СИСТЕМА УПРАВЛЕНИЯ ROS НА ПРИМЕРЕ РОБОТА TURTLEBOT

© Белоусов А.А.[♣], Тихонов И.Н.[♦], Лемех А.В.[♥], Третьяков В.А.

Уральский Федеральный Университет, г. Екатеринбург

Synapticon GmbH, Германия, г. Штутгарт

Современные вычислительные устройства, которыми оснащаются роботы, позволяют обрабатывать большое количество информации за малые промежутки времени. Это привело к тому, что, обходясь малым количеством датчиков и исполняемых устройств, роботы стали иметь большие функциональные возможности, чья реализация выпала на долю программного обеспечения, которым оснащены устройства управления.

В данной статье речь пойдёт об одной из реализаций системы управления роботами на уровне программного обеспечения, именуемой *ROS* (*Robot Operating System*). Она включает в себя драйвера к различным электронным компонентам, библиотеки, визуализаторы, а так же предоставляет систему управления процессами и т.п. ROS имеет открытый исходный код и пользуется большой популярностью в среде зарубежных разработчиков приложений для роботов.

Ключевые слова: мехатроника, робототехника, системы управления.

Особенности системы

Для начала необходимо отметить, какие главные особенности выделяют в ROS её разработчики. В [1] названы следующие пункты:

– Пиринговая сеть (Peer-to-peer);

– Модульность (Tools-based);

♣ Студент кафедры Электронного машиностроения Уральского федерального университета (специальность «Мехатроника и робототехника»).

♦ Заведующий кафедрой Электронного машиностроения Уральского федерального университета, кандидат технических наук, доцент.

♥ Старший преподаватель кафедры Подъёмно-транспортных машин и роботов Уральского федерального университета, магистр техники и технологии.

– Многоязычие (Multi-lingual);
– Эргономичность (Thin);
– Свободное распространение и открытые исходные коды (Free and Open-Source).

Peer-to-peer соединение позволяет использующимся в работе робота компьютерам напрямую подключаться друг к другу, что обеспечивает эргономичный расход трафика в условиях относительно медленного беспроводного соединения.

Пункт Tools-based говорит о том, что деление структуры системы на множество небольших модулей, где каждый модуль отвечает за выполнение своей определённой задачи, предпочтительней, чем наличие одной громоздкой программы.

Multi-lingual: ROS не отдаёт предпочтение какому-то одному языку программирования и поддерживает сразу несколько: C++, Python, Octave, LISP.

Под Thin понимается, что все необходимые программные модули и драйвера эргономичней хранить в библиотеках, дабы иметь возможность пользоваться ими многократно и не увеличивать объём программного кода.

Free and Open-Source означает, что ROS бесплатна и имеет открытый исходный код.

Описание работы системы

В [1] выделены 4 основных сущности, которые обеспечивают работу ROS: это узлы (Nodes), сообщения (Messages), заголовки (Topics), сервисы (Services).

Под узлами (другое название – программные модули (software modules)) понимаются вычислительные процессы, представленные в виде исполняемых файлов. Между собой программные модули обмениваются сообщениями – строго типизированными структурами данных, состоящих из переменных различных типов, их массивов или констант. Результаты своих вычислений узлы публикуют в заголовки. Если какому-то программному модулю нужно использовать информацию из этого заголовка, то он на него подписывается.

Например, один программный модуль получает и обрабатывает сигнал с датчиков, после чего публикует его в определённый заголовок в виде каких-то значений. Второй программный модуль, отвечающий за построение траектории движения, отслеживает изменение этих значений и на их основе корректирует траекторию движения.

Таким образом на базе заголовков организуется система обмена сообщениями publish-subscribe (подписка-публикация).

Помимо неё в ROS используется запросно ответная (request-response) система обмена сообщениями, объектами которой называются сервисами.

Формулировка задачи

Возможности ROS можно пронаблюдать на примере задачи, исполняемой роботом Turtlebot, спроектированным организацией Willow Garage.

Рис. 1. Turtlebot

На рисунке 2 представлена его схема. Этот робот состоит из следующих компонентов:

– Мобильная платформа Kobuki, разработанная Yujin Robots, обладающая шасси, контроллером, рядом встроенных сенсоров (датчик касания, датчик отрыва колеса от земли), несколькими машинно-

машинными интерфейсами и портами для питания дополнительных электронных устройств;
- Сенсор Kinect;
- Ноутбук с предустановленной операционной системой на ядре Linux (желательно Ubuntu);
- Каркас, на котором располагается ноутбук и монтируется Kinect.

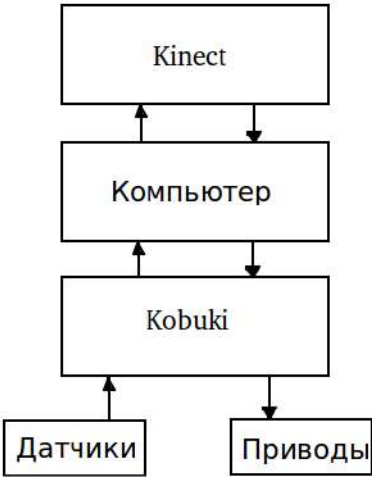

Рис. 2. Схема робота

Задачу для этого робота можно сформулировать следующим образом: Определение роботом своей позиции в помещении и последующее его перемещение в заданную пользователем точку.

Уточнение возможностей робота

Конструкция и программное обеспечение робота позволяет выполнить поставленную задачу. Чтобы в этом убедиться обратимся к мировому опыту. К примеру, из [2] известно, что шасси колёсных роботов классифицируются в зависимости от конструкции колёс, их расположения, количества и роли. В нашем случае конструкция мобильной платформы имеет два активных (ведущих) колеса и два пассивных (ведомых), поворотных, для опоры (см. рисунок 3). Таким образом, траектория движения робота зависит от разно-

сти угловых скоростей вращения обоих колёс (если скорость одинакова и колёса вращаются в одну сторону, то движение будет происходить по прямой, иначе мобильная платформа будет осуществлять поворот). Это позволяет роботу маневрировать в пределах очень узких пространств, ибо поворот на месте робот может осуществлять без каких-либо перемещений. Для расчёта траектории движении такого робота достаточно учитывать показания одометров (для измерения скорости) и гироскопа (для измерения угла поворота), установленных в мобильной платформе.

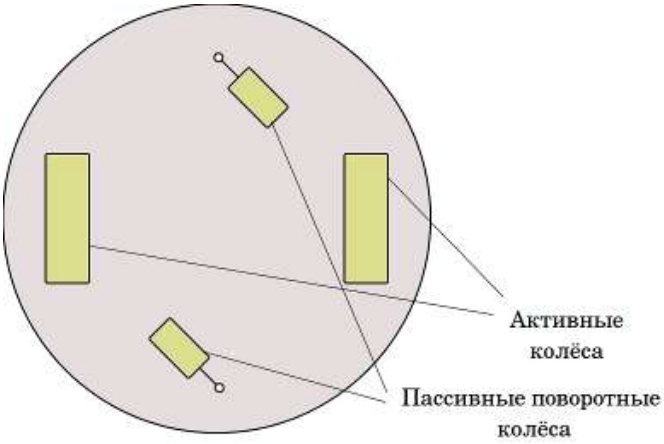

Рис. 3. Расположение колёс на мобильной платформе

В [3] описаны методы, позволяющие роботу определять своё положение внутри известного ему помещения. Информация о помещении собирается при помощи показаний, поступающих с инфракрасного дальномера (в случае с turtlebot'ом – при помощи kinect'а), одометров и гироскопа, затем обрабатывается компьютером робота и представляется в виде карты пространства, на котором отмечены области, по которым робот может свободно перемещаться и элементы интерьера, которые робот преодолеть не в состоянии (препятствия).

На рисунке 4 представлена карта, построенная turtlebot'ом. Чёрная точка в центре карты – turtlebot; тёмносерым цветом отмечены области, которые

неизвестны роботу; светлосерым – области, по которому робот может свободно перемещаться; чёрные контуры робот запомнил как препятствия, цветные области робот считает пока временным препятствием и отслеживает их в режиме реального времени.

Определение роботом своего положения на карте называется локализацией и подробно описывается в [3]. Локализация определяется при помощи методами теории вероятности.

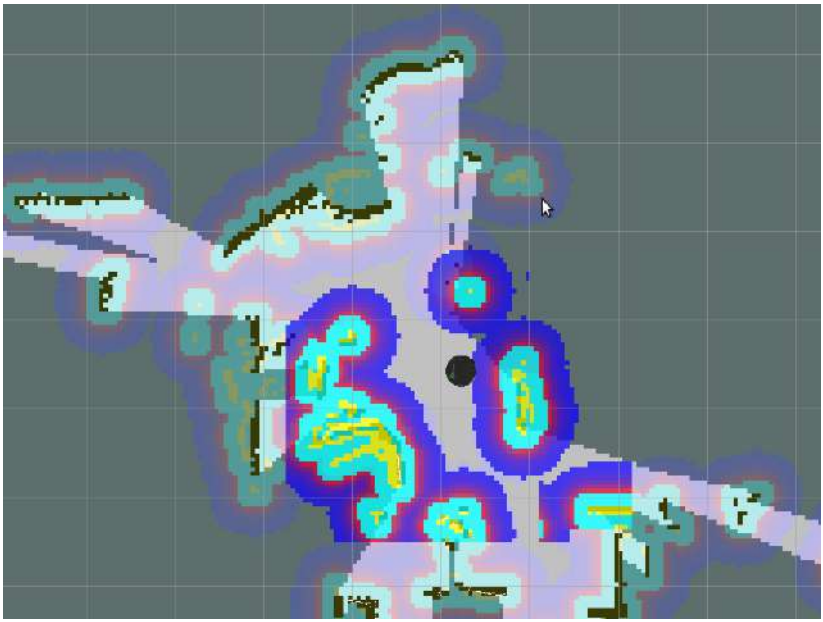

Рис. 4. Карта области помещения

На рисунках 5-8 проиллюстрирован примерный принцип определения положения робота на карте. Предположим, что некий робот двигается по прямой вдоль оси X и может измерять расстояние по оси Y до некоего препятствия с тремя одинаковыми по размерам выступами. В первоначальный момент времени (позиция A, рисунок 5) считается, что робот может находиться практически в любой позиции с равной вероятностью (вероятные позиции робота отмечены на оси P(x)).

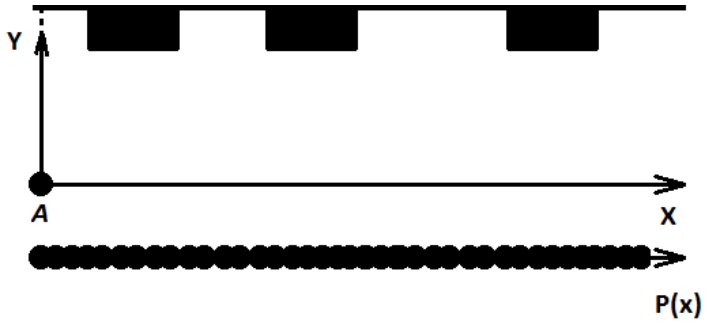

Рис. 5. Позиция А

Переместившись на небольшое расстояние за некоторый промежуток времени (позиция Б, рисунок 6) количество точек, в которых робот вероятно сейчас находится, существенно сократилось.

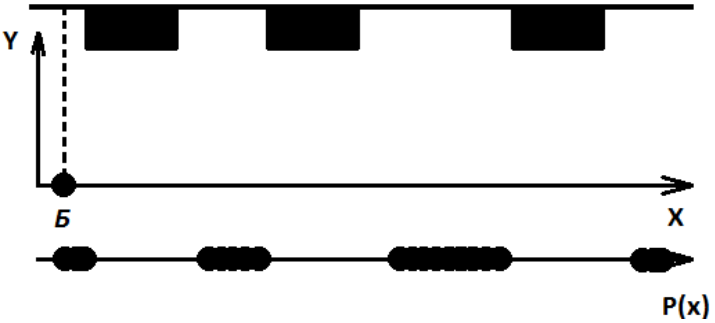

Рис. 6. Позиция Б

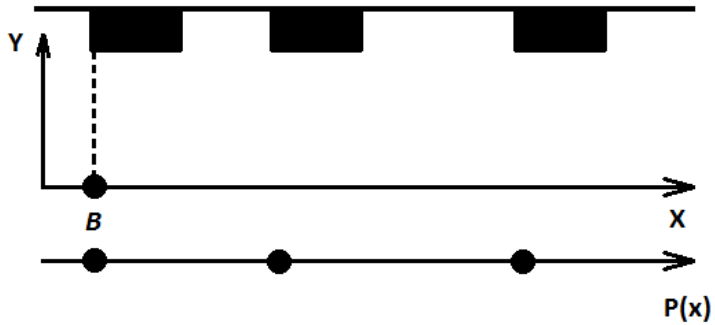

Рис. 7. Позиция В

Переместившись в позицию В (рисунок 7) робот обнаружил выступ и осталось только три вероятных позиции, в которых он мог бы находиться, т.е. это говорит о том, что было лишь три возможности проехать известное расстояние вдоль препятствия и наткнуться на выступ на карте.

Только доехав до позиции Г (рисунок 8) положение текущей и вероятной позиций совпало (вероятная позиция 2 отбрасывается, ибо было преодолено слишком маленькое расстояние между выступами, вероятная позиция 3 отбрасывается, т.к. на её пути выступов не ожидается).

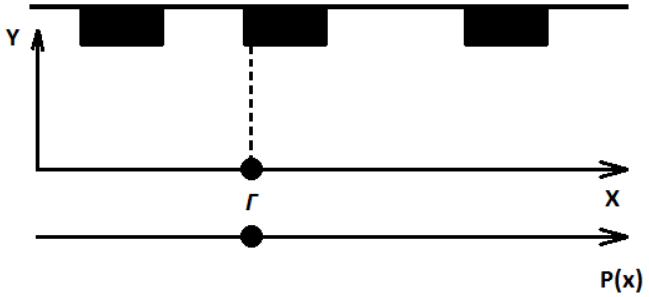

Рис. 8. Позиция Г

Таким образом, отталкиваясь от уже известной информации, робот может корректировать свою локализацию.

Решение задачи в ROS

Для выполнения поставленной задачи достаточно в консоли операционной системы (именуемой терминалом) ввести следующие команды [4; 5]:

roslaunch turtlebot_bringup minimal.launch

roslaunch turtlebot_navigation gmapping_demo.launch

roslaunch turtlebot_rviz_launchers view_navigation.launch

roslaunch turtlebot_teleop keyboard_teleop.launch

Roslaunch является командой запуска программных модулей. Первый аргумент этой команды (начинается на «turtlebot_») является пакетом (package), внутри которого roslaunch ищет файл, указанный во втором аргументе (заканчивается на «.launch»). Эти файлы содержат необходимые инструкции, а так же пути поиска других необходимых для работы файлов.

Результатом выполнения первой команды является перманентный опрос состояния устройств мобильной платформы (состояние аккумулятора, информация с датчиков и т.д.).

Выполнение второй команды организует считывание информации с сенсора Kinect, а так же формирует процессы построения карты помещения (их успешное выполнение требует данные с датчиков мобильной платформы, предоставляемые в процессе выполнения первой команды).

В третьем случае запускается приложение, позволяющее визуализировать процессы выполнения второй команды: отображается положение робота и расположение препятствий перед ним в режиме реального времени (см. рисунок 4).

Четвёртая команда позволяет производить считывание данных с устройства ввода (в данном случае – с клавиатуры) с последующим формированием задающего сигнала исполнительным устройствам (производится управление мобильной платформой).

Таким образом, управляя роботом можно увеличивать размеры создаваемой карты.

В дальнейшем, после сохранения карты и перезапуска приложения визуализации пользователю достаточно будет указать нужную точку на карте и робот сам переместится туда.

Выводы

Будучи системой с открытым исходным кодом, не привязанной к электронным компонентам какого-либо конкретного производителя, ROS позволяет организовывать сложные программные процессы в любых робототехнических системах, практически не влияя как на выбор компонентной базы роботов, так и на инструменты разработки. Данная система может оказаться полезной в учебных целях, позволяя ученикам на практике знакомиться с системой управления роботами и внедрять в них основанные на ROS новые программные алгоритмы, при этом не влияя на работоспособность уже имеющихся.

Список литературы:

1. Quigley M. ROS: an open-source Robot Operating System [Электронный ресурс]. – Режим доступа: http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf (29 декабря 2014).

2. Campion G., Chung W. Wheeled Robots// Springer Handbook of Robotics / B. Siciliano, O. Khatib (Eds.). – Springer Berlin Heidelberg, 2008. – P. 391-410.

3. Thrun S., Burgard W. and Fox D. Probabilistic Robotics. – MIT Press, Cambridge, MA, 2005.

4. SLAM Map Building with TurtleBot (29 декабря 2014).

5. Keyboard Teleop (29 декабря 2014).

ПРЕОБРАЗОВАНИЕ СТРУКТУРЫ ПРОГРАММЫ НА ЭТАПЕ ПРОЕКТИРОВАНИЯ

© Борисенков Д.С.[♣]

Филиал Национального исследовательского университета «МЭИ», г. Смоленск

В работе рассматривается преобразования структуры программного обеспечения (ПО) на стадии проектирования. Рассмотрение происходит в рамках функциональной парадигмы программирования с использованием расширения нотации UML для функционального анализа и проектирования [1].

Ключевые слова: функциональное программирование, проектирование, рефакторинг, UML.

Рефакторингом называется модификация исходных текстов программы, не приводящая к изменению ее поведения [2]. Обычно рефакторинг применяется для улучшения читаемости и модульности программы а так же для уменьшения ее сложности. При этом существуют рефакторинги, затра-

♣ Аспирант кафедры Вычислительной техники.