СИСТЕМА УПРАВЛЕНИЯ МНОГОЗАДАЧНЫМ РОБОТОМ

Шишков Ю. Ю.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Стракович А. И. – ассистент кафедры ЭВМ

Показан подход к созданию системы управления многозадачным роботом. Рассмотрены различные особенности и варианты разработки, а также процесс создания системы управления на платформе ROS поэтапно.

Система управления многозадачным роботом является одним из наиболее важных аспектов в робототехнике. Она отвечает за координацию и контроль всех функций робота, включая механику, электронику, программное обеспечение, сенсоры и другие компоненты.

Одна из наиболее распространенных систем управления многозадачным роботом — это Robot Operating System (ROS, операционная система для роботов), которая предоставляет набор инструментов и библиотек, которые позволяют программистам и инженерам управлять и координировать работу роботов с помощью высокоуровневых команд, а также позволяет решать сложные задачи в робототехнике, такие как навигация, обработка изображений и данных, управление манипуляторами и многие другие.

Данная система имеет графовую архитектуру, где обработка данных происходит в узлах (nodes), которые могут принимать и передавать сообщения (messages) между собой, что позволяет разработчикам создавать гибкие и модульные системы управления роботами. ROS состоит из двух частей. Первая, это ядро — roscore, которое отвечает за работу системы и взаимодействие всех пакетов. Вторая часть — это пользовательские пакеты (packages) или наборы этих пакетов, организованные в стек.

Создание робота на ROS можно разделить на несколько этапов, первый из которых это — выбор аппаратной платформы. Данная система поддерживает взаимодействие с множеством аппаратных платформ, таких как Raspberry Pi, Arduino, NVIDIA Jetson и другие. Выбор платформы зависит от требований проекта и бюджета. После выбора платформы необходимо установить ROS.

Затем создается рабочее пространство, в котором будут храниться все файлы проекта. В рабочем пространстве создаются пакеты, а в пакетах — узлы. Пакет — это основной блок программного обеспечения в ROS, в котором содержится код для управления роботом, сенсорами, алгоритмами и другими компонентами. Узел — это программа, которая выполняет определенную функцию, например, управление двигателями робота или обработка данных с сенсоров. Для создания узлов можно написать код на любом, поддерживаемом ROS, языке программирования: C++, Python или Java.

Следующим шагом создаются и настраиваются сообщения и сервисы — способы передачи данных между узлами, для чего опредяются типы данных, которые будут передаваться между узлами.

После создания всех компонентов проекта необходимо составить файл на языке разметки XML или YAML, который определяет, какие узлы и параметры нужно задействовать для работы робота.

На конечном этапе создания робота необходимо протестировать его работу. Для тестирования можно использовать инструменты ROS, такие как rostopic, rosservice, rqt и другие. После тестирования проводится настройка робота, например, настройка параметров узлов, калибровка сенсоров и других компонентов.

ROS изначально создавалась для использования в управлении сложными роботами, стоимость которых может достигать тысяч долларов. Поэтому прежде, чем затрачивать деньги на дорогостоящие физические компоненты, рекомендуется начать с моделирования и симуляции робота в виртуальной среде, а затем уже переходить к созданию физической модели.

В нашем случае, в связи с тем, что робот был сделан раньше, то используется обратный подход, а именно ROS настраивается под физическую модель. В качестве бортового компьютера, в роботе используется Raspberry Pi 4 Model B с 4 ГБ ОЗУ. Робот оснащен 4 колесами с моторами, драйвером для их управления, инфракрасными датчиками препятствий, микрофоном и динамиками.

Разработка программного обеспечения (ПО) для робота требует мощной визуализации, которая может оказаться очень ресурсоемкой. Например, работа с трехмерными объектами и визуализация на Raspberry Pi могут замедлить разработку ПО. Кроме того, некоторые задачи, которые ставятся перед роботом, такие как навигация в помещении, очень сложные и требуют большой вычислительной мощности. Выполнение этих алгоритмов на Raspberry Pi может быть очень медленным или даже невозможным. При использовании мощного ноутбука вместо Raspberry Pi, можно вести весь проект на одной машине. В данном же случае ROS установлен на две машины

– Raspberry Pi и персональный компьютер (ПК). Система на роботе занимается простыми вычислениями и поддерживает работу драйверов низкого уровня, а настольный компьютер с ROS выполняет сложные вычисления навигации и визуализации.

Поскольку ROS используется на двух компьютерах, необходимо на каждом из них в созданном рабочем пространстве поддерживать одинаковые и актуальные файлы проекта. Хотя такой подход может показаться неудобным, он является необходимым при использовании двух машин вместо одной. Для обеспечения синхронизации рабочих пространств на настольном компьютере и Raspberry Pi можно использовать различные утилиты, такие как git или rsync.

После установки ROS на обе машины можно приступать к написанию низкоуровневых драйверов, но сперва необходимо настроить сеть ROS, в которой может быть запущено только одно ядро roscore. Именно машина с запущенным ядром отвечает за работу всей системы. В сети ROS она называется master, а остальные машины становятся slave. В данном проекте в качестве master выбран настольный компьютер. Для всех ROS-компьютеров в сети нужно указать, какая именно машина является master.

После настройки сети ROS создано рабочее пространство. Затем был написан первый пакет проекта, который отвечает за драйвер моторов, были написаны два узла — talker и listener.

Узел talker запускается только на ПК, он отвечает за передачу сообщений роботу, здесь считываются команды управления с клавиатуры ПК. Этот узел является узлом-публицистом, которая публикует сообщения на шину chatter.

Узел listener запускается на Raspberry Pi, он отвечает за обработку команд пришедших роботу. Данный узел является узлом-подписчиком, который читает сообщения с шины chatter.

Протестировав драйвер моторов, был написан узел ds_talker, который отвечает за управление роботом с помощью геймпада DualSense PS5, который подключается по Bluetooth или USB к настольному компьютеру.

Структура ROS-графа проекта после написания всех узлов представлена на рисунке 1.

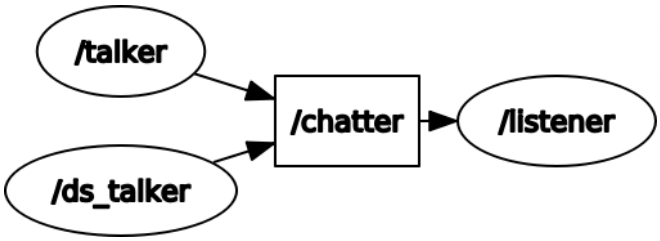

Рисунок 1 – ROS-граф проекта

Собранный робот представлен на рисунке 2. После успешного тестирования написанных узлов управления роботом, можно переходить к дальнейшему развитию проекта, а именно улучшению навигации при помощи датчиков препятствия.

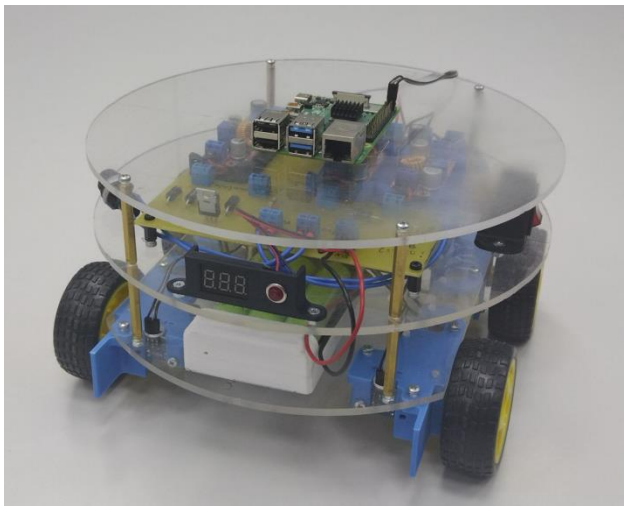

Рисунок 2 – Многозадачный робот

Список использованных источников:

1. ROS [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wiki.ros.org/ - Дата доступа: 01.04.2023