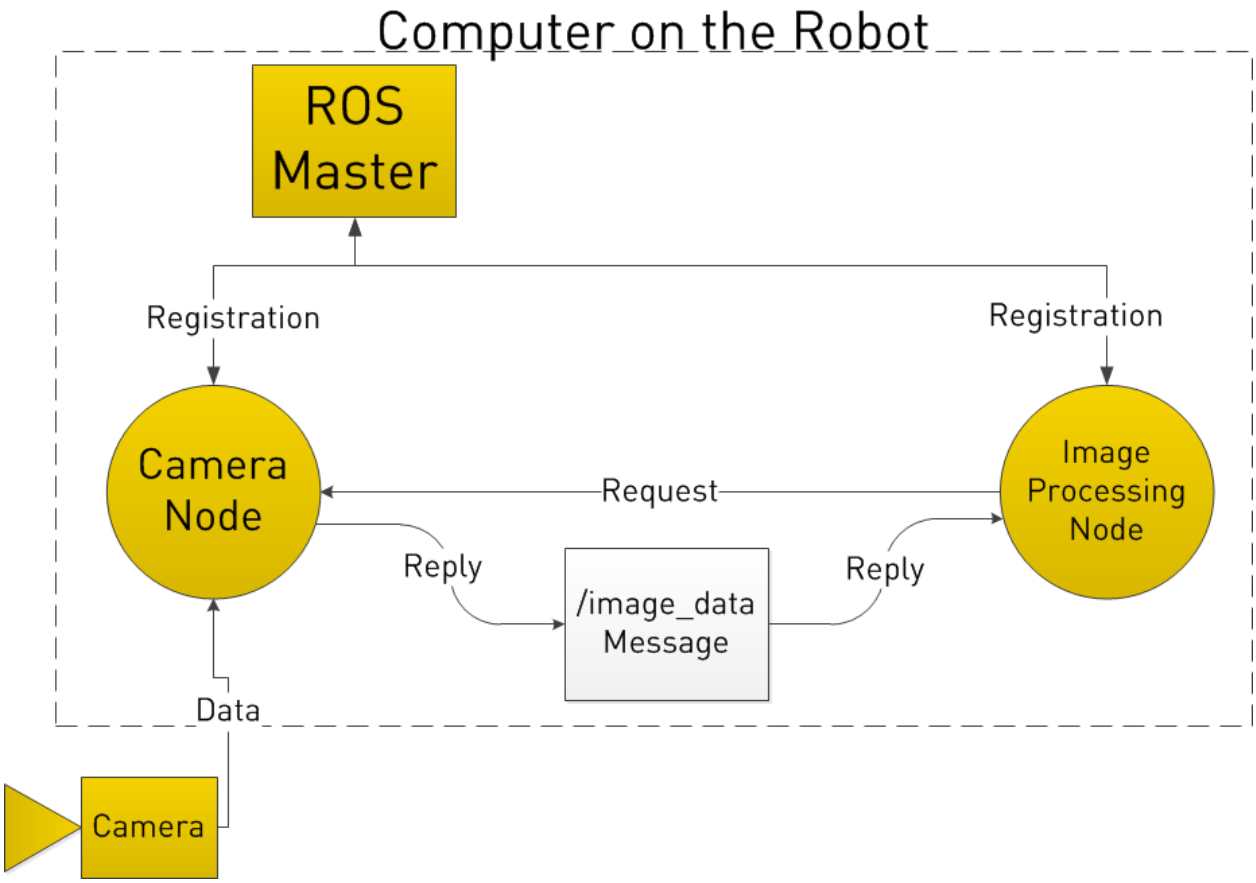

Рис. 3. Пример pull-взаимодействия

3.2.2. Примеры существующих адаптеров

В рамках данной дипломной работы был произведен обзор реализаций адаптеров для ROS следующих датчиков или роботов.

1. Адаптер для инфракрасного сенсора расстояния Sharp для платформы Arbotix[4].
2. Адаптер для акселерометра Bosch Sensortec BMA180[5].
3. Адаптер для роботов серии Lego NXT[6].
4. Адаптер для робота Clearpath Robotics Husky[7].

Основное внимание в обзоре уделялось интерфейсам, которые предоставляют упомянутые адаптеры, с целью разработать адаптер, максимально приближенный к общепринятым стандартам.

3.2.2.1. **Адаптер для инфракрасного сенсора расстояния Sharp для платформы Arbotix**

Данный адаптер представляет собой ros-to-ros обертку над имеющейся реалиацией у контроллера arbotix[8]. Его работа заключает в подписи на внутренние сообщения, и конвертацию и отправку в ROS-like стиле по получению нового. Данный адаптер реализован на языке Python. Этот адаптер интересен тем, что в контроллере ТРИК используется подобный данному инфракрасный сенсор Sharp 2Y0A21. Интерфейс адаптера представляет собой одну опубликованную тему ir_range, в которую посылаются сообщения sensor_msgs/Range, описывающие текущие данные полученные с сенсора.

3.2.2.2. **Адаптер для акселерометра Bosch Sensortec BMA180**

Реализация адаптера для данного акселерометра представляет собой нативную взаимодействие с устройством через шину SPI[9], используя драйвер, прилагающийся к этому адаптеру. Адаптер работает в режиме поллинга с частотой по умолчанию равной 100 Гц. Параметры запуска адаптера позволяют конфигурировать частоту поллинга, максимальное измеряемое ускорение акселерометром и возможность калибровки при инициализации. Интерфейс адаптера содержит в себе одну опубликованную тему bma180, в которую посылаются нестандартные сообщения типа bma180/bma180meas. Адаптер реализован на языке C++.

3.2.2.3. **Адаптер для роботов серии Lego NXT**

Этот адаптер добавляет в ROS поддержку одной из самых популярных образовательных серий роботов Lego NXT. Адаптер реализован на языке Python, с использованием библиотеки удаленного взаимодействия nxt-python[10]. Адаптер представляет собой один узел сети ROS, публикующий сразу несколько тем по различным сенсорам, и подписывающийся на темы, для управления моторами. Большинство тем имеют нестандартные интерфейсы, описанные в модуле nxt_msgs. Данный адаптер также работает по модели поллинга, однако не имеет значения частоты обновления по умолчанию. В примерах, предоставленных разработчиками адаптера, значение варьируется от 5 до 20 Гц.

3.2.2.4. **Адаптер для робота Clearpath Robotics Husky**

Компания Clearpath Robotics является одним из активных участников сообщества ROS. Компания производит различные типы роботов, позиционируя ROS, как один из важных интерфейсов общения с роботом. Адаптер представляет собой набор различных пакетов для отдельных сенсоров. Данные с сенсоров обрабатываются по модели поллинга, однако частота поллинга неизвестна. Сообщения с большинства сенсоров обладают типами из пакета sensor_msgs. Стоит отметить, что Husky является промышленным роботом, обладающий весом в 50 кг, точные вычислительные характеристики которого неизвестны.

3.3. Описание репозитория пакетов meta-ros для OpenEmbedded

Основное ядро контроллера ТРИК работает под управлением ОС на основе ядра Linux. Для сборки ОС используется фрэймворк OpenEmbedded[11]. Сборка производится на основе рецептов, которые располагаются в различных репозиториях. Сборка ROS была портирована на данный фрэймворк представителями BMW Car IT GmbH. Рецепты сборки различных компонентов ROS расположены в репозитории meta-ros [12] . Репозиторий подключается к системе сборки bitbake в качестве одного из уровней (layers). В качестве основы для портирования была выбрана версия ROS Hydro Medusa. Эта версия ROS в свое время была рассчитана на использование с ОС Ubuntu 12.04. В связи с этим в данной работе все тестирование и эксплуатация будет производиться с использованием ОС Ubuntu 12.04. Основным пакетом, который добавляет компоненты ROS на целевую ОС является roslaunch.

3.4. Подходы к реализации модуля управления контроллера на основе ROS

В ресурсах, описывающих идеи взаимодействия узлов ROS, часто упоминается, что узлами сети являются сенсоры, актуаторы и так далее. Однако в результате обзора можно заметить, что для более простого робота NXT в качестве узла рассматривается сам робот полностью, а не каждый из сенсоров, установленных на нем. Это обусловлено недостатком вычислительных мощностей рассмотренных роботов. Как минимум, запуск каждого отдельного узла несет в себе дополнительные затраты на поднятие сервера для взаимодействия узлов. Теоретически, имея программные возможности удаленного взаимодействия с роботом, можно реализовать узел ROS на любом удаленном от него устройстве. Но в случае с ТРИК такой подход имеет недостатки в удобстве реализации, а также и возможности использования, в связи с слишком большими затратами и задержками для удаленной коммуникации. Используя все вышеизложенное, было принято решение производить реализацию с учетом следующих требований.

1. Реализация модуля управления должна быть нативной на самом контроллере ТРИК.
2. В качестве программного интерфейса к контроллеру ТРИК использовать библиотеку trikControl из пакета trik-runtime.
3. Узел ROS должен представлять собой весь контроллер ТРИК.
4. Узел должен опубликовывать темы на основе данных с сенсоров.
5. Узел должен подписываться на темы соответствующие актуаторам.
6. Данные с сенсоров должны опубликовываться по модели поллинга.

4. Реализация

4.1. Система сборки

В самом начале процесса разработки необходимо корректно сконфигурировать систему сборки проекта, чтобы можно было без неудобств производить эксперименты с реализацией. Весь процесс построения можно разделить на 2 части:

1. Построение самого модуля с использованием фрэймворка ROS;
2. Кросс-компиляция под целевую платформу, которой является ТРИК.

За первую часть отвечает утилита catkin из набора ROS. Catkin является оберткой над CMake, реализующей специфичные ROS-требования, на языке Python. Для построения проекта используется декларативная спецификация из файла CMakeLists.txt.

Вторая задача может быть решена несколькими способами:

1) С помощью утилиты bitbake из фрэймворка OpenEmbedded, которая предназначена для сборки пакетов на целевую платформу
2) С помощью SDK для контроллера ТРИК.

Однако использование SDK в текущем случае является невозможным, в связи с тем, что на данный момент в нем отсутствует корректная поддержка интерпретатора языка Python. Таким образом, построение под целевую платформу возможно только используя утилиту bitbake. В рамках работы был создан «рецепт» пакета trik-ros для утилиты bitbake, который доступен в репозитории[13] и в дальнейшем будет перемещен в meta-trik.

Использование bitbake для построения catkin-пакетов не представляет собой особых сложностей, ключевым моментом этого процесса является необходимость унаследования рецепта от bitbake класса catkin, после чего catkin сам построит все файлы, по конфигурации из CMakeLists.txt.

4.2. Исследование производительности утилит фреймворка ROS

Одной из самых важных утилит фрэймворка ROS является roslaunch. Данная утилита позволяет запускать узлы ROS с параметрами, определенными в launch-файлах. Как и большинство прочих утилит из фрэймворка, она реализована на языке Python. Ее работа заключается в нахождении ROS-пакета и/или launch файла, специфицированного пользователем, в каталоге, куда установлен фрэймворк ROS. В традиционных системах, на которые ориентирован ROS, целевым каталогом установки является /opt/ros, но при портировании ROS на OpenEmbedded, разработчики из BMW CarIt GmbH решили соответствовать идеям FHS, и изменили каталог установки на /usr.[14] Перемещение ROS-файлов из /usr в /opt/ros позволяет ускорить производительность в 2-3 раза, уменьшив время на запуск пакетов с нескольких минут до одной. В рамках этой работы использовался срез ветки master репозитория meta-ros по коммиту f7d260c7388135604c7aa7a8da1472b59ebdcda7, в котором эта проблема еще не была исправлена.

4.3. Доработка системы сборки пакета trik-runtime

Пакет trik-ros стал первым отдельным пакетом с зависимостью на пакет trik-runtime, как в процессе работы, так и при сборке. Однако эти зависимости не могли быть удовлетворены просто из-за отсутствия файлов, эти зависимости разрешающих. Утилита bitbake предоставляет удобный механизм для решения подобных проблем, устанавливая dev-версию пакета в sysroot. Для этого достаточно скопировать необходимые заголовочные файлы в ${D}${includedir} и библиотеки в ${D}${libdir}, после чего добавить список файлов к переменной FILES_${PN}-dev. Файлы пакета trik-runtime до этого устанавливались в /home/root/trik, тем самым не давая возможности bitbake сделать их доступными через sysroot. После внесения описанных изменений в bitbake-класс trik-runtime зависимости процесса построения стали разрешены и стало возможным собирать пакет trik-ros.

Однако стоит отметить, что в силу некоторых причин библиотеки пакета trik-runtime собираются с флагом линковщика “–rpath .”, что создает некоторые трудности при сборке пакета trik-ros, а также не дает возможности полной миграции библиотек пакета trik-runtime в /usr/lib. Данный флаг необходимо убрать из процесса построения trik-runtime, после чего окончательно перенести библиотеки в /usr/lib.

4.4. Архитектура модуля управления контроллером ТРИК

Итоговая реализация модуля управления представляет собой один узел ROS, который при инициализации, используя интерфейс BrickFactory и стандартные файлы конфигурации system-config.xml и model-config.xml, получает сконструированный объект типа BrickInterface. Используя этот объект контроллера, модуль производит обнаружение сенсоров и прочих объектов, ассоциированных с контроллером. На основе обнаруженных объектов модуль создает темы:

1) /sensor_{PORT_NAME} для сенсоров, подключенных к портам. Тип сообщений в этих темах – std_msgs/Int32
2) /vsensor_(accelerometer|gyroscope) для акселерометра и гироскопа. Тип сообщений в этих темах – geometry_msgs/Vector3

Также модуль подписывается на темы:

/motor_{PORT_NAME} для моторов, подключенным к портам. Ожидаемый тип сообщений в этих темах std_msgs/Int32

После инициализации все компонентов модуль начинает опрашивать все обнаруженные сенсоры с частотой 100 Гц и публиковать данные с них в соответствующие темы.

4.5. Тестирование

Корректность показаний опубликованных в темах, а также взаимодействие с моторами было проверено с помощью утилиты rostopic на удаленном компьютере под управлением Ubuntu 12.04

4.6. Апробация

В качестве проверки возможности использования реализованного адаптера, была написана программа с помощью фрэймворка ROS на языке Python, описывающая движение вдоль предмета на установленном расстоянии. Эксперимент был проверен в двух конфигурациях:

Master-узел, а также сама программа была запущена на удаленном компьютере, когда адаптер работал на контроллере ТРИК;

Вся ROS-инфраструктура и программа были запущены на контроллере ТРИК.

В обоих случаях эксперимент прошел удачно, результат работы программы можно найти на портале Youtube[15].

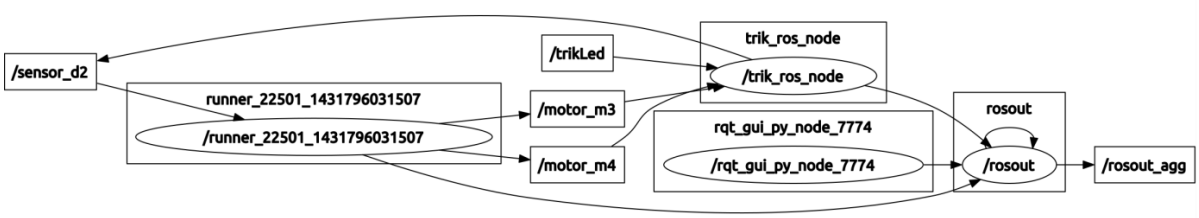

Рис. 4. Пример ROS-сети с запущенной программой.

5. Заключение

В рамках данной дипломной работы были получены следующие результаты.

1. Реализован модуль управления контроллером ТРИК с помощью фрэймворка ROS.
2. Изучены реализации модулей управления для аналогичных контроллеров и датчиков.
3. Выявлены проблемы с производительностью ROS в рамках использованного среза исходного кода.
4. Исправлены недочеты сборки пакета trik-runtime.
5. Произведена демонстрация модуля управления контроллером на базовых моделях.
6. Модуль интегрирован в систему сборки ПО контроллера ТРИК и находится в свободном доступе.

6. Список литературы

[1] Сколково: Lexy стал победителем трека «Инновации» российского финала Imagine Cup 2015. // http://sk.ru/news/b/articles/archive/2015/04/19/lexy-stal-pobeditelem-treka-innovacii-rossiyskogo-finala-imagine-cup-2015.aspx

[2] Википедия: список производителей роботов для образования. // http://en.wikipedia.org/wiki/Educational_robotics#Educational_robot_manufacturers_and_projects

[3] Блог ТРИК. // http://blog.trikset.com

[4] ROS-пакет для сенсоров роботов серии arbotix // http://wiki.ros.org/arbotix_sensors

[5] ROS-пакет для акселерометра Bosch Sensortec // http://wiki.ros.org/bma180

[6] ROS-пакет для роботов Lego NXT // http://wiki.ros.org/nxt_ros

[7] ROS-пакет для робота Clearpath Robotics Husky // http://wiki.ros.org/Robots/Husky

[8] Сайт разработчика роботов arbotix // http://www.vanadiumlabs.com/arbotix.html

[9] Описание шины SPI // https://ru.wikipedia.org/wiki/Serial_Peripheral_Interface

[10] Модуль для взаимодействия с роботами Lego NXT для Python // https://code.google.com/p/nxt-python/

[11] Описание фрэймворка OpenEmbedded // http://www.openembedded.org/wiki/Main_Page

[12] Репозиторий meta-ros // https://github.com/bmwcarit/meta-ros

[13] Репозиторий с рецептами модуля ROS для ТРИК // https://github.com/ujohnny/recipes

[14] Описание проблемы производительности утилиты фрэйворка ROS https://github.com/bmwcarit/meta-ros/issues/214

[15] Адаптер ROS для ТРИК в действии // https://youtu.be/TdqLWODNLbQ