УДК 004.9

НАВИГАЦИОННАЯ СИСТЕМА В ЗАДАЧЕ УПРАВЛЕНИЯ АВТОНОМНЫМ МОБИЛЬНЫМ РОБОТОМ

Рябов Андрей Русланович[1], Решетников Андрей Геннадьевич[2], Ульянов Сергей Викторович[3]

[1]Магистрант;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
Старший лаборант;
Объединённый институт ядерных исследований;
141980, Московская обл., г. Дубна, ул. Жолио-Кюри, д. 6;
e-mail: rar.16@uni-dubna.ru.

[2]Кандидат технических наук, доцент;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
Старший научный сотрудник;
Объединённый институт ядерных исследований;
141980, Московская обл., г. Дубна, ул. Жолио-Кюри, д. 6;
e-mail: agreshetnikov@gmail.com.

[3]Доктор физико-математических наук, профессор;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
Ведущий научный сотрудник;
Объединённый институт ядерных исследований;
141980, Московская обл., г. Дубна, ул. Жолио-Кюри, д. 6;
e-mail: ulyanovsv46_46@mail.ru.

Статья посвящена настройке операционной системы и навигационного программно-алгоритмического пакета автономного мобильного робота для испытательного полигона МЛИТ ОИЯИ. В работе рассматривается реализация навигационной системы роботов. Система управления ROS (robot operating system) рассматривается начиная с подключения к роботу, продолжая предоставленным инструментарием и заканчивая областью применения.

Ключевые слова: *robot operating system* (*ros*), навигация, *ros visualization* (*rviz*).

Для цитирования:
Рябов А. Р., Решетников А. Г. , Ульянов С. В. Навигационная система в задаче управления автономным мобильным роботом // Системный анализ в науке и образовании: сетевое научное издание. 2022. № 1. С. 64–76. URL : http://sanse.ru/download/461.

NAVIGATION SYSTEM IN THE TASK OF CONTROL OF A MOBILE ROBOT

Ryabov Andrey R.[1], Reshetnikov Andrey G.[2], Ulyanov Sergey V.[3]

[1]Master's Degree student;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
Senior Laboratory Assistant;
6 Joliot-Curie Str., Dubna, Moscow region, 141980, Russia;
e-mail: rar.16@uni-dubna.ru.

[2]PhD in Engineering sciences, associate professor;
Dubna State University;

Institute of the system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
Senior Researcher;
Joint Institute for Nuclear Research;
6 Joliot-Curie Str., Dubna, Moscow region, 141980, Russia;
e-mail: agreshetnikov@gmail.com.

[3]Grand PhD in Physical and Mathematical Sciences, professor;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
Leading Researcher;
Joint Institute for Nuclear Research;
6 Joliot-Curie Str., Dubna, Moscow region, 141980, Russia;
e-mail: ulyanovsv46_46@mail.ru.

The article is devoted to setting up, launching, and navigating a mobile robot on a test site. The paper considers the implementation of intelligent control of a robot in a project of intelligent robust interaction of robotic systems. The ROS (robot operating system) control system is considered from the connection to the robot, continuing with the provided tools and ending with the field of application.

Keywords: robot operating system (ros), navigation, ros visualization (rviz).

For citation:
Ryabov A. R., Reshetnikov A. G., Ulyanov S. V. Navigation system in the task of control of a mobile robot. System Analysis in Science and Education, 2021;(1): 64–76(In Russ). Available from: http://sanse.ru/download/461.

Постановка задачи навигации автономного робота в заданном пространстве - помещении

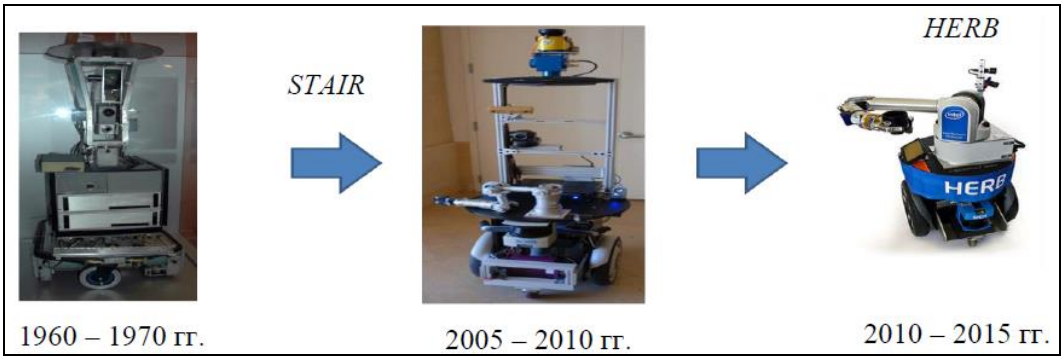
Одним из эффективных методов разработки технологий проектирования систем управления является робототехнический тренажёр, отличительной особенностью применения робототехники в современной *IT*-области является возможность непосредственной апробации алгоритмов управления в реальных условиях, наглядно показывая преимущества одних алгоритмов над другими. С недавнего времени, наблюдается стремительное распространение различных интеллектуальных и робототехнических устройств в повседневной жизни общества. Робототехнические устройства стали востребованной продукцией – роботы-пылесосы, роботы-доставщики, автопилоты. Однако, следует отметить и уменьшение количества открытых выпускаемых публикаций на тему интеллектуального управления, навигационных систем и многих других тем, отражающих современные прикладные и инженерные решения возникающих задач.

Мировой опыт развития технологий проектирования интеллектуальных систем управления автономными роботами чаще всего в основе лабораторного аппаратного базиса содержит самые современные сенсорные и исполнительные системы. В частности, исследовательские проекты в этой области представляют собой работы с максимально свободным допуском к смене и модернизации оборудования. Для этих целей приобретается самое современное оборудование – стерео- и видеокамеры, лазерные лидары, многозвенные манипуляторы, двигатели и различные одноплатные компьютеры. Именно огромное изобилие сенсоров, вычислительных средств и исполнительных устройств и механизмов позволяют создавать всё новые и более совершенные модели и макеты различных прикладных и коммерциализуемых систем.

При этом совершенствуется и алгоритмическая часть: всё большее распространение получают решения в области обучения, адаптации и самоорганизации ИСУ (интеллектуальных систем управления), основанные на применении методов роевого интеллекта, нейронных сетей и нечётких логик. Каждая из таких технологий, закладываемых в конечное изделие, является особым объектом интеллектуальной собственности и подлежит патентованию. Обычно 70% стоимости изделия составляет алгоритмическая часть.

В качестве примера интеллектуальных роботов-тренажёров на рис. 1 представлены учебно-исследовательский робот «мобильный манипулятор» университетов (*HERB*: *Home Exploring Robotic*

Butler и *STAIR*: *STanford Artificial Intelligence Robot* и мн. др.), являющихся исторически источниками и платформой для создания новых информационных технологий, программных фреймворков и конечных продуктов.



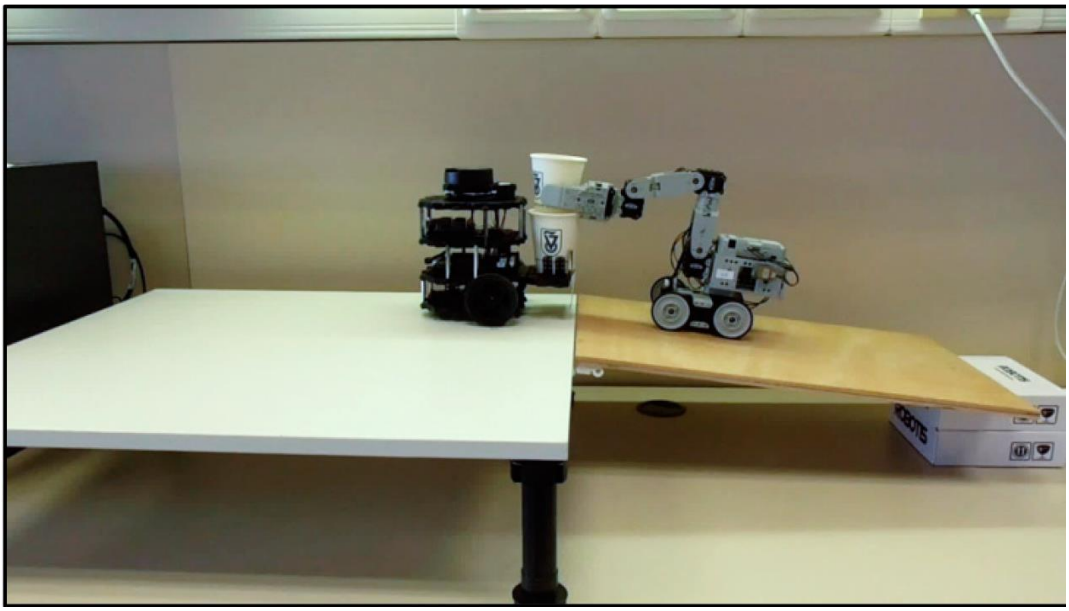


Рис. 1. Примеры современных исследовательских роботов передовых университетов

Основными задачами программной поддержки на основе мягких и квантовых вычислений процесса интеллектуализации физических установок и экспериментов ОИЯИ (объединённого института ядерных исследований) являются сопровождение проектов института, разработка архитектуры программных решений для лабораторий, реализация встраиваемых алгоритмов интеллектуального управления на основе мягких и квантовых вычислений в задачах обучения и адаптации интеллектуальных робототехнических систем, реализация конечных программных продуктов, внедрение разрабатываемых технологий в демонстрационные и образовательные примеры.

Современным отечественным аналогом подобных лабораторных установок является лабораторный испытательный полигон, разрабатываемый в МЛИТ (лаборатории информационных технологий им. М.Г. Мещерякова) ОИЯИ. Одной из задач данного полигона является разработка системы интеллектуального робастного взаимодействия робототехнических систем. На полигоне используются типовые робототехнические устройства (см. рис. 2):

- мобильный робот;
- статический робот-манипулятор;
- робот «перевёрнутый маятник».

Рис. 2. Испытательный полигон

Типовой пример взаимодействия состоит из нескольких подзадач: статический робот-манипулятор захватывает кубик в своей зоне работы и кладёт его на площадку на мобильном роботе. Мобильный робот отвозит кубик до другого манипулятора. Манипулятор забирает кубик и кладёт его на перевёрнутый маятник. Перевёрнутый маятник доставляет кубик до первого робота-манипулятора. В данной работе рассматривается управление мобильным роботом, в частности задача навигации автономного мобильного робота.

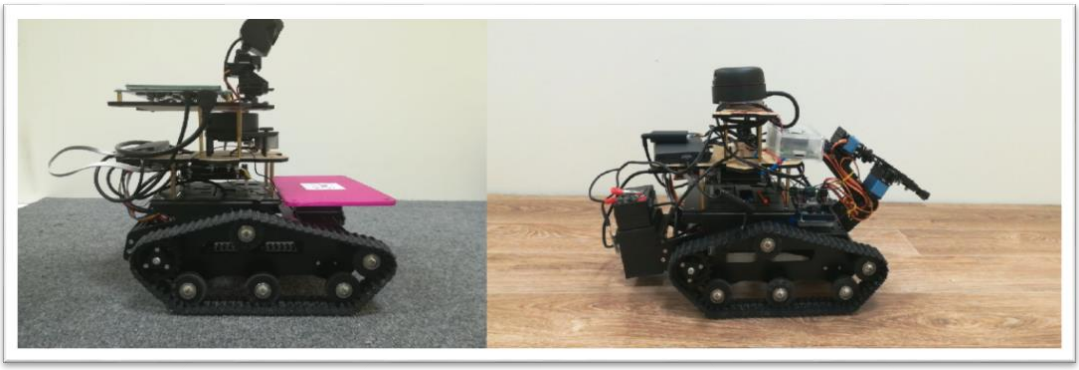


Рис. 3. Мобильные роботы

Задача навигации заключается в перемещении мобильного робота из стартовой точки в целевую. При этом необходимо определить кратчайший путь, обеспечить безопасность выполнения задачи роботом – объезд динамических и статических препятствий, выполнить построение карты рабочей зоны и локализовать себя на этой карте. Для решения задачи навигации были составлены подзадачи:

- построение карты местности;
- локализация робота;
- движение по реперным точкам.

Рассмотрим более подробно задачу навигации автономного мобильного робота. Сканируя помещение с помощью лидара, робот составляет карту местности, на которой отражены препятствия, локализует себя, строит кратчайший маршрут и осуществляет движение. Стоит отметить, что при внезапном появлении объектов на пути следования, робот перестроит маршрут.

Структура аппаратной части

Мобильный робот состоит из центрального микрокомпьютера, который отдаёт команды на движение плате управления, которая в свою очередь управляет моторами; данные о внешнем мире поступают через вебкамеру и лидар (см. рис. 4).

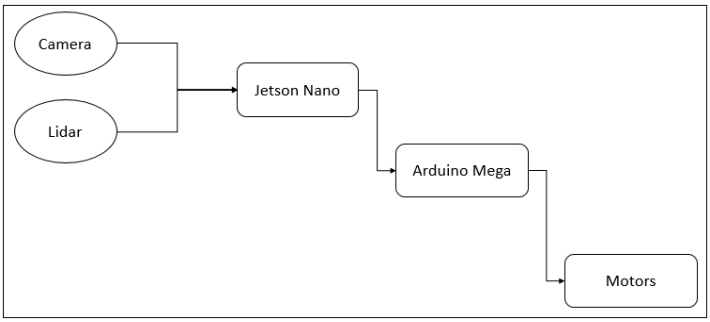


Рис. 4. Схема аппаратной части

Операционная система автономного робота ROS

ROS – это набор библиотек и инструментов для управления роботами. Это проект с открытым исходным кодом, автором которого является команда разработчиков Стенфордского университета (Stanford University). Концепция передачи данных в *ROS* продемонстрирована на рис. 5.

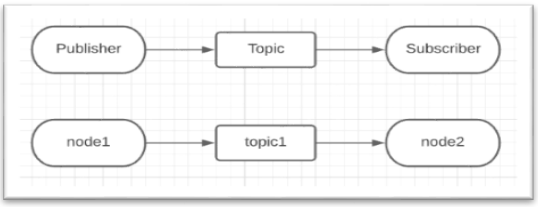


Рис. 5. Передача данных в ROS

Данные публикуются узлом (*node*) в тему (*topic*), затем узел, который подписан на эту тему получает оттуда данные. Так происходит передача сообщений от одного узла к другому. По этому принципу создаются и все библиотеки *ROS*, что позволяет легко встраивать сторонние наработки в уже работающий проект.

Настройка операционной системы робота

Начало работы с *ROS* – это настройка дистанционного управления роботом с удалённого компьютера. Необходимо убедится, что удалённый компьютер (используется ОС *Ubuntu*) и робот подключены к одной WiFi-сети, либо к единой точке доступа. Подключение с компьютера к роботу осуществляется по протоколу *ssh*. В начале необходимо определить *IP*-адрес компьютера и *IP*-адрес робота, для этого на компьютере в терминале используется команда:

- *hostname* -I.

После выполнения команды выводятся четыре числа, разделённых тремя точками, это и есть *IP*-адрес. Необходимо выполнить эту процедуру на роботе.

Подключение по протоколу *ssh*, между роботом и компьютером выполняется командой:

- *ssh user1@ip_addr*, где *user1* — это имя пользователя на роботе, *ip_addr* – это *IP-адрес* робота.

После ввода пароля администратора операционной системы робота имя пользователя в терминале заменится именем пользователя на роботе. Это означает, что в текущем окне терминала мы можем создавать, удалять, редактировать файлы робота, и в целом, управлять его операционной системой.

Далее необходимо отредактировать *bashrc*-файл, который находится в домашней директории. То есть с помощью любого редактора открываем файл *.bashrc* и в конец файла добавляем следующее:

- *export ROS_HOSTNAME = ip_addr*, где *ip_addr* – это *IP*-адрес робота;
- *export ROS_MASTER_URI = http://ip_addr*:11311, где *ip_addr* – это *IP*-адрес робота.

Откроем новый терминал для редактирования файлов компьютера. Нужно открывать файл *.bashrc* и в конец файла добавить почти то же самое:

- *export ROS_HOSTNAME = ip_addr*, где *ip_addr* – это *IP*-адрес компьютера;
- *export ROS_MASTER_URI = http://ip_addr*:11311, где *ip_addr* – это *IP*-адрес робота.

Закрываем оба терминала. Такие настройки позволяют отправлять данные с робота на компьютер и это облегчает дальнейшую работу.

Далее необходимо провести проверку работоспособности, для этого нужно запустить на роботе *ROS*. Для этого откроем терминал на компьютере, подключимся по *ssh* к роботу и выполним команду: *roscore*.

Если нет никаких ошибок и последняя строка звучит как *started core service [/rosout]*, то *ROS* успешно запущен. Оставим текущее окно термина работающим и зайдём на робота через новое окно терминала. Далее введём команду: *rosnode list*.

Эта команда отображает запущенные узлы. Если *ROS* запущен в данный момент, то вывод будет содержать */rosout*. Если же *ROS* не запущен, то вывод будет *ERROR: Unable to communicate with master!* Далее убедимся, что данные с робота передаются на компьютер. Для этого пропишем команду *rosnode list* с компьютера.

Пакеты

Пакет – это совокупность исполняемых файлов с необходимыми зависимостями от библиотек. В *ROS* программы организованы в пакеты, что позволяет удобно искать нужные данные в файловой системе. Пакет обеспечивает использование программ: запуск исполняемых файлов, поиск файлов с параметрами. Также организация в пакеты позволяет сгруппировать функционально связанные файлы в единый логический блок. Файлы из пакета можно запустить из любого расположения в системе с помощью команды *rosrun package_name exec_file*, где *package_name* – название пакета, *exec_file* – имя запускаемого файла.

Основными предустановленными пакетами являются:

- *rosrun rviz rviz* – средство визуализации данных подсистем робота, в том числе представления о среде функционирования;
- *rosrun teleop_twist_keyboard teleop_twist_keyboard.py* – управление роботом при помощи клавиатуры;
- *rosrun rqt_graph rqt_graph* – представление взаимосвязей узлов и тем с помощью направленного графа.

Для создания собственного пакета необходимо и достаточно, чтобы пакет включал в себя файл *CMakeLists.txt*и файл *package.xml*, в которых прописываются необходимые для работы системы зависимости. Для того, чтобы не создавать файлы зависимостей вручную, следует воспользоваться средствами *catkin*. *Catkin* – сторонне средство создания пакетов, которое отличается своей надежностью и рекомендуется разработчиками *ROS* к использованию. По умолчанию *catkin* уже установлен в *ROS*. Пакеты для удобства создаются в рабочем пространстве. Если такое пространство ещё не создано, то следует выполнить команды:

1. source /opt/ros/$ROS_DISTRO/setup.bash – для установки пакета;
2. *mkdir ~/catkin_ws/src/ -p* – создание рабочего пространства пакетов;
3. *cd ~/catkin_ws/src* – переход в папку «источники», в которой будут располагаться пакеты и сторонние библиотеки;
4. catkin_create_pkg package_name rospy std_msgs actionlib_msgs – создание пакета.

В пункте 4 *package_name* – название вашего пакета, далее перечисляются зависимости от сторонних библиотек: *rospy* – для работы с *Python*, *std_msgs* – для формирования стандартных сообщений, *actionlib_msgs* – для формирования сообщений *action*-файлов. На этом создание пакета закончено.

Внутри пакетов будут хранится программы, записанные в виде скриптов. Такие скрипты можно запускать из любого места в системе с помощью средств *ROS*, а именно командой *rosrun package_name exec_file*, где *package_name* – название пакета, *exec_file* – имя, запускаемого файла.

Далее, необходимо убедиться, что скрипты имеют право запускаться как исполняемые программы. Чтобы дать такое право, в папке со скриптом необходимо прописать *sudo chmod +x file_name*, где *file_name* – имя файла.

Launch-файлы

Для работы навигации используется большое количество узлов и, чтобы не запускать их в отдельности, можно запустить их в совокупности используя *roslaunch*. Это файл, который содержит в себе ссылки на узлы и дополнительные аргументы. Например, *launch*-файл может выглядеть так (см. рис. 6):

```
<launch>
        <arg name=”map_file” default=”$(find map_provider)/maps/my_map.yaml” />
        <node pkg=”map_server” type=”map_server” name=”map_server” args=”$(arg map_file)” />
</launch>
```

Рис. 6. Launch-файл

Стандартное управление роботом

Управление роботом можно реализовать через посылание команд «вперёд», «поворот». Написанная программа представляет собой скрипт, который посылает в тему */cmd_vel* сообщения о движении. На рис. 7 представлена простейшая программа, разберём её подробнее.

```
import rospy
from std_msgs.msg import String
from geometry_msgs.msg import Twist

def talker():
        pub = rospy.Publisher('/cmd_vel', Twist, queue_size=5)
        rospy.init_node('my_node')

        msg = Twist()

        while not rospy.is_shutdown():
                msg.linear.x = 2.0
                msg.linear.y = 0.0
                msg.linear.z = 0.0
                msg.angular.x = 0.0
                msg.angular.y = 0.0
                msg.angular.z = 0.0

                pub.publish(msg)
                rospy.sleep(1.0)

                msg.linear.x = 0.0
                msg.linear.y = 0.0
                msg.linear.z = 0.0
                msg.angular.x = 0.0
                msg.angular.y = 0.0
                msg.angular.z = 10.0

                pub.publish(msg)
                rospy.sleep(1.0)

if __name__ == '__main__':
        try:
                talker()

        except rospy.ROSInterruptException: pass
```

Рис. 7. Простейшая программа управления

- *import rospy* – импорт *rospy*, библиотеки для написания кода на *Python*;
- *from std_msgs.msg import String* – импорт типа данных *String*из пакета *std_msgs*;
- *from geometry_msgs.msg import Twist* – импорт типа сообщений *Twist*, в этом сообщении будут передаваться линейная и угловая скорости;
- *pub = rospy.Publisher('/cmd_vel', Twist, queue_size=5)* – публикация в тему */cmd_vel*сообщения типа *Twist*с максимальной очередью сообщений 5;
- *rospy.init_node('my_node')* – создание узла с именем *my_node*;
- *msg = Twist()* – присваивание типа сообщений *Twist*переменной *msg*;
- *while not rospy.is_shutdown():* – пока робот будет работать, выполнять…;
- *msg.linear.x = 2.0* – команда на движение вперёд, далее представлены все поля типа сообщений *Twist*;
- *pub.publish(msg)* – публикация сообщения *msg*;
- *rospy.sleep(1.0)* – задержка в одну секунду;
- *msg.angular.z = 10.0* – поворот налево.

Программа заканчивается стандартной конструкцией запуска. В результате робот движется по контуру квадрата. Подобным способом можно задать движение из одной точки в другую.

Управление роботом с помощью actionlib

В *ROS* также имеется возможность автоматизированного управления. Речь идёт о том, что вместо команд на движение можно отправлять координату места назначения. Целевая координата – точка на карте, в которую роботу необходимо переместиться. Данный механизм также реализуется через темы, и называется «навигационным стеком». Этот навигационный стек работает следующим образом: в тему */move_base_simple/goal* отправляется сообщение, которое содержит в себе координаты точки. Робот, с применением интеллектуального алгоритма, выстраивает путь до точки и начинает движение. В процессе движения робот может отправлять данные о своем местонахождении или о том, сколько времени он находится в пути. При достижении пункта назначения, робот возвращает результат – «цель достигнута» (см. рис. 8).

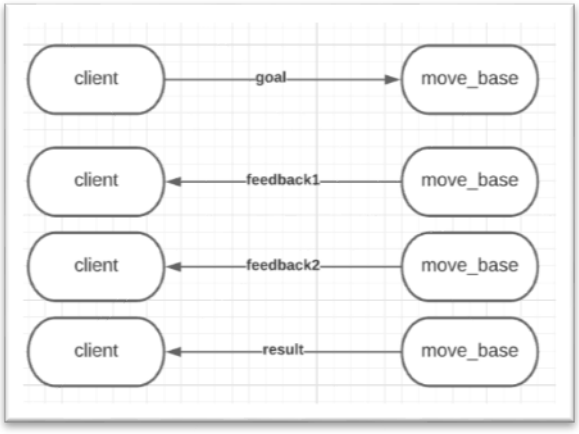


Рис. 8. Общение action-клиента и action-сервера

Action-сервер запускается автоматически в узле *move_base*, а *action*-клиент необходимо написать самостоятельно. Ниже представлены основные строчки кода:

- *import actionlib*;
- *from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal*;
- *client = actionlib.SimpleActionClient('move_base', MoveBaseAction)*;
- *client.wait_for_server()*;
- *goal = MoveBaseGoal()*;
- *goal.target_pose.header.frame_id = 'map'*;

- *goal.taregt_pose.header.stamp = rospy.Time.now()*;
- *goal.target_pose.pose.position.x* = 0.5;
- *goal.target_pose.pose.orientation.w* = 1.0;
- *client.send_goal(goal)*;
- *wait = client.wait_for_result*().

Во время запуска */move_base* создаётся и запускается *action*-сервер с именем *move_base*, который принимает координаты целевой точки. Написанная программа *action*-клиента устанавливает связь с сервером по его названию, создаёт экземпляр *MoveBaseGoal*, указывает необходимые поля и отправляет сообщение (цель) на сервер.

Карта местности

Данные с лидара представляют собой облако точек. В *ROS* эти данные имеют тип *sensor_msgs/LaserScan* и представляют собой массив с расстояниями *float32[] ranges*. С помощью средства визуализации *RViz* эти данные можно представить в наглядном виде, удобном для разработчика и на их основе создать карту местности (см. рис. 9).

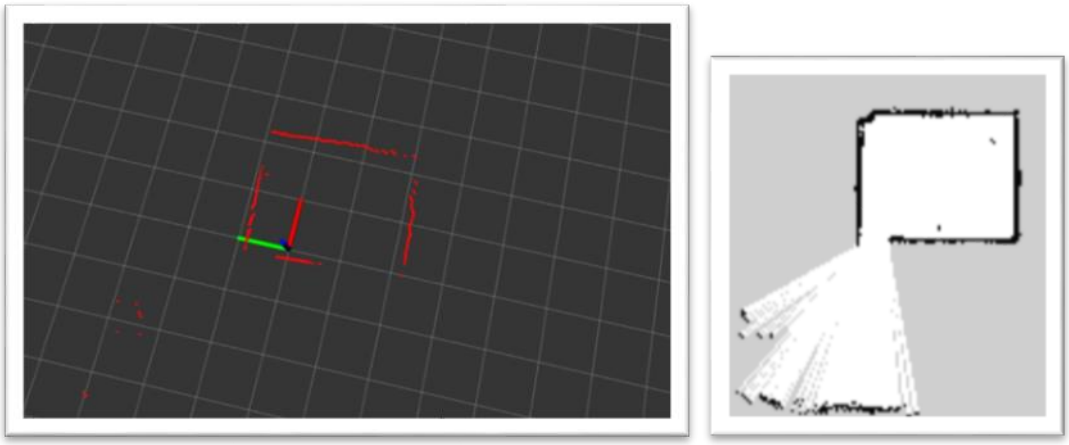


Рис. 9. Данные с лидара и карта местности

Загрузим карту местности можно ограничить движение робота в нужных местах. Например, есть препятствие – «яма с галькой», куда роботу нельзя заезжать, при этом сенсорная система робота не позволяет этого определить. Однако, для ограничения движения робота в этой точке, «яму с галькой» можно выделить на предварительной карте местности, нанеся соответствующее границы (см рис. 10). Данные с карты местности интерпретируются так:

- белый цвет – свободная зона;
- черный цвет – стена;
- серый цвет – неизвестность.

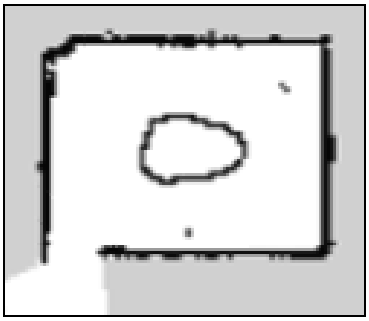


Рис. 10. Отредактированная карта

Бывает так что робот заезжает на территорию, отмеченную как препятствие, и тогда он теряется на карте, поэтому рассмотренное решение ненадёжно и часто мешает нормальному движению робота.

Локализация

В навигационном стеке используются локальная и глобальная системы координат. Локальная система координат имеет начало в центре самого робота и помогает в расчёте угловой и линейной скоростей движения и определения местоположения относительно предыдущей точки. Каждый двигатель робота оснащен датчиком контроля поворота колеса — энкодером. Данный тип датчиков вырабатывает сигнал — импульс, свидетельствующий о движении колеса.

Подсчёт разницы количества импульсов в единицу времени:

$$DeltaTick\_R = Tick\_R' - Tick\_R,$$

$$DeltaTick\_L = Tick\_L' - Tick\_L.$$

Затем считается расстояние, которое прошло каждое колесо:

$$DR = 2*Pi*R*(DeltaTick\_R/N),$$

$$DL=2*Pi*R*(DeltaTick\_L/N).$$

Затем можно определить расстояние, пройденное роботом. Далее высчитываются координаты в глобальной системе координат:

$$X' = X+DC*cos(theta),$$

$$Y'=Y+DC*sin(theta),$$

$$theta' = theta+(DR+DL)/L.$$

Глобальная система координат показывает местоположение робота относительно заранее заданного начала координат. Началом системы координат является местоположение робота в момент запуска узлов навигации.

Если у робота нет заранее заданной карты, он может её построить на ходу с помощью алгоритма *SLAM* (от англ. *simultaneous localization and mapping*). Эта технология представляет собой алгоритм одновременной локализации и построения карты. Она отлично подходит для передвижения по неизвестной территории. Данные с лидара можно также визуализировать в режиме реального времени (см. рис. 11).

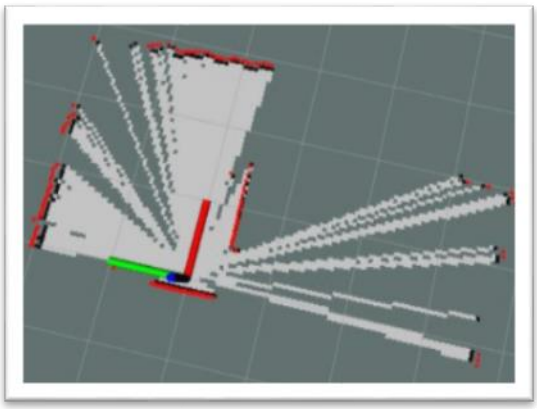


Рис. 11. Карта во время работы SLAM

Навигация

Данные о внешнем мире преобразуются в команды движения в пакете *move_base*. Рассмотрим узлы:

- *imu* – данные с гироскопа;
- *scan* – данные с лидара;
- *map_server* – данные о карте;
- *move_base* – отвечает за передвижение.

На основе карты местности создаётся глобальная карта затрат. Затем при получении цели (тема */move_base_simple/goal*) выстраивается путь до пункта назначения. Далее во время движения строится локальная карта затрат на основе данных с лидара и соответствующий путь движения (см. рис. 12). Таким образом глобальная карта затрат помогает определить путь, а локальная карта затрат помогает избежать внезапно появившиеся препятствия.

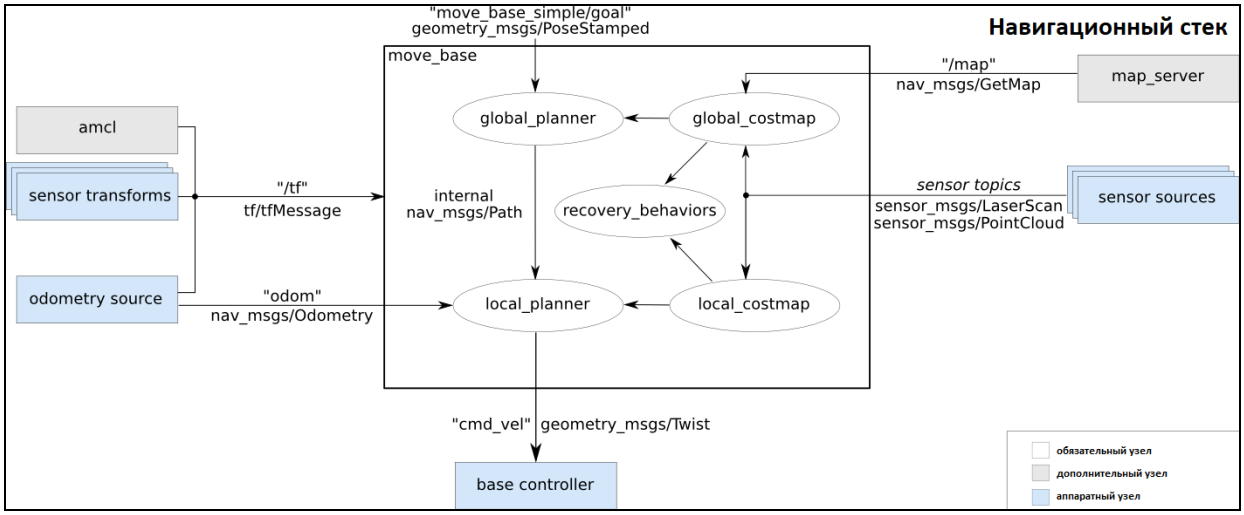


Рис. 12. Навигационный пакет [http://wiki.ros.org/move_base]

В *2D-навигации* существуют серьёзные изъяны. Например, при перемещении робота вдоль оси *z*, скажем, при заезде на порог, робот теряет своё местоположение и не может локализовать себя. В этом случае робот считается потерянным и приходится перезапускать систему навигации. Это можно исправить, если дописать поведение робота в ситуации, когда данные с лидара резко изменились и не соответствуют заранее загруженной карте местности.

Эксперимент

Рассмотрим несколько алгоритмов построения маршрута:

- поиск в ширину;
- алгоритм Дейкстры;
- *A**.

Поиск в ширину

Алгоритм считает расстояние до каждой соседней точки, начиная с начальной. Алгоритм останавливается посчитает последнюю точку. Минус такого алгоритма – он не позволяет объезжать препятствия. Плюс – это то, что для достижения конечной точки не обязательно обходить граф целиком и считать расстояние между каждой парой узлов.

Алгоритм Дейкстры

Алгоритм решает задачу нахождения кратчайшего пути. Задача представляется в виде взвешенного ориентированного графа. До каждого соседнего узла, начиная с начального, рассчитывается

расстояние до начального узла. Если такое расстояние уже посчитано, то выбирается наименьшее. Алгоритм проходит по каждому узлу и возвращает наименьшее расстояние между начальным и конечным узлами. Обход узлов обусловлен их приоритетом. По своей сути алгоритм учитывает стоимость прохождения по каждому маршруту, а значит, вывод основывается на карте затрат, в которой отражён рельеф местности. Минус – большой объём вычислений. В отличии от алгоритма «поиск в ширину» алгоритм Дейкстры останавливает вычисления, когда достигает целевой точки.

Алгоритм «лучший – первый» работает быстрее, однако не гарантирует нахождение кратчайшего пути.

Алгоритм *A**

Алгоритм *A** сочетает в себе преимущества предыдущих алгоритмов.

На основе данных с лидара и карты местности строится оптимальный маршрут. На рис. 13 показан маршрут движения без внезапных препятствий (линия на рисунке отображает маршрут). На рис. 14 при появлении препятствия маршрут перестраивается в реальном времени (точками отображено динамическое препятствие).

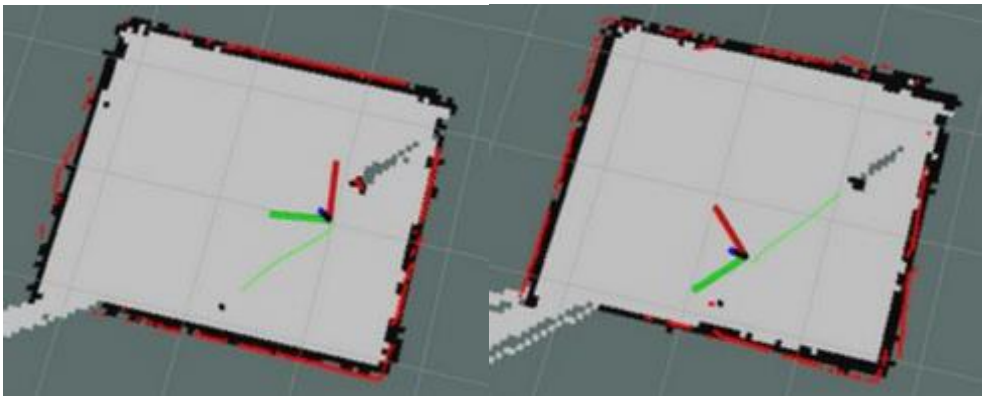


Рис. 13. Маршрут без препятствий

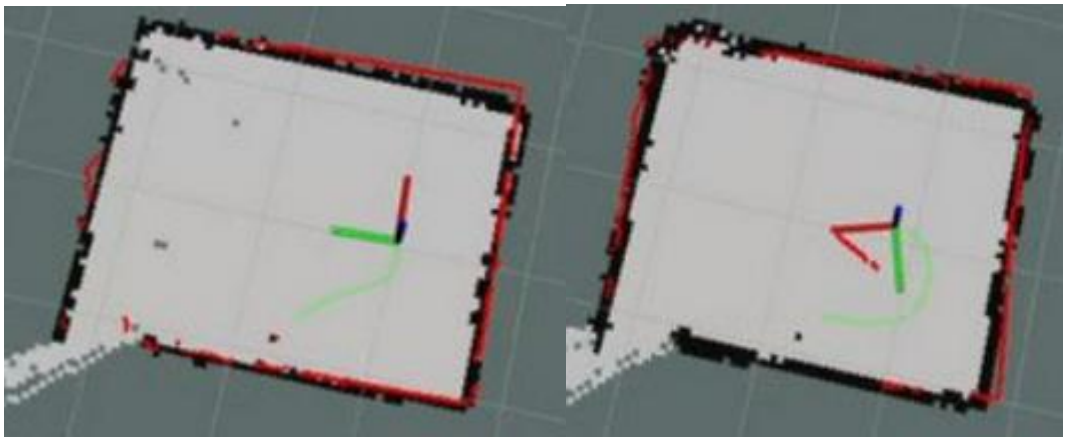


Рис. 14. Динамическое изменение маршрута

Заключение

В статье была рассмотрена платформа для решения задачи навигации *ROS*. Область применения мобильных роботов обширна и число задач возрастает с каждым годом. Уже сейчас есть такие задачи как перемещение опасных для здоровья человека грузов (например, радиоактивных частиц), задачи съёмки помещений. Интеллектуальная навигация уже применима в повседневных задачах таких как уборка помещений с помощью робота-пылесоса и интеллектуальное управление (автопилот) в автомобиле. Спрос на такие задачи растёт и *ROS* является рабочей платформой для реализации простых задач и может служить каркасом для больших проектов.

Список источников

1. Mo, H., Tang, Q., Meng, L. Behavior-Based Fuzzy Control for Mobile Robot Navigation // Mathematical Problems in Engineering. – 2013 . –Vol.2013. –DOI: 10.1155/2013/561451.
2. Quigley M. Programming Robots with ROS: A Practical Introduction to the Robot Operating System / M. Quigley, B. Gerkey,W. D.Smart. – O'REILLY, 2015. – 448 с.
3. Pyo Y. ROS. Robot Programming. From the basic concept to practical programming and robot application/ Y. Pyo, H. Cho, R. Jung, T. Lim.– ROBOTIS, 2017. – 478 с.
4. Zaman S., Slany W., Steinbauer G. ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues // Saudi International Electronics, Communications and Photonics Conference (SIECPC). – 2011. – С. 1-5. – DOI: 10.1109/SIECPC.2011.5876943.
5. Бобырь М. В. Адаптация системы управления мобильным роботом на основе нечеткой логики // Мехатроника, автоматизация, управление. – 2015.– Т.16, №7. – С.449-455. – DOI: https://doi.org/10.17587/mau.16.449-455.
6. Алгоритм управления автономным двухколесным мобильным роботом «Мотобот» / А. А. Бобцов, А. С. Боргуль, К. А. Зименко, А. А. Пыркин // Научно-технический вестник информационных технологий, механики и оптики. – 2011.– №5 (75). – С. 63-68.
7. Салабуха Н. Введение в Robot Operating System/ Н. Слабуха; компания "Братья Вольт". – URL: http://docs.voltbro.ru/starting-ros/
8. Ющенко А.С. Диалоговое управление роботами с использованием нечетких моделей. – Дата публикации: 17.10.2016. –URL: https://dmee.ru/docs/100/index-28441.html.
9. Щербатов И. А., Проталинский И. О., Проталинский О. М. Управление группой роботов: компонентный подход // Информатика и системы управления. – 2015. – № 1(43). – С. 93- 104.