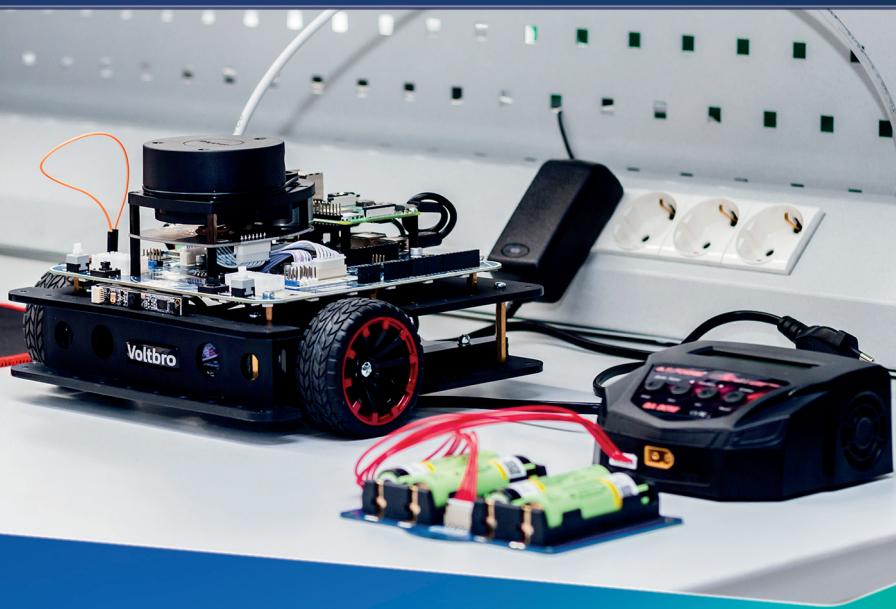




МИНИСТЕРСТВО НАУКИ  
И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ



П. А. Ефимов, А. А. Сыч, И. А. Баранчугов,  
Н. Т. Морозова, Б. С. Ноткин



# ЭКСПЛУАТАЦИЯ СЕРВИСНЫХ РОБОТОВ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

---

П. А. Ефимов, А. А. Сыч, И. А. Баранчугов,  
Н. Т. Морозова, Б. С. Ноткин

## ЭКСПЛУАТАЦИЯ СЕРВИСНЫХ РОБОТОВ

Учебное пособие



УДК 004.896  
ББК 32.816  
Э41

Рецензенты:

кандидат физико-математических наук, ведущий инженер Института механики МГУ им. М. В. Ломоносова, корневой эксперт компетенции «Эксплуатация сервисной робототехники» К. В. Климов;  
кандидат физико-математических наук, руководитель проекта НТИ, ООО «СПУТНИКС», С. О. Карпенко

Э41      Эксплуатация сервисных роботов: учеб. пособие / П. А. Ефимов, А. А. Сыч, И. А. Баранчугов, Н. Т. Морозова, Б. С. Ноткин. – СПб.: ГУАП, 2021. – 155 с.

ISBN 978-5-8088-1653-4

Данное учебное пособие включает в себя шесть разделов, связанных с большими блоками компетенций, необходимых как для формирования у обучающихся знаний, умений и навыков в области сервисной робототехники, так и для выполнения демонстрационного экзамена.

Пособие состоит из следующих блоков: «Основы операционной системы *Linux*», «Программирование на языке *Python*», «*ROS – Robot Operating System*», «Программирование микроконтроллерных плат», «Управление роботизированной платформой *TurtleBro*».

Предназначено для формирования и отработки на практике навыков и компетенций в области сервисной робототехники, операционной системы *Linux*, электроники, программирования системы *ROS*, программирования на языках *Python* и *C++*.

Содержит необходимые сведения, позволяющие обучающимся решать задачи навигации и взаимодействия с датчиками роботов, обслуживать, диагностировать и настраивать сервисных роботов, а также выполнять задания демонстрационного экзамена.

УДК 004.896  
ББК 32.816

ISBN 978-5-8088-1653-4

© Санкт-Петербургский государственный  
университет аэрокосмического  
приборостроения, 2021  
© Дальневосточный федеральный  
университет, 2021

## ПРЕДИСЛОВИЕ

*Настоящее учебное пособие представляет собой разработанный авторами по результатам предпринятых исследований и апробированных методических разработок образовательный материал для изучения и освоения компетенции FutureSkills в рамках актуализированных образовательных программ высшего образования в соответствии с запросом развивающихся индустриальных рынков инновационной экономики.*

*Пособие является результатом работы в рамках проекта по постановке компетенций FutureSkills в образовательную деятельность вузов, цель которого – обеспечение системного подхода к кардинальному преобразованию в кадровом потенциале научно-образовательной сферы, внедрение практико-ориентированных методик, развитие технологических проектов, создание инструментов усовершенствования образовательной инфраструктуры в процессе внедрения образовательных программ высшего образования по компетенциям «Ворлдскиллс» в образовательную деятельность высших учебных заведений.*

*Материал составлен в соответствии с запросом на обновление образовательных программ высшего образования с учетом актуального и перспективного запроса рынка труда и внедрения практико-ориентированной подготовки, структурирован в соответствии с основной учебной программой и обязательной формой контроля уровня образования обучающихся с помощью демонстрационного экзамена.*

*Пособие содержит необходимый теоретический, практический и справочный материал, оно позволит улучшить методику*



МИНИСТЕРСТВО НАУКИ  
И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ





СЕВАСТОПОЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ

преподавания и подготовить обучающихся к успешной аттестации.

Предназначено для обучающихся по направлениям бакалавриата.

Подготовлено в рамках выполнения работы по теме: «Методическое сопровождение внедрения образовательных программ по компетенциям "Ворлдскиллс" в образовательную деятельность организаций высшего образования» по заданию Министерства науки и высшего образования РФ, при поддержке АНО «Агентство развития профессионального мастерства (Ворлдскиллс Россия)» и координации ФГАОУ ВО «Санкт-Петербургский государственный университет аэрокосмического приборостроения» в партнерстве с ФГАОУ ВО «Национальный исследовательский ядерный университет "МИФИ"», ФГАОУ ВО «Севастопольский государственный университет», ФГАОУ ВО «Дальневосточный федеральный университет», ФГАОУ ВО «Южный федеральный университет».

## **ВВЕДЕНИЕ**

Роботы постепенно входят в повседневную жизнь. Еще несколько лет назад роботы присутствовали только на заводах, а сегодня никого не удивить роботом-пылесосом в квартире. Такие роботы получили название сервисных.

*Сервисный робот – это автоматическое устройство, которое помогает людям, выполняя рутинную, удаленную, опасную или повторяющуюся работу, включая работу по дому. Как правило, они автономны и/или управляются встроенной системой управления с возможностью ручного управления.*

Постепенно роботы будут входить в нашу жизнь все больше и больше, выполняя большой фронт работ, заменяя человека. Количество роботов будет расти, и потребуется много специалистов, способных выполнять настройку, обслуживание и ремонт устройств данного типа. Специалисты должны будут обладать набором компетенций, которые включают в себя знание операционных систем, в большинстве случаев *Unix*-подобных, знание и понимание работы микроконтроллерных и микропроцессорных систем, а также знание высокоуровневых языков программирования.

В основном сервисные роботы, которые используются сейчас, построены на базе *Robot Operating System (ROS)* – операционной системы для роботов. *ROS* – метаоперационная система, предоставляет большой набор функций и возможностей по распределенной работе, а также позволяет быстро и эффективно разрабатывать программное обеспечение для роботов.

# 1. ОСНОВЫ ОПЕРАЦИОННОЙ СИСТЕМЫ LINUX

## 1.1. Операционная система Linux

*ROS* – это метаоперационная система, а значит, для ее работы требуется основная операционная система, поверх которой будет развернута *ROS*. В настоящее время есть версии *ROS* для разных платформ, однако основной операционной системой (ОС) для *ROS* является *Ubuntu Linux*.

*Linux* – самая популярная ОС для встраиваемых приложений, промышленной автоматизации и робототехники.

Ядро (*kernel*) – основной компонент ОС, отвечающий за управление процессами, виртуальной памятью и драйверами устройств.

Разделяемые системные библиотеки (*system libraries*) содержат стандартный набор функций, используемых приложениями для запросов к системным сервисам ядра. В библиотеках хранится также код функций отдельных сервисов ядра, исполняемых в обычном режиме без привилегий ядра.

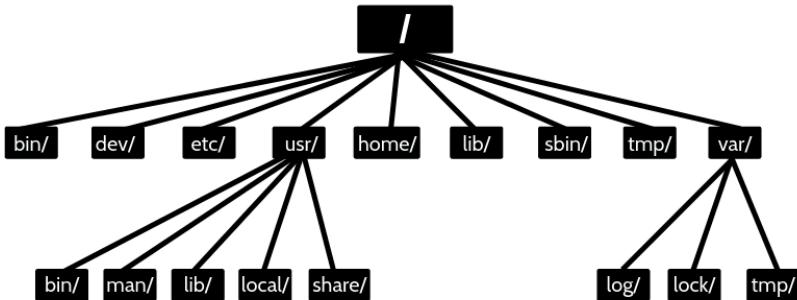
Более подробную информацию об «анатомии» ОС *Linux* можно узнать по ссылке: <https://habr.com/ru/post/531872>.

Важной частью ОС *Linux* является организация ее каталогов. В *Linux* на самом верхнем уровне файловой иерархии находятся не физические и логические диски, а один-единственный корневой каталог, обозначаемый косой чертой «/» (слэш). Каждый последующий вложенный каталог отделяется от родительского также слэшем. Путь заканчивается либо каталогом, либо файлом. Например, */boot/grub* или */home/pl/test.txt*. Второй путь указывает, что в корневом каталоге есть папка *home*, в ней находится каталог *pl*, в котором имеется файл *test.txt*. Общая структура каталогов представляет собой дерево, где «/» является корнем, а каждый последующий выходит из него. Понимание структуры каталогов позволит верно понять, как переходить по каталогам, а также разделит два вида путей: абсолютный и относительный. Структура основных каталогов ОС *Linux* представлена на рис. 1.1.

Абсолютный путь представляет полное имя от корневого каталога (директорий) и содержит в себе полный набор каталогов (ветвей) до необходимого каталога или файла. Абсолютный путь похож на движение по ветви дерева от корня до нужного листа.

Относительный путь представляет собой набор каталогов (директорий) от текущего расположения до желаемого каталога или файла.

Взаимодействие с каталогами возможно двумя основными способами. С использованием проводника и/или командной строки



*Рис. 1.1. Структура основных каталогов ОС Linux*

*Linux*. Учитывая специфику работы с роботом, а также необходимые задачи, которые требуется решать в рамках работы с сервисным роботом, рекомендуется использовать второй вариант, а именно выполнять взаимодействие через командную строку. Данный навык применим не только в процессе взаимодействия с роботом, он дает возможность администрировать и управлять любой другой ОС *Linux*, так как правила аналогичны, а команды универсальны.

Для работы с ОС *Linux* существует несколько основных подходов:

- 1) установка ОС *Linux* как основной ОС на ваш рабочий ПК;
- 2) установка ОС *Linux* в качестве второй ОС рядом с текущей;
- 3) установка ОС *Linux* как подсистемы *Windows* (*Windows Subsystem for Linux*);
- 4) запуск ОС *Linux* как виртуальной ОС на вашем ПК.

В зависимости от ваших навыков и умений вы выбираете наиболее предпочтительный вариант установки или настройки ОС для работы. Если вам не требуется постоянная работа в данной системе, то рекомендуется использовать вариант с виртуализацией.

## 1.2. ОС *Linux* и виртуальная машина

*Виртуализация – это создание изолированных окружений в рамках одного физического устройства (в нашем случае – компьютера). Каждое окружение при этом выглядит как отдельный компьютер со своими характеристиками, такими как доступная память, процессор и т. п. Такое окружение называют набором логических ресурсов, или виртуальной машиной.*

Работа виртуальной ОС аналогична работе обычного приложения-программы на вашем ПК, но такая система называется гостевой – *guest*, а основная ОС – *host*.

На рынке существует несколько основных программ, а также компаний – разработчиков программного обеспечения для запуска виртуальных машин (*Qemu*, *VirtualBox*, *VMWare* и др.). В качестве примера работы предлагается использовать бесплатное решение *VMware Workstation Player* от компании *VMware*. Внешний вид окна *VMware Workstation Player* представлен на рис. 1.2.

Скачать проигрыватель виртуальных машин вы можете по ссылке: <https://www.vmware.com/ru/products/workstation-player.html>.

Для выполнения заданий и работы с тестовой системой вам также требуется скачать образ ОС *Ubuntu Linux*. Для простоты настройки системы и взаимодействия с *ROS* рекомендуется установить готовую сборку-образ, который предоставляется в рамках обучения.

Готовый образ вы можете скачать по ссылке <https://rosi-images.datasys.swri.edu>.

После подготовки вашего рабочего места и запуска виртуальной машины перед вами открывается рабочий стол ОС *Linux*. Данный рабочий стол похож на тот, что вы привыкли видеть, например, в ОС *Windows* или *MacOS*, за исключением некоторых специфичных элементов дизайна и другого внешнего вида. Внешний вид ра-

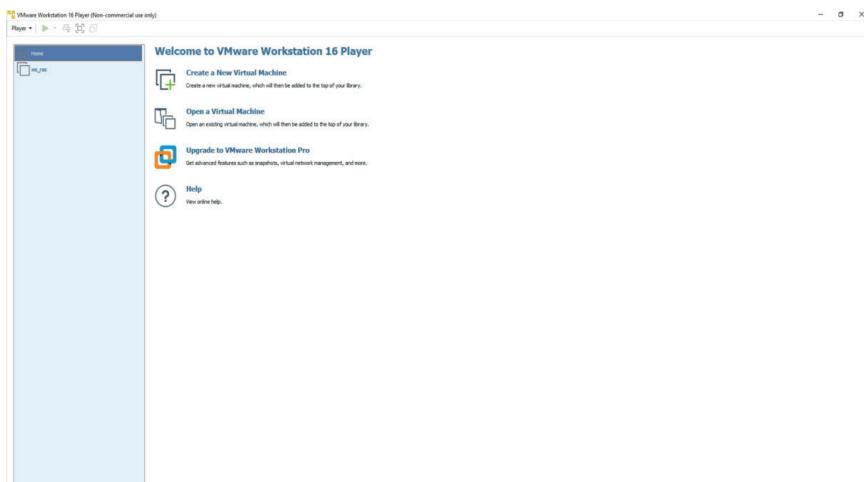


Рис. 1.2. Окно *VMware Workstation Player*



Рис. 1.3. Внешний вид ОС Linux

бочего стола и основного меню ОС *Linux (Ubuntu)* представлен на рис. 1.3.

Основной функционал, к которому вы привыкли, сохраняется: создание, перемещение, копирование файлов, работа с документами и приложениями, которые имеют аналоги в *Linux*.

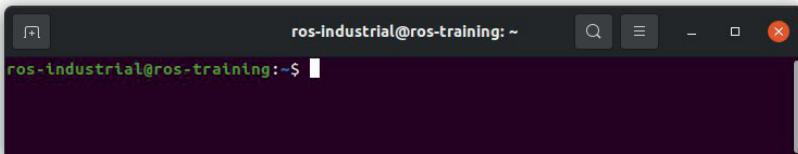
Как было сказано ранее, для оттачивания навыка администрирования как ОС *Linux*, так и робота, важно научиться взаимодействовать с командной строкой, которая дает полный доступ к функциям и возможностям, заложенным в ОС.

Взаимодействие с командной строкой требует соблюдения ряда основных правил и на первом этапе будет вызывать трудности, но постепенно даст неоспоримые преимущества.

### 1.3. Основы работы в командной строке Linux

Командная строка в *Linux* является специальной программой, которая называется *Terminal*. Внешний вид терминальной программы представлен на рис. 1.4.

Для программистов, разработчиков робототехники, операторов роботов и инженеров *Terminal* – это один из основных инструментов работы.



*Рис. 1.4. Внешний вид терминальной программы*

Давайте рассмотрим базовые команды терминала *Linux*, которые пригодятся нам при работе.

Запуск терминала возможен двумя способами:

- 1) через меню списка программ;
- 2) через сочетание клавиш *Ctrl + Alt + T*.

В *Terminal* есть набор быстрых клавиш (горячих клавиш), которые будут использоваться постоянно и ускорят вашу работу:

1. Клавиша *Tab* – автодополнение, которое позволяет быстро набирать команды, имена файлов или пути к ним. Начните набирать команду, путь или имя, нажмите клавишу *Tab*. Если такой объект существует, то оболочка дополнит его. Существуют варианты, когда в системе есть объекты, у которых начало имени совпадает. В этом случае после двойного нажатия на клавишу *Tab* будут показаны все возможные варианты.

2. Клавиша Стрелка вверх – предыдущая введенная команда.
3. Клавиша Стрелка вниз – следующая введенная команда.
4. Сочетание клавиш *Ctrl + C* – прервать выполнение запущенной программы / процесса.
5. Сочетание клавиш *Ctrl + Z* – приостановить выполнение запущенной программы / процесса.
6. Сочетание клавиш *Ctrl + D* – завершить текущий сеанс связи.
7. Сочетание клавиш *Ctrl + Shift + C* – копирование текста.
8. Сочетание клавиш *Ctrl + Shift + V* – вставка текста.

Использование быстрых клавиш ускоряет работу с терминальной программой, а также позволяет не запоминать большие блоки команд, которые уже были использованы в процессе работы. Для копирования и вставки используется комбинация клавиш с *Shift*, потому что стандартная для ОС комбинация клавиш занята в терминале прерыванием выполнения программы.

В разделе об ОС *Linux* были рассмотрены два варианта работы с путями. По умолчанию при открытии терминальной программы вы попадаете в каталог «~» – домашний каталог. Его абсолютный

путь выглядит как путь от корневого каталога до каталога учетной записи. При использовании виртуальной машины абсолютный путь выглядит так: `/home/ros-industrial/`.

Для взаимодействия с путями *Linux* существует ряд специальных символов, помогающих взаимодействовать с ними:

- «..» текущая директория;
- «...» директория на уровень выше;
- «~» домашняя директория;
- «\*» любое количество любых символов;
- «?» ровно один любой символ.

#### 1.4. Команды терминала ОС Linux

Данный подраздел посвящен базовым и часто применяемым командам терминала *Linux*, а также тем командам, которые пригодятся для работы с сервисным роботом. В нашем случае взаимодействие с терминалом выполняется в ОС *Ubuntu Linux*, которая имеет ряд встроенных возможностей, упрощающих работу с ОС. Одной из таких возможностей является наличие быстрых клавиш для запуска терминальной программы (*Ctrl + Alt + T*). Кроме того, существует возможность открытия терминала, используя ярлык в панели меню, представленной на рис. 1.5.

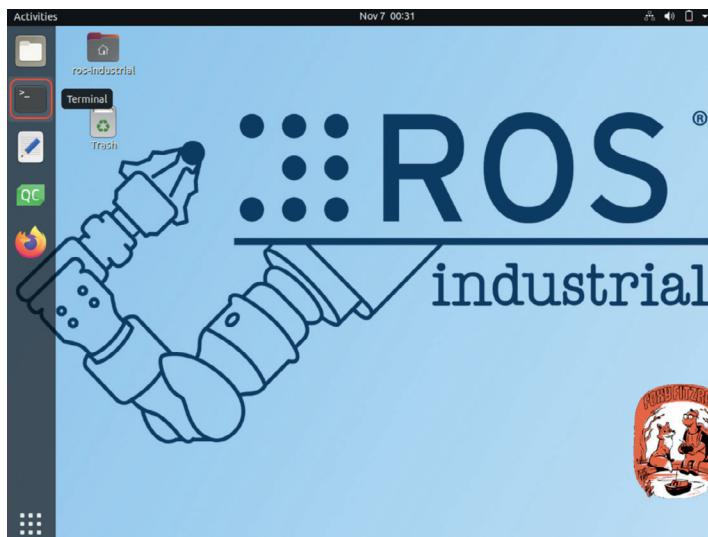


Рис. 1.5. Запуск терминала на панели задач

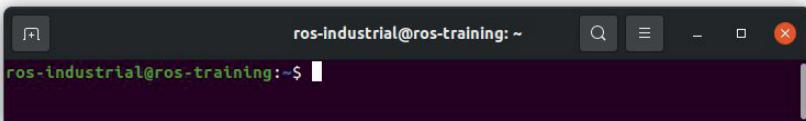


Рис. 1.6. Приглашение пользователя для ввода команд

Большинство команд построено по принципу:  
команда – ключ команды [аргументы]

Изменяя значения ключей и дополняя поле «аргументы», можно добиться различных вариантов работы одной выбранной команды.

*Интересный факт: для вывода информации о команде, ее ключах и аргументах есть отдельные команды терминала (*man*, *whatis*, а также отдельный ключ «*--help*»).*

При запуске командной строки перед вами появляется символ `$`, который является символом приглашения пользователя для ввода команд. Пример окна с приглашением пользователя представлен на рис. 1.6.

Следует обратить внимание, что *ros-industrial* – это имя пользователя, через которое вы входите в систему (совпадает с именем в пути до домашнего каталога), а *ros-training* – имя компьютера. Эти значения позволяют определить, под каким пользователем и на каком компьютере вы в настоящий момент работаете. Данное уточнение пригодится, когда будет осуществляться подключение к работе через сеть.

#### 1.4.1. Команда *ls*

Одной из самых простых, но полезных команд терминала *Linux* является команда *ls* – сокращение от *list* – список, если выполнить

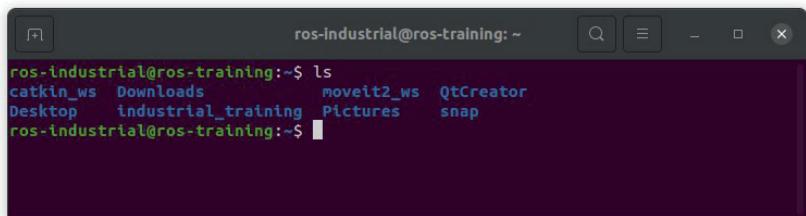
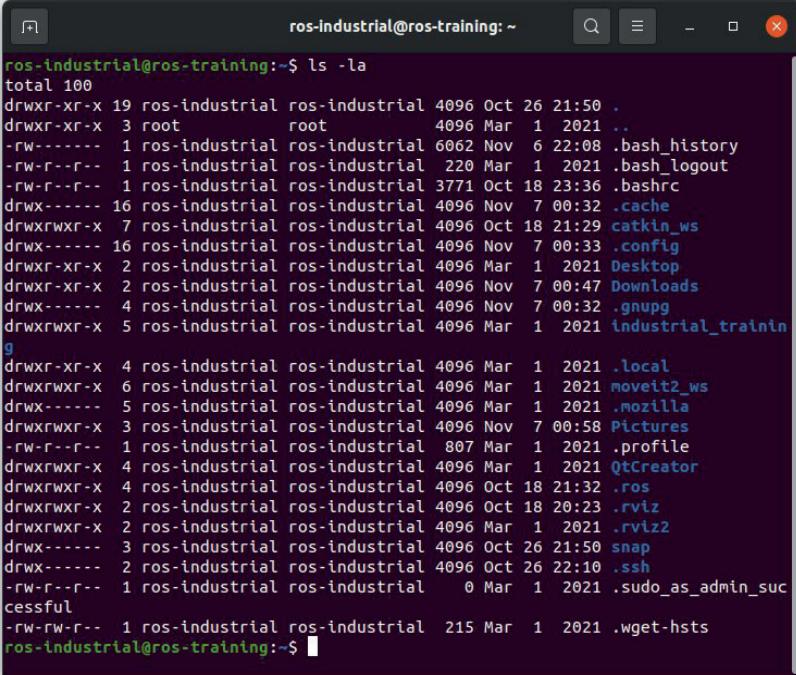


Рис. 1.7. Работа команды *ls*



```
ros-industrial@ros-training:~$ ls -la
total 100
drwxr-xr-x 19 ros-industrial ros-industrial 4096 Oct 26 21:50 .
drwxr-xr-x  3 root      root      4096 Mar  1  2021 ..
-rw-----  1 ros-industrial ros-industrial 6062 Nov  6 22:08 .bash_history
-rw-r--r--  1 ros-industrial ros-industrial 220 Mar  1  2021 .bash_logout
-rw-r--r--  1 ros-industrial ros-industrial 3771 Oct 18 23:36 .bashrc
drwx----- 16 ros-industrial ros-industrial 4096 Nov  7 00:32 .cache
drwxrwxr-x  7 ros-industrial ros-industrial 4096 Oct 18 21:29 catkin_ws
drwx----- 16 ros-industrial ros-industrial 4096 Nov  7 00:33 .config
drwxr-xr-x  2 ros-industrial ros-industrial 4096 Mar  1  2021 Desktop
drwxr-xr-x  2 ros-industrial ros-industrial 4096 Nov  7 00:47 Downloads
drwx----- 4 ros-industrial ros-industrial 4096 Nov  7 00:32 .gnupg
drwxrwxr-x  5 ros-industrial ros-industrial 4096 Mar  1  2021 industrial_trainin
g
drwxr-xr-x  4 ros-industrial ros-industrial 4096 Mar  1  2021 .local
drwxrwxr-x  6 ros-industrial ros-industrial 4096 Mar  1  2021 moveit2_ws
drwx----- 5 ros-industrial ros-industrial 4096 Mar  1  2021 .mozilla
drwxrwxr-x  3 ros-industrial ros-industrial 4096 Nov  7 00:58 Pictures
-rw-r--r--  1 ros-industrial ros-industrial 807 Mar  1  2021 .profile
drwxrwxr-x  4 ros-industrial ros-industrial 4096 Mar  1  2021 QtCreator
drwxrwxr-x  4 ros-industrial ros-industrial 4096 Oct 18 21:32 .ros
drwxrwxr-x  2 ros-industrial ros-industrial 4096 Oct 18 20:23 .rviz
drwxrwxr-x  2 ros-industrial ros-industrial 4096 Mar  1  2021 .rviz2
drwx----- 3 ros-industrial ros-industrial 4096 Oct 26 21:50 snap
drwx----- 2 ros-industrial ros-industrial 4096 Oct 26 22:10 .ssh
-rw-r--r--  1 ros-industrial ros-industrial     0 Mar  1  2021 .sudo_as_admin_suc
cessful
-rw-rw-r--  1 ros-industrial ros-industrial 215 Mar  1  2021 .wget-hsts
ros-industrial@ros-training:~$
```

Рис. 1.8. Полный список файлов и каталогов в домашней папке

ее без аргументов, она отобразит список файлов в текущей директории.

Дополнительными ключами для команды *ls* являются ключи *-al*, которые позволяют вывести список скрытых объектов в текущем каталоге. Пример использования команды представлен на рис. 1.8.

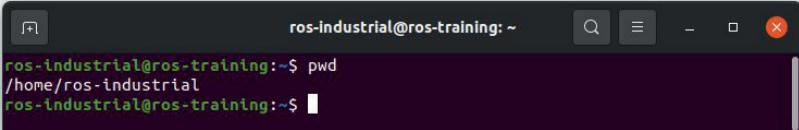
На рис. 1.8 показаны скрытые параметры объектов, общее количество, имя создателя, размер, дата создания, тип объектов, а также права на доступ.

#### 1.4.2. Команда *clear*

Часто возникают случаи, когда количество информации, запрашиваемой через терминальную программу, полностью наполняет окно программы. Для того чтобы очистить текущий экран терминала, используется команда *clear* (очистка).

### 1.4.3. Команда *pwd*

Выводит текущую директорию (*print working directory*). Пример вывода информации о текущем каталоге представлен на рис. 1.9.



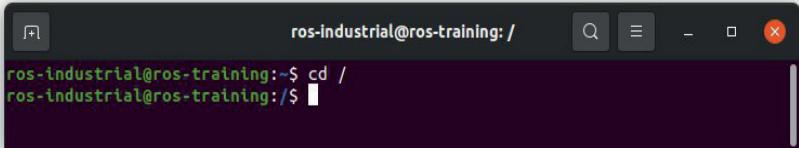
```
ros-industrial@ros-training:~$ pwd
/home/ros-industrial
ros-industrial@ros-training:~$
```

Рис. 1.9. Текущее местоположение пользователя

Для того чтобы не заблудиться в каталогах и определить свое местоположение в дереве каталогов, удобно определять текущую папку. Во время перехода между каталогами значение пути,озвращаемое командой *pwd*, будет изменяться.

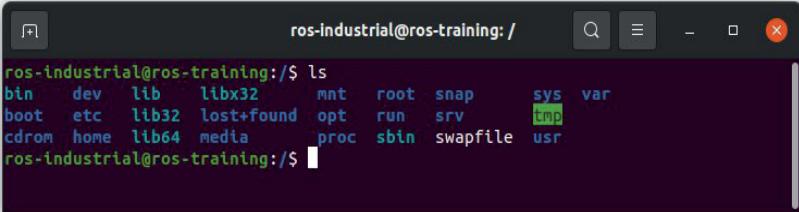
### 1.4.4. Команда *cd*

Переход между каталогами ОС *Linux* выполняется командой *cd* (*change directory* – сменить директорию). При использовании данной команды пользователь дополняет ее через пробел абсолютным или относительным путем – в зависимости от каталога, в который нужно попасть.



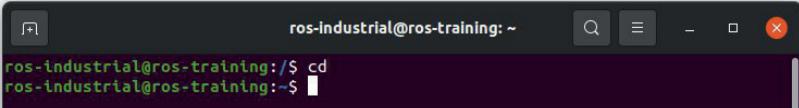
```
ros-industrial@ros-training:~$ cd /
ros-industrial@ros-training:/$
```

Рис. 1.10. Переход в корневой каталог



```
ros-industrial@ros-training:/$ ls
bin  dev  lib  libx32  mnt  root  snap    sys  var
boot  etc  lib32  lost+found  opt  run  srv    tmp
cdrom  home  lib64  media  proc  sbtn  swapfile  usr
ros-industrial@ros-training:/$
```

Рис. 1.11. Вывод содержимого корневого каталога

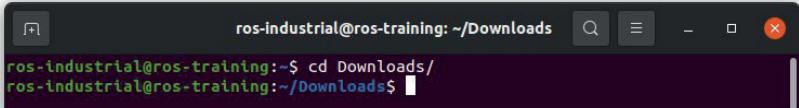


```
ros-industrial@ros-training:~/
```

```
ros-industrial@ros-training:~$ cd
```

```
ros-industrial@ros-training:~$
```

Рис. 1.12. Переход в домашний каталог

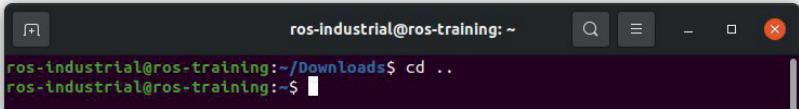


```
ros-industrial@ros-training:~/
```

```
ros-industrial@ros-training:~$ cd Downloads/
```

```
ros-industrial@ros-training:~/Downloads$
```

Рис. 1.13. Переход по относительному пути



```
ros-industrial@ros-training:~/Downloads$
```

```
ros-industrial@ros-training:~/Downloads$ cd ..
```

```
ros-industrial@ros-training:~$
```

Рис. 1.14. Переход на уровень выше каталога *Downloads*

Для возвращения в домашний каталог возможны два варианта: использовать абсолютный путь или команду *cd* без аргументов и ключей, данное свойство задано команде по умолчанию.

Кроме того, можно переходить в домашний каталог, дополняя команду именем пользователя (*cd ros-industrial*). Пример использования представлен на рис. 1.12.

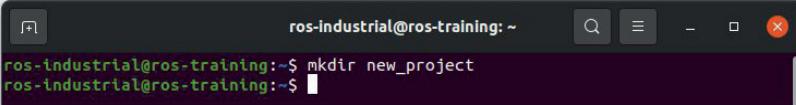
При переходе через относительный путь функционал команды *cd* сохраняется и по факту отличается только отсутствием символа корневого каталога. Ознакомиться с работой команды можно на рис. 1.13.

Выполнив команду *cd* символами двух точек, вы выполните переход на уровень выше. На рис. 1.14 демонстрируется переход на уровень выше директории *Downloads*.

Если вы находитесь на уровне корневого каталога и вызовете команду *cd ..*, то перехода на каталог выше не произойдет.

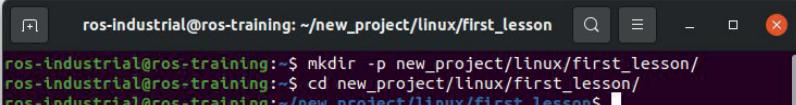
#### 1.4.5. Команда *mkdir*

Данная команда предназначена для создания каталогов (директорий), которые вам необходимы. Аналогом выступает создание



```
ros-industrial@ros-training:~$ mkdir new_project  
ros-industrial@ros-training:~$
```

Рис. 1.15. Создание пустой директории *new\_project*



```
ros-industrial@ros-training:~/new_project/linux/first_lesson$  
  
ros-industrial@ros-training:~$ mkdir -p new_project/linux/first_lesson/  
ros-industrial@ros-training:~$ cd new_project/linux/first_lesson/  
ros-industrial@ros-training:~/new_project/linux/first_lesson$
```

Рис. 1.16. Создание вложенных каталогов и переход в один из них

папки в ОС *Windows* или аналогичная операция в проводниках других ОС. Команда *mkdir* (*make directory* – создать директорию) создает пустую директорию в той директории, в которой вы сейчас находитесь.

Для примера создание директории *new\_project* представлено на рис. 1.15.

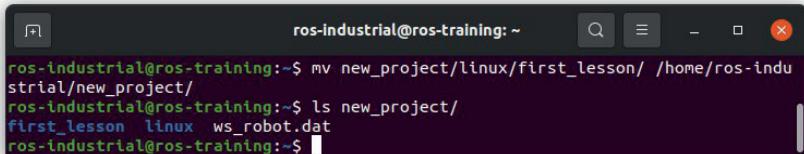
Директория создалась в домашнем каталоге, и при вводе команды *ls* она отобразится в командной строке.

Возникают задачи, когда нужно создать несколько вложенных друг в друга каталогов (директорий). Для решения данной задачи существует ключ *-p* для команды *mkdir*, который позволяет выполнить создание нескольких вложенных каталогов. Для работы данной команды необходимо указать через символ «/» набор имен каталогов. На рис. 1.16 можно ознакомиться с командой создания вложенных директорий.

Использование данного метода позволяет ускорить процесс создания каталогов (директорий). Если возникает вариант, что в пути создаваемых каталогов уже указаны существующие, то их содержимое не перезапишется, а произойдет дополнение новыми каталогами.

#### 1.4.6. Команда *mv*

Используется для перемещения (или переименования) файлов и директорий. Команда является сокращением слова *move* – переместить. Перемещение директории *first\_lesson* в директорию *new\_project* представлено на рис. 1.17.



```
ros-industrial@ros-training:~$ mv new_project/linux/first_lesson/ /home/ros-industrial/new_project/
ros-industrial@ros-training:~$ ls new_project/
first_lesson  linux  ws_robot.dat
ros-industrial@ros-training:~$
```

Рис. 1.17. Перемещение *first\_lesson* в *new\_project*

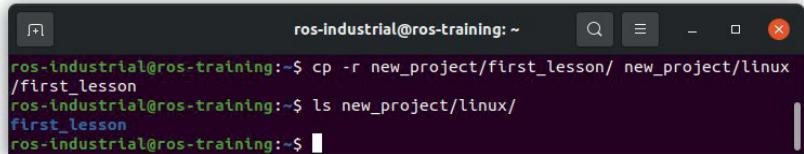
На рис. 1.17 представлен вариант использования команды с комбинацией абсолютного и относительного путей.

Синтаксис: *mv* пробел путь, где находится перемещаемый файл, пробел путь, куда следует переместить файл.

В процессе перемещения важно указывать только существующие директории, иначе возникнет ошибка: *directory does not exist* – директория не существует. Во время перемещения существует возможность сразу изменить имя объекта, который вы перемещаете.

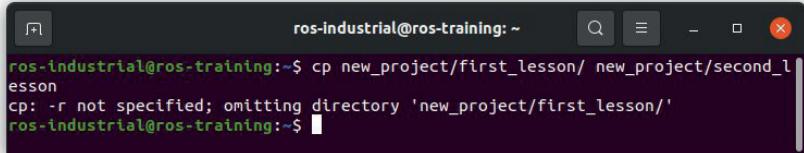
#### 1.4.7. Команда *cp*

Команда является сокращением от слова *copy* – копировать – и выполняет данный функционал для директорий и файлов системы. Ее синтаксис похож на синтаксис команды *mv*. Для копирования директорий следует использовать ключ *-r*, который указывает на факт копирования директорий. На рис. 1.18 демонстрируется работа команды по копированию директории.



```
ros-industrial@ros-training:~$ cp -r new_project/first_lesson/ new_project/linux/
first_lesson
ros-industrial@ros-training:~$ ls new_project/linux/
first_lesson
ros-industrial@ros-training:~$
```

Рис. 1.18. Копирование директории



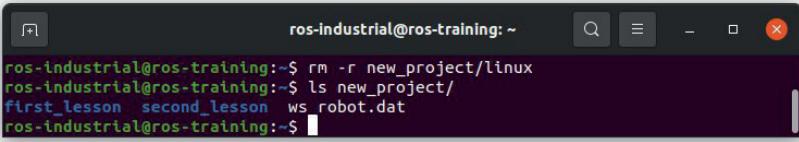
```
ros-industrial@ros-training:~$ cp new_project/first_lesson/ new_project/second_lesson
cp: -r not specified; omitting directory 'new_project/first_lesson/'
ros-industrial@ros-training:~$
```

Рис. 1.19. Вывод ошибки при копировании директории

Если у вас возникнут проблемы при использовании данных команд, то терминальная программа выведет вам сообщение об ошибке и дополнит его рекомендациями по исправлению. На рис. 1.19 показан пример неверного использования команды.

#### 1.4.8. Команда *rm*

Команда для удаления файлов и директорий. Происходит от сокращения слова *remove* – удалить. Для удаления директорий используется ключ *-r*, который позволяет удалить сразу несколько директорий. Пример использования команды *rm* представлен на рис. 1.20.

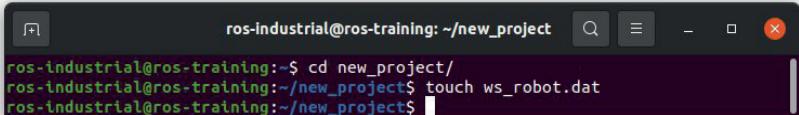


```
ros-industrial@ros-training:~$ rm -r new_project/linux
ros-industrial@ros-training:~$ ls new_project/
first_lesson second_lesson ws_robot.dat
ros-industrial@ros-training:~$
```

Рис. 1.20. Удаление директории Linux в директории *new\_project*

#### 1.4.9. Команда *touch*

Командой *touch* (коснуться) создается пустой файл. Например, создание пустого файла *ws\_robot.dat* С примером использования команды можно ознакомиться на рис. 1.21.



```
ros-industrial@ros-training:~/new_project$ cd new_project/
ros-industrial@ros-training:~/new_project$ touch ws_robot.dat
ros-industrial@ros-training:~/new_project$
```

Рис. 1.21. Создание пустого файла

Создание нового файла с аналогичным именем никак не влияет на существующий файл. Существующий файл не будет перезаписан или изменен, но также и не появится ошибка о существовании файла.

#### 1.4.10. Команда *cat*

Основной функционал команды *cat* (*catenate* – связывать) – выводить содержимое указанного файла в указанный источник, например на экран или в другой файл. На рис. 1.22 демонстрируется синтаксис команды *cat*.

```
ros-industrial@ros-training:~/new_project$ cat ws_robot.dat
```

Рис. 1.22. Вывод содержимого файла *ws\_robot.dat*

Функционал команды гораздо шире, чем просто вывод на экран содержимого. С дополнительными свойствами и возможностями команды можно ознакомиться, если ввести *man cat* – страницу документации на данную команду.

#### 1.4.11. Команда *nano*

На самом деле *nano* не команда, а консольный текстовый редактор. Это один из немногих способов отредактировать файл из командной строки (например, когда графический интерфейс не грузится или отсутствует). В некоторых ОС данный текстовый редактор отсутствует и требуется его установка. На рис. 1.23 показан пример использования команды *nano*.

После запуска вы сможете написать, что-либо в файле, например, «*new data in file*». Также внизу редактора будет панель (строки) с подсказками, какие горячие клавиши доступны для использо-

```
ros-industrial@ros-training:~/new_project$ nano ws_robot.dat
```

Рис. 1.23. Запуск команды / текстового редактора *nano*

```
GNU nano 4.8          ws_robot.dat          Modified
```

[ Read 2 lines ]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos  
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^L Go To Line

Рис. 1.24. Окно редактора *nano*

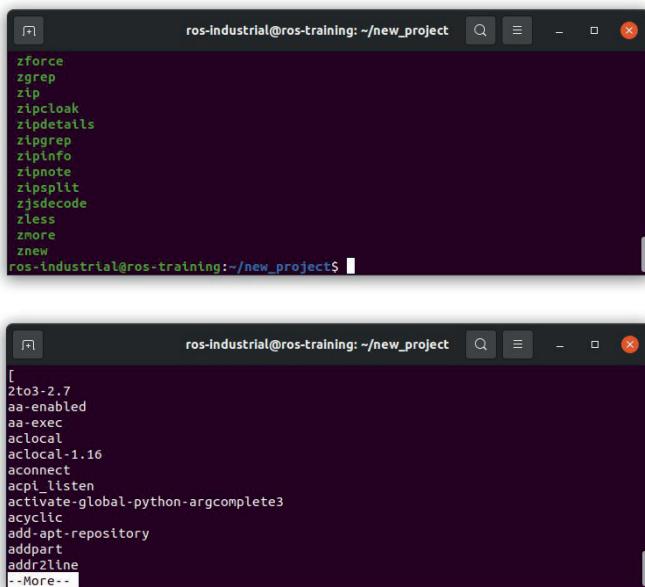
зования. Символ «^» означает клавишу *Ctrl*. Например, для выхода из редактора нужно набрать *Ctrl + X*, а затем либо подтвердить сохранение изменений, либо отвергнуть, нажав, соответственно, *y (yes)* или *n (no)*. Сохраните изменения. Окно текстового редактора изображено на рис. 1.24.

Редактор  *nano* удобен для несложных изменений в файлах, но совсем неудобен для работы над проектами с большим количеством файлов.

#### 1.4.12. | (Вертикальный разделитель)

Команда или конвейер перенаправления вывода – при помощи него вы можете вывод одной команды подать на вход другой. Например, если просто ввести команду *ls* в директории с большим количеством файлов, весь список пролетит по экрану терминала, что не позволит корректно проанализировать необходимую информацию. Вывод данных представлен на рис. 1.25.

Для того чтобы осуществлять поэкранный вывод, есть команда *more*, которая остановит вывод списка файлов при достижении края экрана.

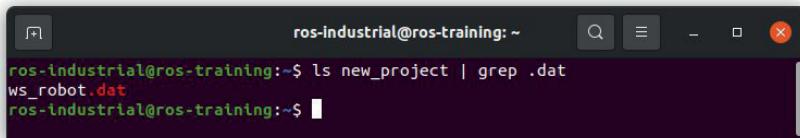


The figure consists of two vertically stacked screenshots of a terminal window. Both screenshots show a dark-themed terminal window with a light-colored text area. The title bar of both windows reads "ros-industrial@ros-training: ~/new\_project".  
The top screenshot shows the output of the command "ls" without any piping or redirection. The text in the terminal is:  
zforce  
zgrep  
zip  
zipcloak  
zipdetails  
zipgrep  
zipinfo  
zipnote  
zipsplit  
zjdecode  
zless  
zmore  
znew  
The bottom screenshot shows the same command "ls" followed by a vertical pipe character "|". The text in the terminal is:  
[  
2to3-2.7  
aa-enabled  
aa-exec  
aclocal  
aclocal-1.16  
aconnect  
acpl\_listen  
activate-global-python-argcomplete3  
acyclic  
add-apt-repository  
addpart  
addr2line  
--More--  
A small rectangular box highlights the word "More" at the bottom of the list.

Рис. 1.25. Использование | для комбинации команд

#### 1.4.13. Команда grep

Команда *grep* (*global regular expression print*) – одна из самых востребованных команд в терминале *Linux*. Она сортирует и фильтрует текст на основе правил. Утилита *grep* решает множество задач, в основном она используется для поиска строк, соответствующих строке в тексте или содержимому файлов. Также она может находить по шаблону или регулярным выражениям. Команда в считанные секунды найдет файл с нужной строкой, текст в файле или отфильтрует из вывода только пару нужных строк. Пример работы с фильтрованием данных представлен на рис. 1.26.



```
ros-industrial@ros-training:~$ ls new_project | grep .dat
ws_robot.dat
ros-industrial@ros-training:~$
```

Рис. 1.26. Вывод файла с расширением .dat

Синтаксис: *grep* [опции] шаблон [имя файла...] или: команда | *grep* [опции] шаблон.

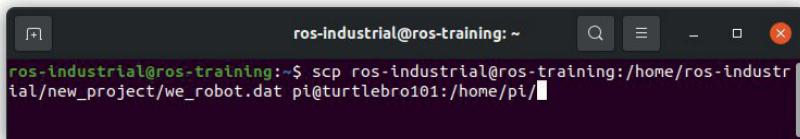
#### 1.4.14. Команда scp

Команда является аналогом команды *cp*, но позволяет копировать файлы с одного компьютера на другой по сети.

Синтаксис: *scp* опции пользователь1@хост1:файл пользователь2@хост2:файл. Подробно можно ознакомиться с примером на рис. 1.27.

В данном примере указано имя робота, который подключен к сети.

Данный набор команд является базовым и необходимым для выполнения основных задач при работе с ОС *Linux* и взаимодействия



```
ros-industrial@ros-training:~$ scp ros-industrial@ros-training:/home/ros-industrial/new_project/ws_robot.dat pi@turtlebro101:/home/pi/
```

Рис. 1.27. Пример использования команды *scp*

с роботом. Существует еще больший набор команд для работы с командной строкой, но данное пособие не предполагает их полного изучения.

## 1.5. Запуск программ

Командная строка является универсальным инструментом, который позволяет не только выполнять манипуляции с файлами и директориями, но и запускать системные или пользовательские программы.

Запуск команды выполняется с помощью набора символов «./», который запускает программу в текущем каталоге.

Также есть возможность запуска программы, используя полный путь до нее. Ознакомиться с работой команды можно на рис. 1.28.

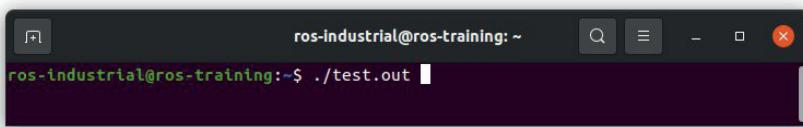


Рис. 1.28. Запуск пользовательской программы

В командной строке *Linux* существует возможность запускать программы в фоновом режиме. Для запуска в фоновом режиме используется символ & – амперсанд, который записывается в конце команды. Это очень удобное свойство, которое позволит в будущем запускать команды *ROS* без создания дополнительных окон.

## 1.6. Удаленное управление операционной системой

Ранее была рассмотрена команда терминала, которая позволяла перемещать файлы на удаленный рабочий стол – команда *scp*. Используя командную строку и специальный протокол взаимодействия, можно организовать взаимодействие с внешней ОС, которая может быть установлена как на роботе, находящемся в той же сети, что и виртуальная машина с ОС, так и на удаленном сервере. Большинство удаленных серверов построено на ОС *Linux*, а значит, для них действуют те же правила работы с командной строкой, а также возможно использование специальных протоколов удаленного доступа. В качестве такого протокола выступает протокол *SSH*.

*SSH* – защищенный протокол для удаленного доступа к компьютерам. Через *SSH* можно выполнять операции в командной строке компьютера, который физически находится в другом месте.

Для работы данного протокола необходимо наличие запущенного *SSH*-сервера на удаленной машине, а на ПК, с которого выполняется доступ, необходимо наличие *SSH*-клиента.

Удобство командной строки *Ubuntu* в том, что в ней по умолчанию установлен *SSH*-клиент, который легко запустить из терминальной программы.

*В основанных на Unix ОС (Linux, macOS) SSH-клиент обычно установлен в системе по умолчанию и достаточно открыть терминал. Начиная с Windows 10 SSH как отдельная команда также доступна для работы и ее синтаксис совпадает.*

Для подключения нужно указать адрес сервера и – опционально – имя пользователя и порт.

Синтаксис: *ssh username@remote\_host -p port*.

В примере, когда рассматривался синтаксис команды *scp*, было показано наличие робота с именем пользователя *pi* и именем компьютера *turtlebro101*. Данные параметры могут быть подставлены в команду *SSH* и использоваться для подключения к данному роботу. Работа утилиты представлена на рис. 1.29.

В данной команде не указан порт для подключения, так как используется значение по умолчанию для протокола – 22. Желательно при настройке вашей системы менять общеизвестные порты для подключения, чтобы дополнительно обезопасить свою систему.

Подключение к удаленной системе также возможно с использованием *IP*-адреса. *IP*-адрес выступает в качестве адреса пути назначения, как в реальной жизни адрес дома, в который есть необходимость отправить письмо. *IP*-адрес состоит из четырех значений (октетов), разделенных точками. Пример *IP*-адреса: 192.168.1.1

Существует несколько вариантов поиска *IP*-адреса устройства, к которому требуется подключиться. Если идет процесс подключения к удаленному серверу, то его адрес доступен в настройках системы хостинга.

*Хостинг* (англ. *hosting*) – услуга по предоставлению ресурсов для размещения информации на сервере (обычно Интернет).

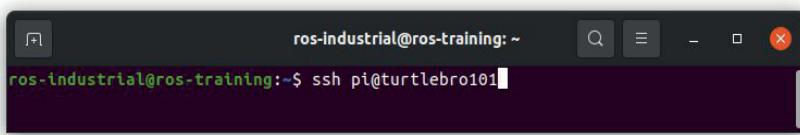


Рис. 1.29. Подключение к роботу через протокол *SSH*

Если стоит задача найти устройства в сети и *IP*-адрес неизвестен, одним из простых способов является способ просканировать сеть и найти нужное устройство для подключения. Существуют разные утилиты для сканирования сети, платные и бесплатные. Одной из самых простых в использовании является утилита *Angry IP Scanner*, представленная на рис. 1.30.

*Angry IP Scanner* – бесплатная *OpenSource* утилита для сканирования сети. Скачать ее можно по ссылке <https://angryip.org/download/#linux> – для ОС *Linux*. Кроме того, данная утилита разработана под большинство распространенных ОС и работает одинаково в каждой из них. Установка выполняется через .deb пакет – аналог .exe файла в ОС *Windows*. После установки утилита доступна как из панели меню, так и из командной строки.

Работа с утилитой не представляет больших трудностей. В поле *IP Range* задается диапазон адресов сети для сканирования. В основном этих параметров достаточно для сканирования сети и поиска, например, роботов. Сканирование начинается после нажатия на кнопку *Start*.

Основное окно программы содержит список найденных устройств в сети, а также их имена.

Отличительной особенностью адресов робота является наличие порта 8080, через который осуществляется вход в веб-интерфейс робота, а также *Hostname*, содержащий имя *turtlebro*.

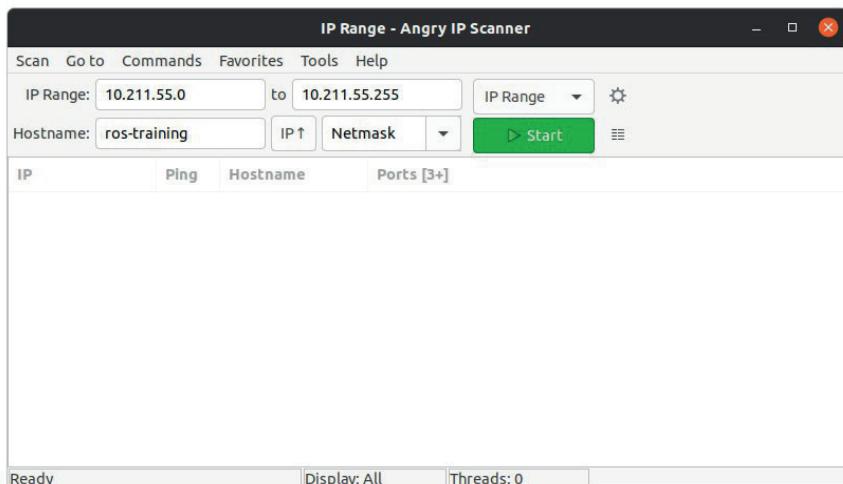


Рис. 1.30. Внешний вид утилиты для сканирования сети

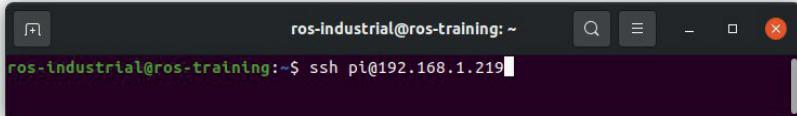
A screenshot of a terminal window titled "ros-industrial@ros-training: ~". The window has standard Linux-style window controls at the top. Inside the terminal, the command "ssh pi@192.168.1.219" is typed and is highlighted in green, indicating it is being processed or has been entered.

Рис. 1.31. Подключение к удаленному устройству с использованием IP-адреса

Найденный IP-адрес устройства используется для подключения по протоколу *SSH* и выглядит так, как показано на рис. 1.31.

При первом подключении будет выполнен процесс запроса ключа безопасности. По умолчанию его генерирует *SSH*-сервер, и ключ безопасности завязан на использовании пароля-фразы безопасности, который придумал пользователь. Если не было дополнительных манипуляций и настройки, то сервер сам генерирует ключ безопасности и помещает его в файл *SSH*-клиента (файл *know\_hosts*). Данный ключ является частью системы безопасности *fingerprint* («отпечатки пальцев»). Ваш компьютер запоминает эту комбинацию и сверяет ее при каждом новом соединении (если кто-то переустановит *SSH*-сервер или всю ОС или вообще заменит удаленный компьютер, сохранив его адрес). В текущей конфигурации используется простой вариант входа в систему без дополнительных (открытых и закрытых) ключей. Подключение выполняется с использованием пароля пользователя, к которому идет подключение.

При первом запуске система выведет сообщение:

*The authenticity of host <192.168.31.119 (192.168.31.119)> can't be established. ECDSA key fingerprint is SHA256:qPZXFXw///x7BqpPUHH8n6UhqtSN/5odOrRvij41aM. Are you sure you want to continue connecting (yes/no/[fingerprint])? yes*

Обязательно следует ввести полностью слово *yes*, чтобы сохранить настройки входа.

При работе с роботом пароль является стандартным *brobro*, данный пароль установлен внутри образа ОС для логина *pi*. При каждом открытии нового окна терминала требуется вводить пароль для входа в удаленную систему.

## 1.7. Пользователи и права пользователей

### 1.7.1. Основная информация о пользователях

Операционная система *Linux* предполагает наличие пользователя в системе. В отличии от *Windows* в *Linux* при создании поль-

зователя сразу вступают правила работы с суперпользователем, от имени которого можно выполнять ряд расширенных функций. Терминальная программа предоставляет нам инструменты ввода-вывода данных, утилиты, которые позволяют нам создавать и анализировать информацию и доступ к данным.

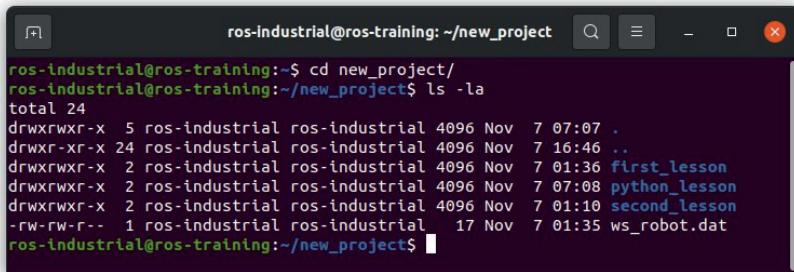
Запуск процесса в терминале предполагает запуск от имени некоторого пользователя. Интересно, что у разных пользователей разные права на изменение тех или иных данных. Например, обычный пользователь сможет получить информацию из каталога */etc*, но изменить его содержимое не сможет, так как у него недостаточно прав на выполнение данной операции.

Взаимодействие с файловой системой происходит через запуск тех или иных утилит, модифицирующих, создающих или анализирующих файловую структуру. Это значит, что, запуская, например, *touch*, мы стараемся процесс от своего имени, внутри которого запускается программа *touch*. Она, в свою очередь, создает файл (если его не было) и делает нас владельцем нового файла. В ОС *Linux* существуют два владельца: пользователь и группа.

У пользователей есть возможность понять, кто они в системе. Для этого есть команда *whoami* – (кто я). Команда выводит имя текущего пользователя, а также обладает рядом дополнительных возможностей, которые позволяют определить права текущего пользователя.

#### 1.7.2. Отображение информации о владельце файла и каталога

В пособии уже была рассмотрена команда *ls*, которая позволяла нам выводить информацию о содержимом каталога в виде списка. У данной команды есть ключи, позволяющие модифицировать ее ра-



```
ros-industrial@ros-training:~/new_project$ cd new_project/
ros-industrial@ros-training:~/new_project$ ls -la
total 24
drwxrwxr-x  5 ros-industrial ros-industrial 4096 Nov  7 07:07 .
drwxr-xr-x 24 ros-industrial ros-industrial 4096 Nov  7 16:46 ..
drwxrwxr-x  2 ros-industrial ros-industrial 4096 Nov  7 01:36 first_lesson
drwxrwxr-x  2 ros-industrial ros-industrial 4096 Nov  7 07:08 python_lesson
drwxrwxr-x  2 ros-industrial ros-industrial 4096 Nov  7 01:10 second_lesson
-rw-rw-r--  1 ros-industrial ros-industrial   17 Nov  7 01:35 ws_robot.dat
ros-industrial@ros-training:~/new_project$
```

Рис. 1.32. Вывод данных о содержимом директории

боту, что было показано в разделе описания данной команды. Рассмотрим подробнее информацию, которую возвращает данная команда. Запросим информацию из директории, созданную командой `mkdir`. Пример использования команды представлен на рис. 1.32.

Основным пользователем, от лица которого выполняются команды, является `ros-industrial`. Группа, к которой он относится, совпадает с именем пользователя. Существуют варианты конфигурации, когда группа и пользователь отличаются, что дает дополнительные возможности и правила настройки безопасности.

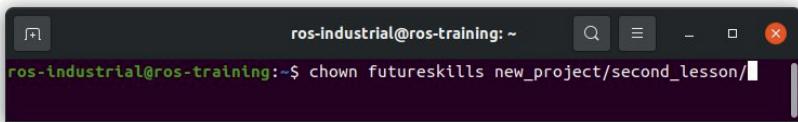
У пользователя есть возможность передать права другому владельцу. Как было сказано ранее, есть два варианта передачи прав: пользователю и группе. Рассмотрим оба варианта.

### 1.7.3. Команда `chown`

Данная команда позволяет изменять владельцев директории или файла. Важно, чтобы пользователь существовал. Если попытаться передать права несуществующему пользователю, возникнет ошибка. Ознакомиться с работой команды можно в примере на рис. 1.33.

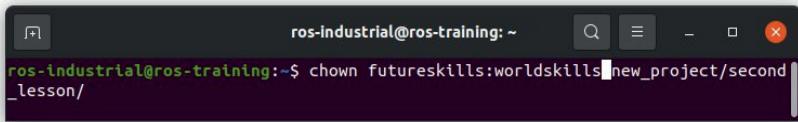
Команда `chown` имеет несколько опций, одна из которых особенно полезна: `-R`. Это позволяет вам рекурсивно устанавливать владельцем все вложенные директории и файлы.

Команда также позволяет изменить владельца группы. Отличительной особенностью команды в данном случае является дополнение имени пользователя именем группы, которые разделены символом «`:`». Пример использования команды представлен на рис. 1.34.



```
ros-industrial@ros-training:~$ chown futureskills new_project/second_lesson/
```

Рис. 1.33. Изменение владельца каталога



```
ros-industrial@ros-training:~$ chown futureskills:worldskills new_project/second_lesson/
```

Рис. 1.34. Изменение владельца группы

Нужно помнить, что пользователь может изменить группу файла только на ту, в которой он находится.

#### 1.7.4. Права доступа

Если внимательно рассмотреть рис. 1.32, то можно увидеть первый столбец. Данный столбец указывает информацию о нашем объекте:

1) первый символ – тип объекта (файл – «-», директория – «d» или иное);

2) от второго до десятого – права доступа.

В системе *Linux* права доступа задаются для трех категорий пользователей:

1) владельца – *Owner*;

2) пользователей, находящихся в той же группе, что и владелец, – *Group*;

3) остальных, тех, кто не попал в предыдущие две категории, – *Other*.

Для каждой категории задается свой набор прав. Права позволяют делать следующие операции:

1) *r* – читать. Возможность чтения данных из каталога, файла или иного объекта. При отсутствии данного символа в столбце прав у одной из категорий пропадает возможность читать и выводить информацию об объектах;

2) *w* – записывать. Возможность записи данных в каталог, файл или объект. По факту добавляет возможность изменять объекты;

3) *x* – исполнять. Возможность выполнять файл. Не рекомендуется давать права на исполнение директориям;

4) символ «-» означает отсутствия права.

Права доступа возможно задавать не только через символы *r*, *w*, *x*, но также используя числовые коды. Числовые коды выражены в десятичной системе счисления, каждая категория имеет свое двоичное число от 0 до 7, где 0 – нет никаких прав, а 7 – есть все права. Каждому символу *r*, *w*, *x* соответствует значение: в двоичной системе счисления 1, если в группе установлено право, и 0, если его нет. Подробнее с кодами доступа можно ознакомиться в табл. 1.1.

В примере на рис. 1.32 для нашего созданного файла *ws\_robot.dat* заданы права *-rw-rw--r--*, которым соответствует численный код 664. Для установки полных прав для всех категорий групп пользователей используется код 777.

Таблица 1.1

Таблица соответствия кодов и символов прав

Число	Доступ	Группа	Двоичный код
7	чтение, запись, исполнение	rwr	111
6	чтение, запись	rw-	110
5	чтение, исполнение	r-x	101
4	чтение	r--	100
3	запись, исполнение	-wx	011
2	запись	-w-	010
1	исполнение	--x	001
0	нет прав	---	000

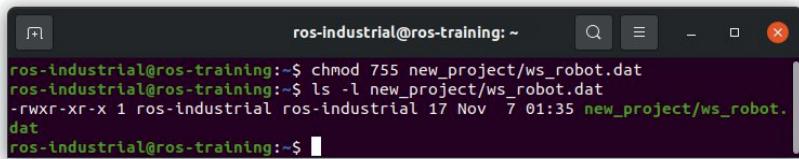
Использовать права, заданные по умолчанию, не всегда является возможным для работы с объектами в ОС. В командной строке *Linux* есть команда, которая позволяет изменять права доступа.

### 1.7.5. Команда chmod

Для управления правами используется команда *chmod*. Использование команды *chmod* позволяет устанавливать разрешения для пользователя (*user*), группы (*group*) и других (*other*).

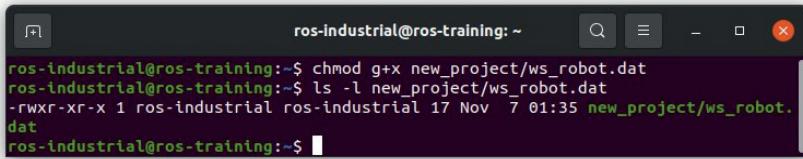
При настройке разрешений необходимо рассчитать значение числового кода. Если стоит задача установить чтение, запись и выполнение для пользователя, чтение и выполнение для группы, а также чтение и выполнение для других файлу, который был создан ранее, *ws\_robot.dat*, то команда будет выглядеть аналогично рис. 1.35.

Имеется возможность настройки существующих прав доступа с помощью символов *r*, *w*, *x*, что требует использования их в коман-



```
ros-industrial@ros-training:~$ chmod 755 new_project/ws_robot.dat
ros-industrial@ros-training:~$ ls -l new_project/ws_robot.dat
-rwxr-xr-x 1 ros-industrial ros-industrial 17 Nov  7 01:35 new_project/ws_robot.dat
ros-industrial@ros-training:~$
```

Рис. 1.35. Изменение прав доступа к файлу



```
ros-industrial@ros-training:~$ chmod g+x new_project/ws_robot.dat
ros-industrial@ros-training:~$ ls -l new_project/ws_robot.dat
-rwxr-xr-x 1 ros-industrial ros-industrial 17 Nov  7 01:35 new_project/ws_robot.dat
ros-industrial@ros-training:~$
```

Рис. 1.36. Добавление прав для группы *g*

де *chmod*. Данный режим называется *относительным*. При использовании *chmod* в относительном режиме работа выполняется с тремя индикаторами, чтобы указать, что нужно сделать:

- 1) указание, для кого происходит изменение разрешения. Выбор пользователя (*u*), группы (*g*) или другого (*o*);
- 2) применение оператора для добавления или удаления разрешений из текущего режима или установление их абсолютно. Операции «+» и «-»;
- 3) добавление символов *r*, *w* и *x*, чтобы указать, какие разрешения необходимо установить.

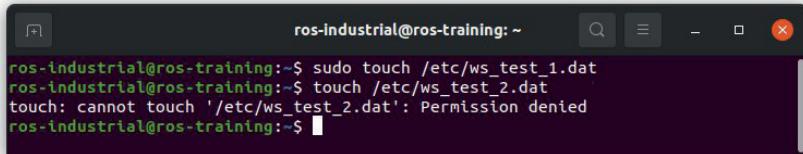
Пример использования команды представлен на рис. 1.36.

Возможна комбинация групп, которым необходимо установить отдельные права. Для этого используется символ «,».

Возможны случаи, когда необходимо запустить файлы от имени администратора без изменения прав доступа. В *Linux* существует группа пользователей, которая может выполнять команды от имени администратора системы при известном пароле. Такая группа называется *sudo*.

#### 1.7.6. Команда *sudo*

Данная команда (*substitute user & do – подменить пользователя и выполнить*) используется для выполнения команд от имени суперпользователя. Но пользоваться ею нужно аккуратно, так как можно повредить систему. На рис. 1.37 демонстрируется работа команды *sudo*.



```
ros-industrial@ros-training:~$ sudo touch /etc/ws_test_1.dat
ros-industrial@ros-training:~$ touch /etc/ws_test_2.dat
touch: cannot touch '/etc/ws_test_2.dat': Permission denied
ros-industrial@ros-training:~$
```

Рис. 1.37. Использование команды *sudo*

Использование данной команды позволяет перезапускать системные процессы, удалять и редактировать системные файлы, а также устанавливать новые пакеты и программы в систему.

#### **1.7.7. Команда *apt***

*Apt*, или *apt-get*, – это программа менеджер пакетов, который служит для установки и удаления программ (пакетов), обновления системы. В качестве ее инструкции используется слово *install*.

В результате выполнения команды у вас установится программа файловый менеджер. При запуске от имени суперпользователя система запросит ввод пароля, а также подтверждение согласия на установку.

### **1.8. Практические задания**

**Задание 1.** Выполните установку менеджера пакетов *pip* для языка *Python3*, используя команду *apt*.

**Задание 2.** Создайте папку *catkin\_ws* в домашней директории.

**Задание 3.** Создайте файл *data.dat*, в который запишите Ф. И. О. (можно латиницей).

**Задание 4.** Измените права для файла *data.dat*, чтобы его мог запускать только суперпользователь.

## 2. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON

*Python* – язык программирования общего назначения. Имеет библиотеки и фреймворки для разных направлений: веб-разработки (например, *Django* и *Bottle*), научных и математических вычислений (*Orange*, *SymPy*, *NumPy*), для настольных графических пользовательских интерфейсов (*Pygame*, *Panda3D*).

Работать на *Python* приятно и легко, потому что он позволяет сфокусироваться на задаче, а не пробираться сквозь сложный синтаксис.

### 2.1. История Python

Разработка *Python* началась Гвидо Ван Россумом в конце 1980-х гг., и в феврале 1991 г. вышла первая версия.

### 2.2. Зачем создан Python

В конце 1980-х гг. Гвидо Ван Россум работал над группой ОС *Amoeba*. Он хотел использовать интерпретируемый язык, такой как ABC, язык, имеющий простой и доступный в понимании синтаксис, который мог бы получить доступ к системным вызовам *Amoeba*. Поэтому он решил создать масштабируемый язык. Это привело к созданию нового языка, у которого позже появилось название *Python*.

Почему выбрали название *Python*. Нет, он не назван в честь змеи. В 1970-х гг. Россум был фанатом комедийного сериала *Monty*

Таблица 2.1

#### Версии языка программирования и даты релизов

Версия и описание	Дата выпуска
Python 1.0 (первый стандартный выпуск) Python 1.6 (последняя выпущенная версия)	Январь 1994 – 5 сентября 2000
Python 2.0 (представлены списки) Python 2.7 (последняя выпущенная версия)	16 октября 2000 – 3 июля 2010
Python 3.0 (сделан акцент на удаление дублирующих конструкций и модулей) Python 3.10 (текущая релизная версия на момент написания пособия)	3 декабря 2008 – настоящее время

*Python's Flying Circus* («Летающий цирк Монти Пайтона»). Слово *Python* было взято из названия. Информация о версиях языка и временных рамках работы представлена в табл. 2.1.

### 2.3. Особенности программирования на Python

1. Язык простой, легкий и доступный в изучении, у *Python* читаемый синтаксис. Гораздо проще читать и писать программы на *Python* по сравнению с другими языками, такими как *C++*, *Java*, *C#*. *Python* делает программирование интересным и позволяет сфокусироваться на решении, а не на синтаксисе. Для новичков отличный выбор начать изучение программирования с *Python*.

2. Бесплатный и с открытым кодом. Можно свободно использовать и распространять программное обеспечение, написанное на *Python*, даже для коммерческого использования. Вносить изменения в исходный код *Python*. Над *Python* работает большое сообщество, постоянно совершенствуя язык с каждой новой версией.

3. Портативность. Перемещайте и запускайте программы на *Python* с одной платформы на другую без каких-либо изменений кода. Код работает практически на всех платформах, включая *Windows*, *Mac OS X* и *Linux*. Существуют версии *MicroPython*, созданные для работы на микроконтроллерах, ограниченных в ресурсах и памяти.

4. Масштабируемый и встраиваемый. Предположим, что приложение требует повышения производительности. Вы можете с легкостью комбинировать фрагменты кода на *C/C++* и других языках вместе с кодом *Python*. Это повышает производительность приложения, а также дает возможность написания скриптов, создание которых на других языках требует больше настроек и времени.

5. Высокоуровневый, интерпретируемый язык. В отличии от *C/C++*, с *Python* вам не нужно беспокоиться о таких сложных задачах, как «сборка мусора» или управление памятью. Также, когда вы запускаете код *Python*, он автоматически преобразует ваш код в язык, который понимает компьютер. Не нужно думать об операциях более низкого уровня.

6. Стандартные библиотеки для решения общих задач. *Python* укомплектован рядом стандартных библиотек, что облегчает жизнь программиста, так как нет необходимости писать весь код самостоятельно. Например, чтобы подключить базу данных *MySQL* на веб-сервер, используйте библиотеку *MySQLdb*, добавляя ее в код строкой *import MySQLdb*. Стандартные библиотеки в *Python* про-

тестированы и используются тысячами людей. Поэтому будьте уверены, они будут работать именно так, как описано в документации.

7. Объектно-ориентированный. В *Python* все объект.

## 2.4. Приложения на Python

Использование языка программирования *Python* позволяет писать приложения разных уровней сложности: от простых скриптов, выполняющих вывод данных, до сложных клиент-серверных приложений и игр. Рассмотрим основные виды приложений, которые возможно создать с использованием языка программирования.

### 2.4.1. Веб-приложения

Создавайте масштабируемые веб-приложения с помощью фреймворков и CMS (систем управления содержимым) на *Python*. Популярные платформы для создания веб-приложений: *Django*, *Flask*, *Pyramid*, *Plone*, *Django CMS*. Кстати, сайты *Mozilla*, *Reddit*, *Instagram* и *PBS* написаны на *Python*.

### 2.4.2. Научные и цифровые вычисления

У *Python* много библиотек для научных и математических вычислений. Есть библиотеки, такие как *SciPy* и *NumPy*, которые используются для общих вычислений. И специальные библиотеки, такие как *EarthPy* (для науки о Земле), *AstroPy* (для астрономии) и т. д. *Python* – язык номер один в машинном обучении, анализе и сборе данных и прототипировании нейросетей.

### 2.4.3. Создание прототипов программного обеспечения

*Python* менее эффективен в сравнении с компилированными языками, такими как *C++* и *Java*. Это не очень практичный выбор, если ресурсы ограничены и при этом нужна максимальная эффективность. Тем не менее *Python* – прекрасный язык для создания прототипов. Используйте *Rygame* (библиотеку для создания игр), чтобы создать прототип игры. Если прототип понравился, используйте вставки на *C/C++* для повышения производительности.

1. Простой элегантный синтаксис. Программировать на *Python* интересно. Легче понять и написать код на *Python*. Почему? Синтаксис кажется естественным и простым. Одна из самых простых программ на языке *Python* “Hello, World!” выглядит как вывод текста на экран: `print("Hello, World!")`.

2. Не слишком строгий. Не нужно определять тип переменной в *Python*. Нет необходимости добавлять «;» в конце строки. *Python* принуждает следовать методам написания читаемого кода (например, одинаковым отступам). Эти мелочи могут значительно облегчить обучение новичкам.

3. Выразительность языка. *Python* позволяет писать программы с большей функциональностью и с меньшим количеством строк кода.

4. Большое сообщество и поддержка. У *Python* большое сообщество с огромной поддержкой. Множество активных форумов в Интернете, где помогают, когда возникают вопросы.

Ответы на основные вопросы, возникающие в процессе изучения, можно найти в официальной документации на язык по ссылке: <https://docs.python.org/3/>. Дополнительную информацию можно также найти на специализированных форумах и ветках блогов:

- *Python* на Хабре (<https://habr.com/ru/hub/python/>).
- Вопросы о *Python* на Тостер (<https://qna.habr.com/tag/python/info>).
- Вопросы о *Python* на *Stack Overflow* (<https://stackoverflow.com/tags/python>).

Это только часть представленных ресурсов для поиска ответов, в поисковых системах существует намного больше различных вариантов ресурсов и площадок, где возможно найти ответ.

5. Большой выбор курсов для обучения. По *Python* есть огромное количество курсов в Интернете, как платных, так и бесплатных. Вот некоторые из них:

- <https://www.coursera.org/learn/python-osnovy-programmirovaniya>
  - <https://stepik.org/course/67/syllabus>
  - <https://skillbox.ru/course/profession-python/>
  - <https://netology.ru/programs/python>
  - <https://compscicenter.ru/courses/python/2015-autumn/classes/>

Существуют площадки для подготовки специалистов разных уровней и разных возрастных категорий, например, проект компании «Яндекс» – Лицей Академии Яндекса, где обучение начинается с 8-го класса. Кроме того, язык *Python* используется во многих профилях Национальной технологической олимпиады, где его применяют участники в возрасте от 12 до 25 лет.

## 2.5. Инструменты для разработки

В данном учебном пособии уже рассмотрен один из вариантов написания кода программ – редактор *nano*. Его функционал прост и не предназначен для написания сложных программных решений. Для эффективной работы и разработки будет использован *Visual Studio Code (VSCode)* – кросс-платформенный редактор кода. Он обладает возможностями интеллектуального написания кода (подсказки функций, классов и объектов, подсветка синтаксиса, автоматическое форматирование кода, подсветка ошибок, а также наличие инструментов отладки).

Установка *VSCode* возможна при скачивании установочного файла с официального сайта (<https://code.visualstudio.com/Download>). Учитывая умение работать с командной строкой, возможно установить *VSCode*, используя менеджер пакетов *snap*. Использование команды *snap* и установка среды разработки представлены на рис. 2.1.

Пакет скачивается и устанавливается внутри ОС. Запуск редактора возможен как через терминал с помощью команды *code*, так и через ярлык в панели меню.

При первом запуске появляется меню настройки среды разработки, которое можно проигнорировать и перейти к написанию кода и созданию файлов на языке *Python*. Ознакомиться с окном программного обеспечения *VSCode* можно на рис. 2.2.

Создание файла для написания скриптов на языке *Python* выполняется через меню *File* → *New File* или сочетанием клавиш *Ctrl + N*. Рекомендуется сразу после создания файла его сохранить, дать ему уникальное имя и поместить в папку, где лежат разрабатываемые проекты. Имя позволит найти файл для дальнейшей работы. При сохранении следует указать расширение файла *\*.py*, которое является расширением файлов языка *Python*. Не следует давать имена файлам, которые являются названием встроенных в язык конструкций, функций, классов, объектов – это вызовет

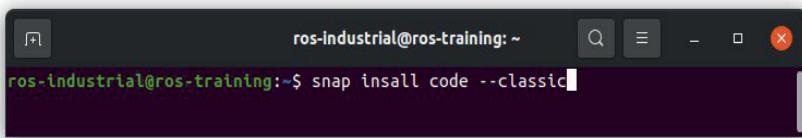


Рис. 2.1. Установка Visual Studio Code

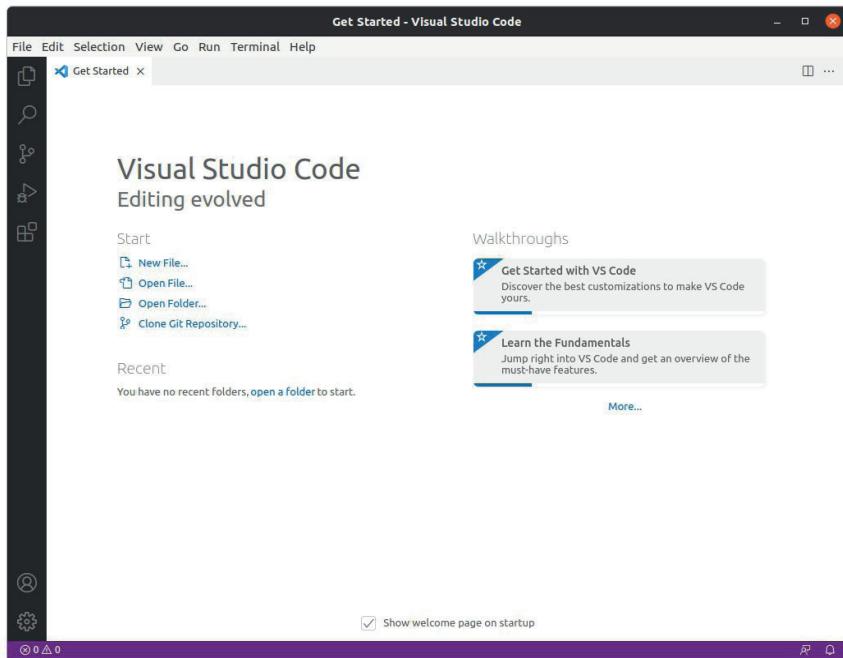


Рис. 2.2. Окно среды разработки Visual Studio Code

ошибки и нестабильность работы как интерпретатора, так и программ пользователя.

## 2.6. Интерпретатор Python

Для того чтобы превратить код, написанный на языке *Python*, в последовательность команд для процессора, нужно передать этот код интерпретатору *Python* – специальной программе, которая преобразует написанный и понятный человеку текст кода в команды машинного кода, понятного компьютеру.

Чтобы запустить программу на *Python*, нужно вызвать интерпретатор *Python* из директории с программой и передать ему название файла с программой.

В текущий момент существуют одновременно две версии языка *Python* (вторая и третья версии). Команда *python* вызывает выполнение интерпретатора второй версии, а команда *python3* – третьей версии.

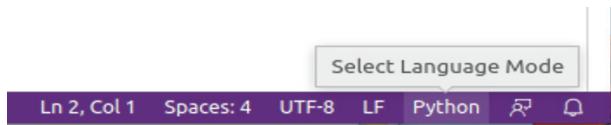


Рис. 2.3. Выбор интерпретатора языка

По умолчанию *Python* уже встроен в систему *Linux*. Существует возможность выбирать версию интерпретатора, который будет выполнять написанный код. Данное меню можно вызвать, если нажать в правом нижнем углу редактора *VSCode*. Информация о выборе интерпретатора представлена на рис. 2.3.

В появившемся окне следует выбрать необходимую конфигурацию и после этого выполнить запуск программы.

## 2.7. Простые программы на языке Python

### 2.7.1. Вывод данных на экран

Как было сказано ранее, писать код на языке *Python* очень просто и интуитивно понятно. Создание любой программы начинается с создания файла и сохранения с расширением `*.py`.

Основным инструментом для отображения работы программ, написанных на языке *Python*, является функция `print()`, которая выводит данные в скобках на экран. Описание интерфейса программы представлено на рис. 2.4.

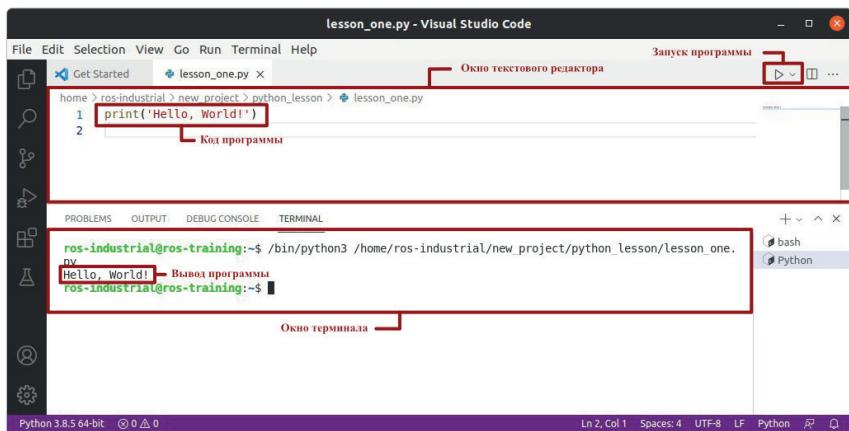


Рис. 2.4. Окно редактора Visual Studio Code

В верхней части расположено окно текстового редактора, которое содержит код программы: `print('Hello, World!')`.

В нижней части экрана представлен терминал ОС *Linux*, в котором будет выполнен вывод данных программы. Данный терминал обладает всеми свойствами, что и терминальная программа ОС *Ubuntu*, так как это та же оболочка (*bash*).

После запуска программы произойдет вывод в окне терминала.

### 2.7.2. Синтаксис языка программирования

Стоит отметить основную особенность синтаксиса *Python*. Он не содержит операторных символов (*begin..end* в *Pascal* или скобок `{..}` в *C*), вместо этого блоки выделяются отступами: пробелами или табуляцией, а вход в блок из операторов осуществляется двоеточием. Важно, чтобы в вашем редакторе табуляция имела четыре пробела, иначе это вызовет ошибку работы интерпретатора. Количество пробелов следует смотреть в правом нижнем углу редактора кода *Spaces*: 4. Однострочные комментарии начинаются с «`#`», многострочные – начинаются и заканчиваются тремя двойными кавычками «`"""`». На рис. 2.5 демонстрируется пример программы.

В данной программе комбинированный вывод результата. Вывод значения переменной, а также вывод текста.

Изучение синтаксиса языка *Python* будет выполняться в процессе изучения его конструкций, возможностей и особенностей.

The screenshot shows the Visual Studio Code interface. The top bar displays 'lesson\_one.py - Visual Studio Code'. The left sidebar has icons for file operations, search, and other tools. The main area shows the code editor with the following content:

```
1 a = 1
2 b = 2
3 c = a + b
4 print('c =', c)
5
```

Below the code editor are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab shows the following command-line session:

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_one.py
c = 3
ros-industrial@ros-training:~$
```

The status bar at the bottom indicates 'Python 3.8.5 64-bit', 'Ln 4, Col 10', 'Spaces: 4', 'UTF-8', 'LF', 'Python', and icons for file operations.

Рис. 2.5. Пример программы

Запускать программы также можно, используя командную строку. Для этого нужно перейти в папку с файлом, который вы подготовили, и выполнить запуск.

Запуск выполняется с использованием *Python* – аналогично использованию команды. Информация о запуске программы попадает в область вывода консоли, программной среды *VSCode*. Пример, демонстрирующий вывод на экран, показан на рис. 2.6.

Существует способ запуска программ из терминала, без прямого указания команды *python* перед исполняемым файлом. В *Linux* реализован механизм «шебанг» – последовательность из двух символов: решетки и восклицательного знака «#!» в начале файла скрипта. Когда скрипт с шебангом выполняется как программа в *Unix*-подобных ОС, загрузчик программ рассматривает остаток строки после шебанга как имя файла программы-интерпретатора. Загрузчик запускает эту программу и передает ей в качестве параметра имя файла скрипта с шебангом. Например, если полное имя файла скрипта «*path/to/script*» и первая строка этого файла. Для нашего варианта код программы требует дополнения в начале строки: `#!/usr/bin/env python3`.

Также файлу необходимо установить специальный флаг, информирующий, что файл может быть исполнен. Для этого существует команда *Linux chmod*. Пример использования шебанга представлен на рис. 2.7.

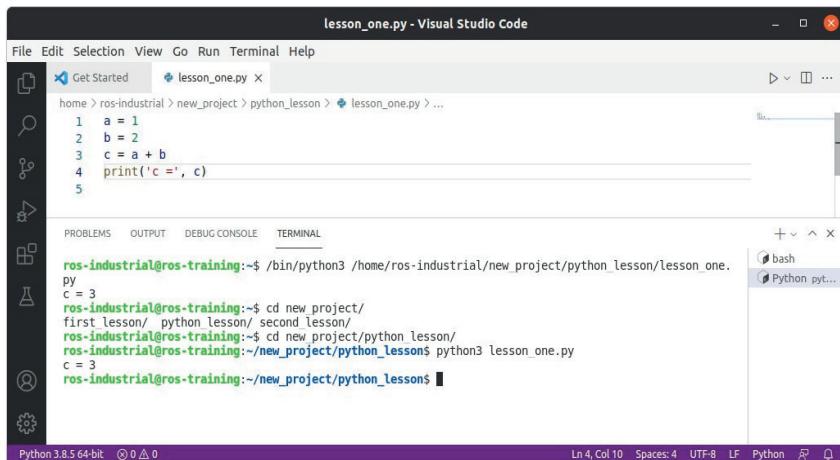


Рис. 2.6. Запуск программы из терминала

The screenshot shows the Visual Studio Code interface with the title bar "lesson\_one.py - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help. The terminal tab is active, showing the following command-line session:

```
ros-industrial@ros-training:~$ cd new_project/python_lesson/
ros-industrial@ros-training:~/new_project/python_lesson$ chmod +x lesson_one.py
ros-industrial@ros-training:~/new_project/python_lesson$ ./lesson_one.py
Sharebug
ros-industrial@ros-training:~/new_project/python_lesson$
```

The status bar at the bottom indicates "Ln 3, Col 1 Spaces: 4 UTF-8 LF Python".

Рис. 2.7. Запуск программы с использованием шебанга

При запуске программы не требуется указание названия интерпретатора.

## 2.8. Ввод и вывод данных

### 2.8.1. Вывод данных

Вывод данных в *Python* реализован с помощью функции *print()*. Внутри скобок прописываются объекты, которые нужно вывести на экран.

The screenshot shows the Visual Studio Code interface with the title bar "lesson\_one.py - Visual Studio Code". The terminal tab is active, showing the following command-line session:

```
ros-industrial@ros-training:~/new_project/python_lesson$ ./lesson_one.py
9
18
18.0
18.12
18.125
18.125
1
ros-industrial@ros-training:~$
```

The status bar at the bottom indicates "Ln 3, Col 20 Spaces: 4 UTF-8 LF Python".

Рис. 2.8. Пример программы для вывода данных

Полный синтаксис функции `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`. Значения в скобках называются *аргументами функции*. В процессе написания кода происходит передача аргументов и функция определяет, что является частью вывода, а что является частью форматирования вывода. Специальные слова `sep`, `end`, `file`, `flush` называются *именованными аргументами*, и для их изменения нужно напрямую выполнить их вызов и задать им значение. Пример программы пользователя с использованием основных математических операторов представлен на рис. 2.8.

### 2.8.2. Ввод данных

Для ввода данных используется функция `input()`. Она считывает все, что введет пользователь с клавиатуры до нажатия клавиши `Enter`.

Простая программа, которая будет спрашивать у пользователя ввести имя. Пример программы представлен на рис. 2.9.

Следующий шаг – написать программу, которая считывает два числа и выводит их сумму. Для этого считаем два числа и сохраняем их в переменные `a` и `b`, пользуясь оператором присваивания `==`. Слева от оператора присваивания `==` в программах на *Python* ставится имя переменной (подробнее про операторов см. далее). Справа от оператора присваивания ставится любое выражение (значение, математическая формула, символ, строка, текст, результат

```
lesson_one.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_one.py ...
home > ros-industrial > new_project > python_lesson > lesson_one.py > ...
1 print('Как вас зовут?')
2 name = input() # считываем строку и "кладём" её в переменную name
3 print('Привет, ' + name + '!')

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_one.py
Как вас зовут?
Pavel
Привет, Pavel!
ros-industrial@ros-training:~$
```

Python 3.8.5 64-bit ① 0 △ 0 Ln 3, Col 15 Spaces: 4 UTF-8 LF Python ⚙ ⌂

Рис. 2.9. Приглашение и ввод данных

работы функции, имя файла и т. д.). После выполнения операции присваивания в нашем случае переменная станет указывать на результат умножения переменных *a* и *b*.

Отличительной особенностью языка *Python* является то, что переменные на самом деле не являются переменными. Это некоторые имена, связанные с объектами. В языке *Python* почти все конструкции являются объектами, с которыми пользователь выполняет взаимодействие. Преобразование типов данных представлено в примере программы на рис. 2.10.

Программа выведет значения, которые пользователь присвоил объектам *a* и *b*. Важно отметить, что функция *input()* читает данные из потока ввода в виде символов и строк. Следовательно, когда вы будете выполнять сложение значений, то результат представит собой «слепление» двух строковых значений. Это происходит потому, что по умолчанию данные переменные относятся к типу *str()* – *string* (строка). В данном случае операция сложения имеет иное название – конкатенация символов и строк.

Объекты в языке *Python* имеют тип, который задается им при создании. Существуют простые типы данных, которые мы используем постоянно: строки хранятся в объектах типа *str()*, целые числа хранятся в объектах типа *int()*, дробные числа (вещественные числа) – в объектах типа *float()*. Тип объекта позволяет понять, какие действия можно с ним выполнять. Например, строки можно складывать, но нельзя перемножать.

```
lesson_one.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_one.py ...
home > ros-industrial > new_project > python_lesson > lesson_one.py > ...
1 a = input("Введите a = ")
2 b = input("Введите b = ")
3 s = a + b
4 print(s)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~/bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_o
ne.py
Введите a = 1
Введите b = 2
12
ros-industrial@ros-training:~$
```

Python 3.8.5 64-bit    Ln 2, Col 24    Spaces: 4    UTF-8    LF    Python    ⌂

Рис. 2.10. Присваивание значений объектам

```
lesson_one.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_one.py
home > ros-industrial > new_project > python_lesson > lesson_one.py > ...
1 a = int(input("Введите a = "))
2 b = int(input("Введите b = "))
3 s = a + b
4 print(s)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_o
ne.py
Введите a = 1
Введите b = 2
3
ros-industrial@ros-training:~$
```

Python 3.8.5 64-bit ⑧ 0 ▲ 0 Ln 2, Col 31 Spaces: 4 UTF-8 LF Python ⚙

Рис. 2.11. Преобразование данных

Для преобразования одного типа данных в другой необходимо использовать встроенные функции. Например, преобразование данных из функции *input()* в целые числа реализуется с помощью функции *int()*. На рис. 2.11 демонстрируется работа преобразования типов данных.

Важно помнить, что вся строка должна состоять из чисел, иначе возникнет ошибка. Эта ошибка вызвана процессом преобразования данных из одного типа в другой, так как все типы данных имеют свой принцип хранения в памяти.

## 2.9. Математические операторы и библиотеки

### 2.9.1. Целые числа и операции над ними

Одним из простых типов данных является целое число, как со стороны работы с ним, так и со стороны представления в памяти. В *Python* для целых чисел определены операции *+*, *-*, *\** и *\*\**. Операция деления */* для целых чисел возвращает вещественное число (значение типа *float*). Также функция возведения в степень возвращает значение типа *float*, если показатель степени – отрицательное число.

В *Python* определены специальные операции для вычислений целого и остатка от деления. Целочисленное деление выполняется с отбрасыванием дробной части, которая обозначается *//*. Она возвращает целое число: целую часть частного. Остаток от деления

The screenshot shows a Visual Studio Code interface. In the center, there's a code editor with the file `lesson_one.py` open. The code contains three print statements: `print(17 / 3)`, `print(17 // 3)`, and `print(17 % 3)`. The output terminal below shows the results: 5.666666666667, 5, and 2 respectively. The status bar at the bottom indicates Python 3.8.5 64-bit, and the bottom right shows the file is saved with 0 changes.

Рис. 2.12. Операции деления в языке Python

обозначается символом %. На рис. 2.12 изображено окно среды разработки VSCode, в котором продемонстрирована работа операторов деления Python.

Следует помнить, что при нахождении остатка от деления, при значении делимого меньше делителя, операция возвращает значение делимого.

### 2.9.2. Действительные числа

Действительные числа – это все числа числовой прямой: целые, дробные, иррациональные, отрицательные и положительные. В Python действительные числа имеют тип `float()`. Они ограничены по числу знаков после запятой (не более 16), но для наших типовых задач данной точности вполне хватает. Пример использования преобразования в действительные числа представлен на рис. 2.13.

Действительные (вещественные) числа представляются в виде чисел с десятичной точкой (а не запятой, как принято при записи десятичных дробей в русских текстах).

Для записи очень больших или очень маленьких по модулю чисел используется так называемая запись с плавающей точкой (также называемая научной записью). В этом случае число представляется в виде некоторой десятичной дроби, называемой мантиссой, умноженной на целочисленную степень десяти (порядок). Например, расстояние от Земли до Солнца равно  $1.496 \cdot 10^{11}$ , а масса молекулы воды  $2.99 \cdot 10^{-23}$ .

The screenshot shows a Visual Studio Code interface with the title bar 'lesson\_one.py - Visual Studio Code'. The left sidebar contains icons for file operations like Open, Save, and Find. The main editor area has a tab for 'lesson\_one.py'. Below the editor is a terminal window showing the command line output:

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_one.py
1.5
1.5
ros-industrial@ros-training:~$
```

The status bar at the bottom indicates 'Python 3.8.5 64-bit' and other settings.

Рис. 2.13. Ввод и вывод дробного числа

Числа с плавающей точкой в программах на языке *Python*, а также при вводе и выводе записываются так: сначала пишется мантисса, затем пишется буква «*e*», затем пишется порядок. Пробелы внутри этой записи не ставятся. Например, указанные ранее константы можно записать в виде 1.496e11 и 2.99e-23. Перед самим числом также может стоять знак «минус». В *Python* также есть возможность разделения чисел в сотнях, тысячах, миллионах и т.д. символом нижнего подчеркивания для более удобного восприятия ввода данных. С работой программы, которая выводит числа с плавающей точкой, можно ознакомиться на рис. 2.14.

The screenshot shows a Visual Studio Code interface with the title bar 'lesson\_one.py - Visual Studio Code'. The left sidebar contains icons for file operations like Open, Save, and Find. The main editor area has a tab for 'lesson\_one.py'. Below the editor is a terminal window showing the command line output:

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_one.py
1.3e12
1.134_000
print(x)
print(y)
5
ros-industrial@ros-training:~$
```

The status bar at the bottom indicates 'Python 3.8.5 64-bit' and other settings.

Рис. 2.14. Ввод данных с плавающей точкой и больших чисел

Результатом операции деления / всегда является действительное число (float), в то время как результатом операции // является целое число (int). Преобразование действительных чисел к целому производится с округлением в сторону нуля, то есть  $\text{int}(1.7) == 1$ ,  $\text{int}(-1.7) == -1$ .

### 2.9.3. Библиотека math

Для проведения вычислений с действительными числами Python содержит много дополнительных функций, собранных в библиотеку (модуль), которая называется *math*.

Для использования этих функций в начале программы необходимо подключить математическую библиотеку, что делается командой *import math*. Вызов и работа модуля *math* показана на рис. 2.15.

В качестве примера на рис. 2.15 представлено вычисление синуса значения  $x$ . Данная функция принимает на вход радианы, которые преобразованы функцией *radians()* из модуля *math*.

Использование слова «импорт» перед модулем говорит интерпретатору импортировать все функции и объекты из модуля. Иногда возникают случаи, когда не требуется импорт всего содержимого, возможно, есть функции в импортируемом модуле с одинаковыми именами, что вызовет ошибку. Существует возможность импорта отдельных функций и объектов из модуля. Пример импорта отдельных модулей представлен на рис. 2.16.

При таком импорте не нужно прописывать *math* перед каждой функцией, что позволяет быстрее писать код, но требует понима-

The screenshot shows the Visual Studio Code interface with a Python file named 'lesson\_one.py' open. The code contains the following:

```
1 import math
2
3 x = 90
4 y = math.sin(math.radians(x))
5
6 print(y)
7
```

Below the code editor, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab shows the command line output:

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_o
ne.py
1.0
ros-industrial@ros-training:~$
```

The status bar at the bottom indicates Python 3.8.5 64-bit, 0 errors, and 0 warnings.

Рис. 2.15. Использование модуля *math*

```

lesson_one.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_one.py ...
home > ros-industrial > new_project > python_lesson > lesson_one.py > ...
1   from math import sin, radians
2
3   x = 90
4   y = sin(radians(x))
5
6   print(y)
7

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_o
ne.py
1.0
ros-industrial@ros-training:~$ 

Python 3.8.5 64-bit ① 0 △ 0
Ln 5, Col 1 Spaces: 4 UTF-8 LF Python ⚙ ②

```

Рис. 2.16. Импорт отдельных функций модуля *math*

Таблица 2.2

**Часто используемые функции из модуля *math***

Функция	Описание
<i>round(x)</i>	Округляет число до ближайшего целого. Если дробная часть числа равна 0.5, то число округляется до ближайшего четного числа
<i>round(x, n)</i>	Округляет число <i>x</i> до <i>n</i> знаков после точки. Это стандартная функция, для ее использования не нужно подключать модуль <i>math</i>
<i>floor(x)</i>	Округляет число вниз («пол»), при этом <i>floor(1.5) = 1</i> , <i>floor(-1.5) = -2</i>
<i>ceil(x)</i>	Округляет число вверх («потолок»), при этом <i>ceil(1.5) = 2</i> , <i>ceil(-1.5) = -1</i>
<i>sqrt(x)</i>	Квадратный корень. Использование: <i>sqrt(x)</i>
<i>log(x)</i>	Натуральный логарифм. При вызове в виде <i>log(x, b)</i> возвращает логарифм по основанию <i>b</i>
<i>e</i>	Основание натуральных логарифмов <i>e = 2.71828</i>
<i>sin(x)</i>	Синус угла, задаваемого в радианах
<i>cos(x)</i>	Косинус угла, задаваемого в радианах
<i>tan(x)</i>	Тангенс угла, задаваемого в радианах
<i>asin(x)</i>	Арксинус, возвращает значение в радианах

Функция	Описание
$\text{acos}(x)$	Арккосинус, возвращает значение в радианах
$\text{atan}(x)$	Арктангенс, возвращает значение в радианах
$\text{atan2}(y, x)$	Полярный угол (в радианах) точки с координатами $(x, y)$
$\text{degrees}(x)$	Преобразует угол, заданный в радианах, в градусы
$\text{radians}(x)$	Преобразует угол, заданный в градусах, в радианы
$\pi$	Константа $\pi = 3.1415$

ния, какие именно функции нужны при решении текущей задачи. Полная документация на модуль *math* доступна по ссылке: <https://docs.python.org/3/library/math.html>.

В табл. 2.2. представлены часто используемые функции из модуля *math*.

Интересно, что некоторые функции являются частью стандартных функций языка *Python*, например функция *abs()*, которая находит абсолютное значение числа.

## 2.10. Условия и логические операторы

### 2.10.1. Условия

Существуют случаи, когда программа требует разветви, есть два и более путей по которым пойдет работа нашей программы. В этом случае необходимо использовать оператор условия. С технической точки зрения программа получает возможность ветвления. В зависимости от выбранной ветки программа движется по отдельной ветви для получения уникального и необходимого результата.

Процесс написания условий аналогичен переводу на язык программирования. Условие проще всего строить в виде предложения, начинать со слова «если», продолжать словом «то» и завершать словом «иначе», «то».

В *Python* данные слова заменяются условными операторами:

- 1) *if* – если;
- 2) *else* – иначе.

В качестве слова «то» используется символ «`:`» – разделитель, который начинает тело условия.

The screenshot shows a Visual Studio Code interface. On the left is a sidebar with icons for file operations like Open, Save, Find, and Run. The main area has tabs for 'Get Started' and 'lesson\_one.py'. Below the tabs is a code editor with the following Python code:

```
1 n = int(input())
2
3 if n % 2 == 0:
4     print('even')
5 else:
6     print('odd')
7
```

Below the code editor are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active, showing the command `/bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_one.py` and its output:

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_one.py
10
even
ros-industrial@ros-training:~$
```

At the bottom of the interface, there's a status bar with 'Python 3.8.5 64-bit', 'Ln 5, Col 6', 'Spaces: 4', 'UTF-8', 'LF', 'Python', and other icons.

Рис. 2.17. Проверка условия четности в виде программного кода

Например, рассмотрим возможность проверки условия четности числа.

*Четным является число, остаток от деления на два которого равен 0 и нечетным во всех остальных случаях.*

Таблица 2.3

#### Операторы условия

<code>==</code>	Проверка на равенство
<code>!=</code>	Проверка на неравенство
<code>&gt;</code>	Больше
<code>&gt;=</code>	Больше или равно
<code>&lt;</code>	Меньше
<code>&lt;=</code>	Меньше или равно
<code>is</code>	Проверка на идентичность
<code>is not</code>	Проверка на неидентичность
<code>in</code>	Проверка вхождения объекта
<code>not in</code>	Проверка, что объект не входит
<code>not</code>	Логическое НЕ (отрицание)
<code>and</code>	Логическое И
<code>or</code>	Логическое ИЛИ

Переформатируем наше условие.

*Если остаток от деления числа на два равно нулю, то число четное, иначе – число нечетное.*

Представим это выражение в виде программного кода, изображенного на рис. 2.17, в окне среды разработки *VSCode*.

Следует обратить внимание, что мы используем при сравнении два символа `==`, потому что одиночный символ является оператором присваивания. Данный оператор является логическим. Оператор условия работает во всех случаях, когда его выражение является не нулем, ложью или пустым значением. Об этом следует помнить при конструировании условий.

### 2.10.2. Логические операторы

Пример двойного символа `<==>` показывает наличие специальных операторов условия, которые позволяют проверять ту или иную конструкцию проектируемого условия. Они позволяют комбинировать и объединять несколько условий в одно. В табл. 2.3 представлены основные операторы условий языка *Python*.

Представленные в табл. 2.3 операторы выстроены по приоритету их работы. Это значит, что оператор *or* имеет самый низкий приоритет и будет выполняться последним. С помощью логических операторов происходит сравнение двух объектов слева и справа от оператора.

Логическое И является бинарным оператором (то есть оператором с двумя operandами: левым и правым) и имеет вид *and*. Оператор *and* возвращает *True* тогда и только тогда, когда оба его операнда имеют значение *True*.

Логическое ИЛИ является бинарным оператором и возвращает *True* тогда, когда хотя бы один operand равен *True*. Оператор «логическое ИЛИ» имеет вид *or*.

Логическое НЕ (отрицание) является унарным оператором (то есть оператором с одним operandом) и имеет вид *not*, за которым следует единственный operand.

Логическое НЕ возвращает *True*, если operand равен *False*, и наоборот.

## 2.11. Коллекции Python

В языке программирования *Python* реализовано несколько типов коллекций. Коллекции – это наборы данных или значений одного или различных типов, которые позволяют обращаться к ним.

В данном пособии будут рассмотрены несколько основных видов коллекций: строки, списки и словари.

### 2.11.1. Строки

Строка состоит из последовательности символов. Любой другой объект в *Python* можно перевести к строке, которая ему соответствует. Для этого нужно вызывать функцию *str()*, передав ей в качестве аргумента объект, переводимый в строку.

*Каждая строка, с точки зрения Python, – это объект класса str. Чтобы получить на базе одного объекта другой объект другого класса, как-то ему соответствующий, можно использовать функцию приведения. Имя этой функции совпадает с именем класса, к которому приводится объект. (Для знатоков: эта функция – это конструктор объектов данного класса.) Пример: int – класс для целых чисел. Перевод строки в число осуществляется функцией int().*

Из примера на рис. 2.18 видно, что строку можно получить как значение функции *input()*, а также присвоить значение строки в явном виде. Также есть возможность создать пустую строку с помощью функции *str()*, присвоив ее значение переменной.

В примере используется функция *len()*, которая находит длину объекта. Данная функция применима для большинства коллекций языка программирования *Python*.

```
lesson_two.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_two.py ...
home > ros-industrial > new_project > python_lesson > lesson_two.py > ...
1 data_one = input()
2 data_two = ' World!'
3 data_one += data_two
4 print(data_one)
5 print(len(data_one))
6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_t
wo.py
Hello
Hello World!
12
ros-industrial@ros-training:~$
```

Python 3.8.5 64-bit ① 0 △ 0 Ln 1, Col 14 Spaces: 4 UTF-8 LF Python ⚙ ②

Рис. 2.18. Взаимодействие со строками

### 2.11.2. Срезы

Срез (*slice*) – извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности. Срезы чем-то напоминают интервалы отрезков, которые используются в математике.

Срезы позволяют выделять не только часть последовательности, но и одиночные символы. Операция среза выполняется через квадратные скобки.

Рассмотрим пример строки, содержащей слово *Hello*, которое помещено в переменную *s*.

Индексация в *Python* начинается с 0. Соответственно, каждый следующий индекс больше на единицу. Последним индексом является значение *len( )-1*. В табл. 2.4 демонстрируется принцип индексации в языке *Python*.

Для получения отдельного символа из заданной строки используется срез вида *s[0] == 'H'*, *s[4] == o*. Согласно табл. 2.4, значение последнего индекса соответствует четырем. Возможно получить данное значение и с помощью более сложной конструкции *s[len(s)-1]*. Данный пример демонстрирует гибкость при написании кода, а также факт наличия нескольких подходов и возможностей для решения поставленной задачи.

В *Python* реализована возможность брать значение, используя отрицательную индексацию. Пример индексации представлен в табл. 2.5.

В табл. 2.5 показано, что нумерация элементов при срезе начинается с -1, что является отличной возможностью получить последнее значение, даже если не известна длина строки – *s[-1] == o*.

Таблица 2.4

#### Строка и индексы

Слово str()	<i>H</i>	<i>e</i>	<i>l</i>	<i>l</i>	<i>o</i>
Индекс	0	1	2	3	4

Таблица 2.5

#### Строка и отрицательные индексы

Слово str()	<i>H</i>	<i>e</i>	<i>l</i>	<i>l</i>	<i>o</i>
Индекс	-5	-4	-3	-2	-1

Важно помнить, что индексы ограничены и не могут быть больше в положительную сторону до `len(s)-1` и отрицательную сторону до `-len(s)`. Если попробовать вызвать индекс, значение которого превышает данные лимиты, то возникнет ошибка `IndexError: string index out of range`.

Возможно получить срез не только одним символом, но и задав ограничения для левого и правого края – `s[1:4]`. Данная конструкция вернет строку вида `'ell'`. Срез реализуется через оператора «двоеточие», который помещается между двумя значениями среза. При реализации среза важно помнить, что значение индекса среза слева является началом отсчета, а значение индекса справа не входит в срез. При использовании такой формы среза ошибка `IndexError` никогда не возникает.

Например, срез `s[1:5]` вернет строку `'ello'`, таким же будет результат, если сделать второй индекс очень большим, например, `s[1:100]` (если в строке не более 100 символов). Это дает возможность не указывать второй аргумент среза, чтобы дойти до конца строки, и позволяет использовать конструкцию вида `s[2:] == 'lo'`.

Последним важным аргументом среза является шаг, который можно использовать для перебора не каждого символа, а, например, через один символ: `s[0::2]`. По умолчанию в срезе используется шаг 1, каждый символ, но дополнение третьим аргументом позволяет выводить только четные значения символов `s[0::2] == 'el'`.

Каждое применение среза не изменяет строку, так как строка – это неизменяемый тип данных. Нельзя записать `s[0] = 'h'`, это вызовет ошибку. Поэтому изменить исходную строку возможно только в том случае, если значение присваивается новой строке и дополняется символом, который нужно заменить.

### 2.11.3. Списки

Списки являются очень гибкой структурой данных и широко используются в программах. Основные свойства списка в сравнении с другими коллекциями языка `Python`:

- 1) список хранит несколько элементов под одним именем (как и множество);
- 2) элементы списка могут повторяться (в отличие от множества);
- 3) элементы списка упорядочены и проиндексированы, доступна операция среза (как в строке);
- 4) элементы списка можно изменять (в отличие от строки);
- 5) элементами списка могут быть значения любого типа: целые и действительные числа, строки и даже другие списки.

The screenshot shows a Visual Studio Code interface. The top bar has the title "lesson\_two.py - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help. The left sidebar has icons for Get Started, lesson\_two.py, and other project files. The main editor window contains the following Python code:

```
new_l = [1, 2.2, 3e3, '4', '55']
old_l = list()
print(new_l)
print(old_l)
```

The terminal tab at the bottom shows the output of running the script:

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_two.py
[1, 2.2, 3000.0, '4', '55']
[]
```

The status bar at the bottom indicates Python 3.8.5 64-bit, Ln 5, Col 1, Spaces: 4, UTF-8, LF, Python.

Рис. 2.19. Создание и вывод значения списков

Список представляет собой последовательность элементов. Список можно задать перечислением элементов списка в квадратных скобках, а также используя функцию *list()*. Использование списков представлено на рис. 2.19.

В примере на рис. 2.19 показано создание двух списков. Как и длину строк, длину списка можно найти, используя функцию *len()*, а также использовать срезы для извлечения отдельных значений.

Изменение отдельного элемента списка ведет к изменению исходного списка. Как и строки, списки поддерживают отрицательную индексацию.

#### 2.11.4. Цикл *while*

Цикл *while* («пока») позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл *while* используется, когда невозможно определить точное значение количества проходов исполнения цикла. На рис. 2.20 показано использование оператора цикла, а также вывод данных.

При выполнении цикла *while* сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла *while*. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие

```
lesson_two.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_two.py ...
home > ros-industrial > new_project > python_lesson > lesson_two.py > ...
1 n = int(input())
2 i = 0
3 while i < n:
4     print(i, end=' ')
5     i += 1
6 print()
7
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~/bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_tw
wo.py
5
0 1 2 3 4
ros-industrial@ros-training:~$
```

Python 3.8.5 64-bit ⚡ 0 △ 0 Ln 6, Col 8 Spaces: 4 UTF-8 LF Python ⚡

Рис. 2.20. Использование цикла *while*

станет ложно, работа цикла завершается и управление передается следующей инструкции после цикла.

В примере на рис. 2.20 видно, что переменная *i* внутри цикла изменяется от 1 до *n*. Такая переменная, значение которой меняется с каждым новым проходом цикла, называется *счетчиком*. Заметим, что после выполнения этого фрагмента значение переменной *i* будет равно 10, поскольку именно при *i == 10* условие *i < n* впервые перестанет выполняться.

#### 2.11.5. Цикл *for*

Цикл *for*, также называемый *циклом с параметром*, в языке *Python* обладает большим набором возможностей. В цикле *for* ука-

```
lesson_two.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_two.py ...
home > ros-industrial > new_project > python_lesson > lesson_two.py > ...
1 n = int(input())
2 for i in range(n):
3     print(i, end=' ')
4 print()
5
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
/bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_tw
wo.py
5
0 1 2 3 4
ros-industrial@ros-training:~$
```

Python 3.8.10 64-bit ⚡ 0 △ 0 Ln 4, Col 8 Spaces: 4 UTF-8 LF Python ⚡

Рис. 2.21. Пример использования цикла *for*

зывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном. Данное множество является итерируемым объектом, по которому выполняется последовательный проход на каждом шаге цикла. Пример использования цикла *for* представлен на рис. 2.21.

Данный пример представляет собой программу, выполняющую тот же функционал, что и в цикле *while*. Следует заметить, что при таком подходе код программы немного сократился в количестве строк, но это не главное положительное свойство использования данного примера. Данный пример показывает, что переменная для итерации задается при описании цикла, а также создается отдельная последовательность значений с помощью конструкции *range*.

## 2.12. Функции языка Python

### 2.12.1. Функции

Функции – это такие изолированные участки кода, которые выполняются только тогда, когда это необходимо. Для того чтобы указать интерпретатору *Python* на те участки кода, которые надо выполнять обособленно, нужно определить их как функции. У функции есть имя (обычно), перечень аргументов, которые она принимает (иногда), и значение, которое она возвращает (не часто).

В пособии уже рассматривались несколько примеров функций, встроенных или вызываемых из модуля *math*. Они обладают свойствами принимать аргументы. Например, *int()*, *len()* возвращают результат работы. Функция *print()* принимает аргументы, но ничего не возвращает.

При написании своей функции следует пользоваться синтаксисом, при котором значение функции начинается с оператора *def*. Пример написания пользовательской функции представлен на рис. 2.22.

На рис. 2.22 показана функция *function\_one*, которая принимает на вход аргумент со значением *value*, которое выводится функцией *print()*. Представленная функция не возвращает значения.

Вызов функции выполняется с помощью имени функции. Обязательно необходимо сначала объявлять функции, а потом вызывать их, чтобы не возникало ошибок выполнения кода.

```
lesson_three.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_three.py
home > ros-industrial > new_project > python_lesson > lesson_three.py > ...
1 def function_one(value):
2     print(value)
3
4 function_one(10)
5

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_t
hree.py
10
ros-industrial@ros-training:~$ 

Python 3.8.10 64-bit ⚡ 0 ⚡ 0
Ln 5, Col 1 Spaces: 4 UTF-8 LF Python ⚡ ⚡
```

Рис. 2.22. Пример объявления и вызова пользовательской функции

```
lesson_three.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_three.py
home > ros-industrial > new_project > python_lesson > lesson_three.py > function_one
1 def function_one(value):
2     return value * 2
3
4 a = function_one(10)
5 print(a)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new project/python_lesson/lesson_t
hree.py
20
ros-industrial@ros-training:~$ 

Python 3.8.10 64-bit ⚡ 0 ⚡ 0
Ln 2, Col 21 Spaces: 4 UTF-8 LF Python ⚡ ⚡
```

Рис. 2.23. Пример программы, которая возвращает измененное значение аргумента

Если необходимо возвращать значение функции, следует использовать конструкцию *return* – возврат. Пример такой программы представлен на рис. 2.23.

Пользовательская функция возвращает *None*, если в ней нет возвращаемого значения.

### 2.12.2. Стандартные функции в языке Python

Стандартная библиотека *Python* богата различными функциями для работы с данными, сетевыми протоколами, ОС, файлами, *XML*, регулярными выражениями и прочим.

Зачастую нет необходимости писать какой-то алгоритм с нуля. Существует большой набор готовых функций из стандартной библиотеки или из большого количества сторонних модулей.

Полный список стандартных функций *Python* доступен по ссылке: <https://docs.python.org/3/py-modindex.html>.

## 2.13. Основы объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) – это парадигма программирования, где различные компоненты компьютерной программы моделируются на основе реальных объектов. Объект – это сущность, которая обладает характеристиками, а также выполняет какой-либо функционал. Программа в *Python* создается как совокупность объектов, которые взаимодействуют друг с другом и внешним миром. Каждый объект является экземпляром класса.

### 2.13.1. Создание классов

Для создания класса необходимо прописать ключевое слово *class* и далее название для класса. Стиль языка программирования *Python* регламентирует названия классов с буквы в верхнем регистре. Класс содержит поля (переменные), методы (функции), а также конструкторы. Создание класса и наполнение его информацией позволяет описывать объекты. Объекты будут иметь доступ к характеристикам класса.

Класс состоит из объявления (инструкция *class*), имени класса и тела класса, которое содержит атрибуты и методы. На основе одного класса возможно создание множества объектов с одинаковыми свойствами. На рис. 2.24 объявлен класс «Автомобиль».

Представленный на рис. 2.24 пример демонстрирует объявление класса *Vehicle*.

*my\_vehicle* является экземпляром класса, который описывает некий автомобиль черного цвета с четырьмя колесами. Вызов атрибутов и методов класса выполняется через символ «.».

В данном классе объявлены три метода: *\_\_init\_\_*, *brake*, *drive*. Каждый метод выполняет ту или иную роль.

Метод *\_\_init\_\_* позволяет настроить поля цвета автомобиля и количества колес автомобиля. Метод *brake* сообщает об остановке. Метод *drive* возвращает сообщение о движении. Объявление и использование методов аналогично использованию функций, их

The screenshot shows a Visual Studio Code interface with the following details:

- Title Bar:** lesson\_three.py - Visual Studio Code
- File Explorer:** Shows a file tree with lesson\_three.py selected.
- Code Editor:** Displays the following Python code:

```
1  class Vehicle:
2      def __init__(self, color, tires):
3          self.color = color
4          self.tires = tires
5
6      def brake(self):
7          return 'Braking'
8
9      def drive(self):
10         return "I'm driving"
11
12 my_vehicle = Vehicle('black', 4)
13 print(my_vehicle.color)
14 print(my_vehicle.drive())
15 
```
- Terminal:** Shows the output of running the script:

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_three.py
black
I'm driving
ros-industrial@ros-training:~$ 
```
- Status Bar:** Python 3.8.10 64-bit, Ln 15, Col 1, Spaces: 4, UTF-8, LF, Python

Рис. 2.24. Объявление класса «Автомобиль»

синтаксис почти идентичен, кроме одного элемента – аргумента *self*.

### 2.13.2. Использование *self* в классах

*Self* – специальный аргумент методов, ссылка на текущий экземпляр класса. Данное слово позволяет организовать общее пространство имен внутри класса и разделять ресурсы внутри памяти программы.

#### 2.13.3. Метод *\_\_init\_\_*

Данный метод является специальным для класса. В языке *Python* существует несколько специальных методов, подробнее с ними можно ознакомиться по ссылке: <https://habr.com/ru/post/186608>.

В примере используется метод *\_\_init\_\_*, который является конструктором класса. Его задача – на стадии создания экземпляра класса выполнить первоначальное конструирование класса, указав, какие атрибуты задать вначале для данного экземпляра.

## 2.14. Концепции ООП

Концепции ООП включают в себя абстракцию, наследование, инкапсуляцию и полиморфизм. В данном пособии будут рассмо-

трены три главных подхода. Абстракция не затрагивается, так как на стадии проектирования класса сразу дается понимание, какие поля важны, и отбрасываются лишние атрибуты и методы.

### **2.14.1. Наследование**

Наследование является одним из ключевых понятий ООП. За счет наследования можно создать один общий класс (класс-родитель) и создать множество других классов (классы-наследники), которые будут наследовать все поля, методы и конструкторы из главного класса. Зачем использовать наследование? Предположим, что у нас есть один исходный класс «Транспортное средство». В классе описываются базовые характеристики для всех транспортных средств: поля – скорость, цвет, запас хода и др.; методы – получение информации из полей, установка новых значений; конструктор – пустой и по установке всех полей.

Данный класс является основой для объявления объекта «Автомобиль», объекта «Грузовик», объекта «Мотоцикл». У всех объектов будут одинаковые характеристики и методы. Из этого примера следует, что транспортные средства не равны по своим особенностям, а иногда могут сильно отличаться друг от друга, например, если произойдет объявление объекта «Самолет» или «Поезд». Они также сохраняют свойства и методы класса «Транспортное средство», но у них есть специфические поля, которые уникальны.

Есть два подхода к решению данной проблемы:

1. Создать три отдельных класса: «Автомобиль», «Самолет», «Поезд». В каждом классе будут все методы, поля и конструкторы, повторно переписанные из класса «Транспортное средство», а также будут новые методы, которые важны только для конкретного класса.

2. Создать три класса-наследника: «Автомобиль», «Самолет», «Поезд». Эти три класса будут наследовать все свойства класса «Транспортное средство» и при этом содержать свои дополнительные функции.

При втором подходе нет необходимости повторения кода, кроме того, код станет меньше и чище.

### **2.14.2. Создание классов-наследников**

Для создания класса-наследника требуется создать класс и указать наследование от главного класса. Указание класса-родителя происходит с помощью круглых скобок при объявлении класса. Пример класса-наследника представлен на рис. 2.25.

```
lesson_three.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_three.py ...
home > ros-industrial > new_project > python_lesson > lesson_three.py > ...
1 class Vehicle:
2     def __init__(self, color, tires):
3         self.color = color
4         self.tires = tires
5
6 class Autonomousvehicle(Vehicle):
7     def on_autopilot(self):
8         return 'On'
9
10 my_vehicle = Autonomousvehicle('white', 4)
11 print(my_vehicle.color)
12 print(my_vehicle.on_autopilot())
13

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~/bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_three.py
white
On
ros-industrial@ros-training:~$ 

Python 3.8.10 64-bit ① 0 △ 0
Ln 12, Col 32 Spaces: 4 UTF-8 LF Python ⚙
```

Рис. 2.25. Создание класса-наследника

В данном примере родительский класс *Vehicle* («Транспортное средство») имеет конструктор, который задает цвет и количество колес.

Второй класс не имеет конструктора класса и отвечает за создание автономного транспортного средства. Так как класс-потомок получил все методы и атрибуты родительского класса, то интерпретатор не выводит ошибку при объявлении объекта класса *my\_vehicle*.

Следует помнить, что методы класса-родителя доступны всем классам-потомкам. Если есть наследование в два уровня, то самый маленький потомок ищет методы и аргументы на втором уровне класса-потомка. Если потомок третьего уровня не находит нужные методы на втором уровне, то он обращается к первому родительскому классу.

### 2.14.3. Инкапсуляция

Позволяет ограничить доступ к какому-либо методу в классе. Благодаря такому подходу существует возможность создать защищенные поля, из которых невозможно получить данные, огравив их от злоумышленников, или оградить себя от изменения данных, или намеренно вызвать или изменить метод. На рис. 2.26

```

class Personal_data:
    def __password(self):
        print('qwerty123456')

my_data = Personal_data()
my_data.__password()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

our.py  
Traceback (most recent call last):  
 File "/home/ros-industrial/new\_project/python\_lesson/lesson\_four.py", line 6, in <module>  
 my\_data.\_\_password()  
AttributeError: 'Personal\_data' object has no attribute '\_\_password'  
ros-industrial@ros-training:~\$

Ln 6, Col 21 Spaces: 4 UTF-8 LF Python ⌂ ⌂

*Рис. 2.26. Вызов инкапсулированного метода*

представлен пример программы с инкапсулированным методом *\_password*.

В примере на рис. 2.26 заметно, что запрос данных вызвал ошибку, так как у пользователя нет доступа к данному методу. Инкапсулированные методы начинаются с двойного символа нижнего подчеркивания. Аналогичным образом возможно инкапсулировать атрибуты класса.

#### **2.14.4. Полиморфизм**

Полиморфизм позволяет одинаково взаимодействовать с объектами, имеющими однотипный интерфейс, независимо от внутренней реализации объекта. Например, с объектом класса «Грузовой автомобиль» можно производить те же операции, что и с объектом класса «Автомобиль», так как первый является наследником второго, при этом обратное утверждение неверно.

Другими словами, полиморфизм предполагает разную реализацию методов с одинаковыми именами. Это очень полезно при наследовании, когда в классе-наследнике можно переопределить методы класса-родителя. Пример программы, в которой реализован полиморфный класс, представлен на рис. 2.27.

В примере на рис. 2.27 показан вариант, при котором существуют два метода с названиями *drive*, отвечающие за возможность управления транспортным средством. Первый класс использует данный метод просто для возврата слова *drive* – возможность управлять. Второй класс имеет более интересную структуру. В нем также есть метод *drive*, который зависит от объекта *self.on*. Вызов метода

```
lesson_three.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started lesson_three.py
home > ros-industrial > new_project > python_lesson > lesson_three.py > ...
1  class Vehicle:
2      def __init__(self, color, tires):
3          self.on = False
4          self.color = color
5          self.tires = tires
6      def drive(self):
7          return 'Drive'
8
9
10 class Autonomousvehicle(Vehicle):
11     def on_autopilot(self):
12         self.on = True
13     def drive(self):
14         if self.on:
15             return "You can't manual drive"
16         return 'Drive'
17
18
19 my_vehicle = Autonomousvehicle('white', 4)
20 print(my_vehicle.color)
21 my_vehicle.on_autopilot()
22 print(my_vehicle.drive())
23
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ros-industrial@ros-training:~$ /bin/python3 /home/ros-industrial/new_project/python_lesson/lesson_t
Vehicle.py
white
You can't manual drive
ros-industrial@ros-training:~$
```

Ln 21, Col 26 Spaces: 4 UTF-8 LF Python ⚙

Рис. 2.27. Реализация полиморфного класса

*on\_autopilot()* изменяет значение данного объекта, что ведет к изменению метода *drive*. По итогу метод реализует условие, которого не было в родительском классе, запрещающее вести транспортное средство в ручном режиме, если значение объекта *self.on == True*.

## 2.15. Практические задания

**Задание 1.** Используя менеджер пакетов *pip*, выполните установку модуля *pyserial* в язык *Python*.

**Задание 2.** Создайте программу на языке *Python*, которая считывает строку с клавиатуры и выводит символы в обратном порядке.

**Задание 3.** Добавьте последовательность шебанг в начало скрипта.

**Задание 4.** Настройте права для файла так, чтобы пользователь мог запускать скрипт программы без интерпретатора *Python*.

**Задание 5.** Напишите программу, которая проверяет, что пользователь при вводе почтового ящика ничего не перепутал и ввел корректный логин (не содержащий символ «@») и корректный ре-

зарвный адрес (содержащий символ «@»). Иных проверок, кроме указанных здесь, выполнять не надо.

#### Ввод

Вводятся две строки: предлагаемые пользователем логин и резервный адрес.

#### Вывод

Выводится одна строка: если все условия выполнены, то выводится «OK» (латиницей); если в логине присутствует «@», то выводится «Некорректный логин»; если логин корректный, но в адресе отсутствует «@», то выводится «Некорректный адрес».

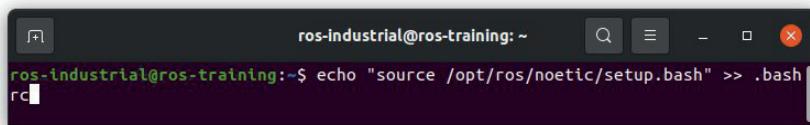
Задание 6. Создайте класс с именем *student*, содержащий поля: фамилия и инициалы, номер группы, успеваемость (массив из пяти элементов). Создать массив из десяти элементов такого типа, упорядочить записи по возрастанию среднего балла. Добавить возможность вывода фамилий и номеров групп студентов, имеющих оценки только 4 или 5.

### 3. ROS – ROBOT OPERATING SYSTEM

*Robot Operating System (ROS)* – это гибкий фреймворк для написания программного обеспечения для роботов. Это набор инструментов, библиотек и рекомендаций, которые направлены на упрощение задачи создания сложных, но надежных роботов на самых разных роботизированных платформах.

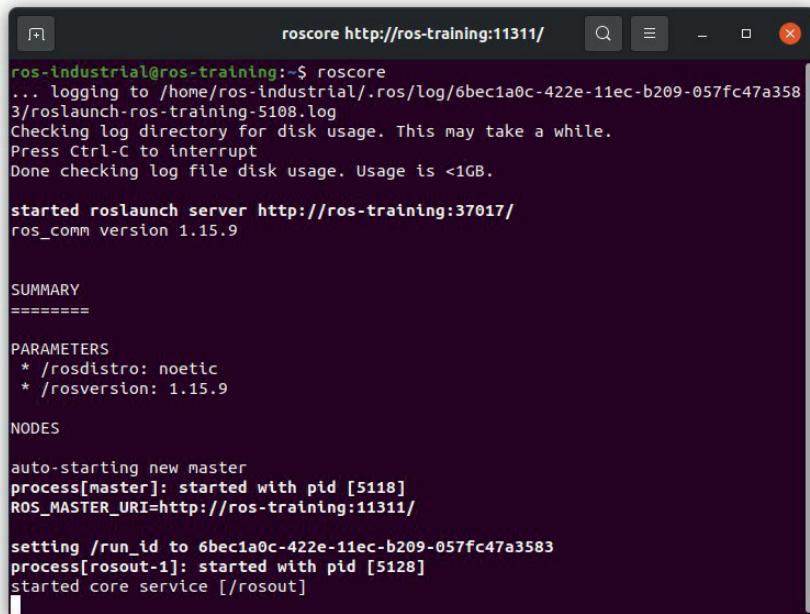
Предоставленный образ ОС, развернутый в виртуальной машине, уже имеет предустановленную версию *ROS*, а также набор утилит и библиотек.

Для начала работы с *ROS* необходимо выполнить ряд операций: прописать в терминале окружение *ROS*, чтобы возможно было вы-



```
ros-industrial@ros-training:~$ echo "source /opt/ros/noetic/setup.bash" >> .bashrc
```

Рис. 3.1. Добавление окружения *ROS* в терминальную программу



```
roscore http://ros-training:11311/
... logging to /home/ros-industrial/.ros/log/6bec1a0c-422e-11ec-b209-057fc47a358
3/roslaunch-ros-training-5108.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ros-training:37017/
ros_comm version 1.15.9

SUMMARY
=====

PARAMETERS
  * /rosdistro: noetic
  * /rosversion: 1.15.9

NODES

auto-starting new master
process[master]: started with pid [5118]
ROS_MASTER_URI=http://ros-training:11311/

setting /run_id to 6bec1a0c-422e-11ec-b209-057fc47a3583
process[rosout-1]: started with pid [5128]
started core service [/rosout]
```

Рис. 3.2. Запуск ядра *ROS*

зывать команды, утилиты, предустановленные и связанные с мета-операционной системой. Для выполнения данной задачи необходимо выполнить команду в терминале, представленную на рис. 3.1.

Данная команда прописывает информацию об окружении *ROS* в файл *bashrc*, из которого терминальная программа берет параметры настройки запуска.

Для проверки работоспособности системы и возможности взаимодействия с *ROS* рекомендуется запустить ее ядро. Запуск ядра (мастера) *ROS* показан на рис. 3.2.

В случае успешного запуска ядра в окне терминала будет показана информация об успешной работе. Присвоено название мастеру *ROS* с портом 11311 по умолчанию. Запускать *roscore* следует только один раз и требуется постоянно держать данную утилиту в запущенном режиме. Для того чтобы случайно не закрыть окно с ядром, возможно запустить данный процесс в скрытом режиме, используя символ &.

### 3.1. Причины использования ROS

В данный момент для целей изучения и погружения в робототехнику можно однозначно рекомендовать *ROS*. Важными критериями на этапе погружения в робототехнику являются активность сообщества, наличие различных библиотек, расширяемость и простота использования.

Следует отметить, что сообщество *ROS* чрезвычайно активно. При возникновении проблем найти решение и получить помочь становится проще не только от разработчика *ROS* – компании *Open Robotics*, но и от других энтузиастов и профессионалов.

### 3.2. Положительные качества ROS

*ROS* обладает рядом положительных характеристик, которые позволяют эффективно вести разработку, администрирование и масштабирование программного обеспечения, разработанного для роботов. Кроме того, предполагает:

1. Повторное использование программных модулей.

Разработанный программный модуль легко запускается и используется в любом другом приложении. Вопросы установки зависимостей и других библиотек хорошо проработаны и автоматизированы.

2. Готовый протокол коммуникации.

Основная проблема комплексных робототехнических систем – это решение задач коммуникации в рамках одного приложения. Для

решения этих задач ROS содержит все необходимые утилиты. Любой программный модуль может быть представлен как отдельный процесс, взаимодействующий с другими процессами по сетевому протоколу. Такой подход позволяет создавать независимые и простые в повторном использовании программные модули, которые возможно запустить / остановить / модифицировать на любом устройстве.

### 3. Широкий набор средств разработки и отладки.

ROS предоставляет готовые инструменты для отладки, инструмент 2D-визуализации (*rqt*), инструмент 3D-визуализации (*Rviz*), инструмент 3D-симуляции (*Gazebo*).

### 4. Активное и открытое сообщество.

Сообщества разработчиков робототехники из академического мира и промышленности были относительно закрытыми до последнего времени. Но сейчас существует активное и, главное, открытое сотрудничество всех участников. В центре этого изменения – программная платформа с открытым исходным кодом. В случае ROS существует более 5000 пакетов, которые были разработаны и выложены в общий доступ. Описание этих пакетов, инструкций и другой полезной информации превышает 18 000 *wiki*-страниц.

### 5. Собственная экосистема.

Вокруг ROS сформирована собственная экосистема (по аналогии с платформами *Android* и *Apple*). В ней существуют разработчики аппаратных платформ и программных модулей, энтузиасты и компании – производители промышленного оборудования, единое место распространения и хранения готовых модулей, документации. Все участники взаимодействуют и работают в рамках единой платформы.

## 3.3. Введение в ROS

Роботизированная система – это набор программ и подпрограмм, которые вместе путем постоянного взаимодействия обеспечивают работу и функционирование робота как единой системы.

Написание одной сложной большой программы, которая описывает работу всех систем, является непростой задачей, так как требует высокого уровня контроля и выявления возможных ошибок и неполадок. Отладка и поддержка такого программного обеспечения становится почти невозможной. При этом накладывается факт большого набора функций, которые должен выполнять робот.

Отличным вариантом является решение, при котором одновременно должно работать множество небольших программ. Каждая

такая программа выполняет одну или несколько простых функций. В формулировках *ROS* такая программа называется *нод (node)* – узел. Писать и поддерживать такие мини-программы значительно проще, чем одну большую.

При таком подходе возникает вопрос, каким образом выстроить межпрограммное взаимодействие и передачу данных от одной программы к другой. Иными словами, как ноды должны общаться?

В *ROS* реализован инструмент, который позволяет выполнять данные взаимодействие и передачу данных между отдельными узлами программ внутри его ядра. Данный инструмент – это подход, при котором программы имеют возможность публиковать данные и подписываться на данные. Подход называется *PubSub (publisher – издаватель, публикует данные, subscriber – подписчик, получает данные)*. Важно понимать, что данные сущности являются отдельными процессами, которые создаются внутри ОС, что позволяет получать к ним неограниченный доступ.

Издаватель (*publisher*) – это процесс, который имеет какие-либо данные и хочет сообщить о них всему роботу. Например, программа получает данные с датчика температуры, переводит их в понятные нам градусы Цельсия и отправляет эти данные в *ROS*.

Подписчик (*subscriber*) – это процесс, который хочет получить существующие в системе данные для дальнейшей их обработки. Например, это может быть часть программы (нода), которая получает данные о температуре. Если температура превышена, то включает вентилятор для охлаждения или выводит эти данные на экран.

Нод может быть одновременно и издавателем, и подписчиком, что позволяет организовать взаимодействие нескольких систем. Такой подход дает возможность строить очень сложные взаимодействия, используя очень простой подход обмена данными.

Обмен данными происходит через канал связи внутри ядра *ROS*. Данный канал связи для каждого нода имеет свое значение и называется *Topic* (тема), через которую идет связь нодов. У каждой темы есть свое название, которое выступает в качестве идентификатора и прописывается в коде программы каждого нода.

Обмен данными происходит с помощью сообщений (*message*). Важно понимать, что в рамках одного топика должны находиться сообщения только одного типа, например, температуры. Если нам необходимы данные о давлении, то необходимо работать с другим топиком, например *pressure*.

Через топики могут передаваться абсолютно разные данные, например, данные с лазерного дальномера или камеры.

### 3.4. Сообщения

Сообщение представляет собой структуру данных, которая используется при обмене информацией между нодами.

Топики (*topics*), службы (*services*) и действия (*actions*) используют сообщения для взаимодействия между собой. Сообщения могут включать в себя как базовые типы (целое число, число с плавающей точкой, логические и т. д.), так и массивы сообщений.

Помимо этого, сообщения могут инкапсулировать в себе другие существующие типы сообщений и специальные заголовки. Сообщения описываются в файлах *.msg* как пары значений: тип поля и имя поля. Например, *int32 x* и *int32 y* представляют собой сообщение, содержащее две переменные типа *int32* с именами *x* и *y*.

#### 3.4.1. Типы данных в ROS

Типы данных *ROS* не заимствованы напрямую ни из одного из языков. В момент «сборки» происходит преобразование типа *ROS* к типу используемого языка. В табл. 3.1 содержится описание основных типов *ROS* и их представление в языках *C++* и *Python* [1, с. 29].

Таблица 3.1

Соотношение типов данных *ROS*, *C++*, *Python*

<i>ROS</i>	<i>C++</i>	<i>Python</i>
<i>bool</i>	<i>uint8_t</i>	<i>bool</i>
<i>int8</i>	<i>int8_t</i>	<i>int</i>
<i>uint8</i>	<i>uint8_t</i>	<i>int</i>
<i>int16</i>	<i>int16_t</i>	<i>int</i>
<i>uint16</i>	<i>uint16_t</i>	<i>int</i>
<i>int32</i>	<i>int32_t</i>	<i>int</i>
<i>uint32</i>	<i>uint32_t</i>	<i>int</i>
<i>int64</i>	<i>int64_t</i>	<i>long</i>
<i>uint64</i>	<i>uint64_t</i>	<i>long</i>
<i>float32</i>	<i>float</i>	<i>float</i>
<i>float64</i>	<i>double</i>	<i>float</i>
<i>string</i>	<i>std::string</i>	<i>str</i>

<i>ROS</i>	<i>C++</i>	<i>Python</i>
<i>time</i>	<i>ros::Time</i>	<i>rospy.Time</i>
<i>duration</i>	<i>ros::Duration</i>	<i>rospy.Duration</i>
<i>fixed-length</i>	<i>boost::array, std::vector</i>	<i>tuple</i>
<i>variable-length</i>	<i>std::vector</i>	<i>tuple</i>
<i>uint8[]</i>	<i>std::vector</i>	<i>bytes</i>
<i>bool[]</i>	<i>std::vector</i>	<i>list of bool</i>

### 3.4.2. Получение информации о сообщениях

Для верной работы с сообщениями необходимо понимать их внутреннюю структуру. В *ROS* реализован инструмент, позволяющий выполнять анализ и вывод справочной информации о полях сообщений. Таким инструментом является команда *rosmsg*. Данная команда содержит несколько встроенных инструментов, позволяющих проводить анализ сообщений. Данные инструменты команды представлены в табл. 3.2.

Пример использования команды представлен на рис. 3.3.

Данный пример команды выводит структуру сообщения *Twist*, которое используется для хранения и передачи данных линейной и угловой скорости. Сообщение представляет собой два вектора-спи-

Таблица 3.2  
Инструменты команды *rosmsg*

Команда	Описание
<i>rosmg show</i>	Показать информацию о сообщении
<i>rosmg info</i>	Сокращение для команды <i>rosmg show</i>
<i>rosmg list</i>	Вывести все существующие типы сообщений
<i>rosmg md5</i>	Отобразить <i>md5sum</i> сообщения
<i>rosmg package</i>	Список всех сообщений в пакете
<i>rosmg packages</i>	Список пакетов, использующих сообщение

```

ros-industrial@ros-training:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z

ros-industrial@ros-training:~$ █

```

Рис. 3.3. Вывод информации о сообщении типа *Twist*

ска, в каждом по три значения. Данная информация пригодится при написании издателей и подписчиков на данные.

### 3.4.3. Topic

Модель работы в режиме *topic* подразумевает использование одного типа сообщения как для издателя (*publisher*), так и для подписчика (*subscriber*).

Модель *topic* является односторонней и подразумевает непрерывную отправку или получение сообщений. Такой способ коммуникации подходит для датчиков, которым требуется периодическая передача данных. Несколько подписчиков могут получать сообщения от одного издателя, и наоборот (возможна работа нескольких издателей).

На рис. 3.4 показана модель работы датчика температуры, когда его данные получают различные ноды [2, с. 75].

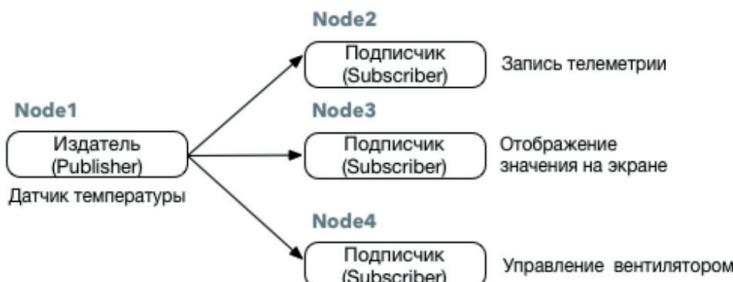


Рис. 3.4. Взаимодействие нод через топики

Датчик температуры является источником данных, который через именованный топик передает значения о температуре подписчикам. В качестве подписчиков выступают три узла системы, которые отвечают за отдельные блоки работы системы.

#### 3.4.4. Утилита *rostopic*

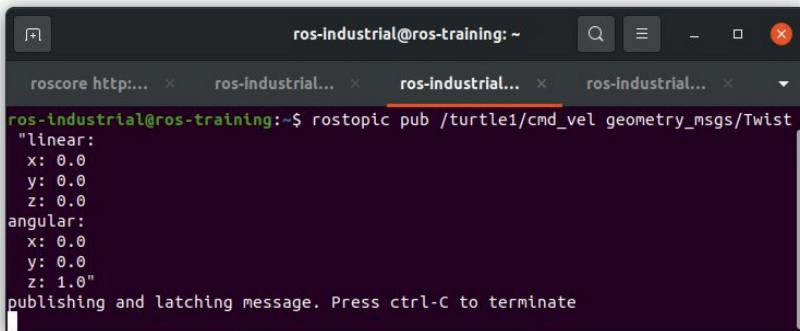
Большинство команд *ROS* представляют собой утилиты и построены по принципу добавления слова *ros* перед командами (*rosmsg*, *rostopic*, *roscore* и др.). Утилита *rostopic* предназначена для вывода информации о топиках в *ROS*. Каждый топик содержит метаинформацию о сообщении, о частоте, с которой идет публикация данных, а также о канале связи, который он занимает. Подробная информация об инструментах утилиты *rostopic* представлена в табл. 3.3.

В качестве примера рассмотрим публикацию сообщения в топик */turtlesim/cmd\_vel*, который отвечает за управление скоростью встроенного симулятора *ROS*. Подписка на топик показана на рис. 3.5, на котором используется строка кода, публикующая сообщения.

В предложенном примере на рис. 3.5 видно, что кроме указания сообщения необходимо было указать еще тип сообщения, а также значения для каждого элемента угловой и линейной скорости. При написании команды публикации следует активно использо-

Таблица 3.3  
Информация об инструментах утилиты *rostopic*

Команда	Описание
<i>rostopic bw</i>	Показать занимаемый сетевой канал
<i>rostopic echo</i>	Вывести сообщения на экран
<i>rostopic find</i>	Поиск топика по типу
<i>rostopic hz</i>	Показать частоту обновления топика
<i>rostopic info</i>	Показать информацию о топике
<i>rostopic list</i>	Показать список существующих топиков
<i>rostopic pub</i>	Опубликовать данные в топик
<i>rostopic type</i>	Показать тип сообщения для топика



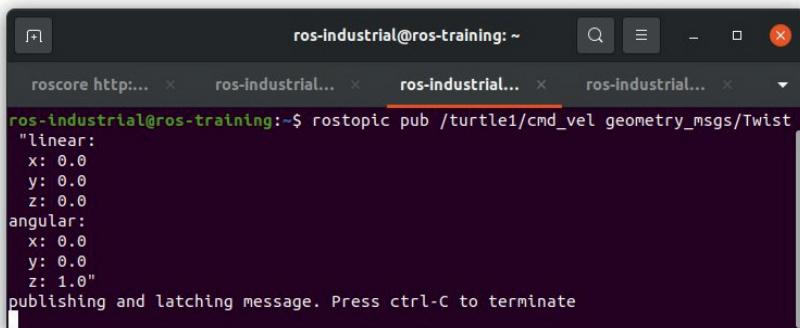
```
roscore http... ros-industrial... ros-industrial... ros-industrial...
ros-industrial@ros-training:~$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist
"linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0"
publishing and latching message. Press ctrl-C to terminate
```

Рис. 3.5. Публикация данных в топик */turtlesim/cmd\_vel*

вать клавишу *Tab*, которая позволяет автоматически дополнять текст утилиты *rostopic*, что позволяет не запоминать большие блоки информации о внутренней структуре сообщения, ускоряет написание, а также устраняет проблемы и ошибки при вводе текста команды. У утилиты *rostopic* есть дополнительные ключи для модификации ее работы, как у команд терминала *Linux*, например установка частоты публикации сообщений, а также количество публикаций.

На рис. 3.6 показан пример, который выводит информацию из того же топика */turtlesim/cmd\_vel* и позволяет просмотреть содержимое сообщения типа *Twist*.

В примере на рис. 3.6. выводится информация о скоростях движения объекта в симуляторе. Линейная скорость равна нулю, а



```
roscore http... ros-industrial... ros-industrial... ros-industrial...
ros-industrial@ros-training:~$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist
"linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0"
publishing and latching message. Press ctrl-C to terminate
```

Рис. 3.6. Подписка на топик */turtlesim/cmd\_vel*

угловая скорость изменяется по оси  $z$  и равна 1. По умолчанию константы в ROS приведены к системе СИ, соответственно, линейная скорость измеряется в м/с, а угловая – в рад/с.

### 3.5. Сервисы ROS

Модель коммуникации в режиме «сервис» представляет собой двунаправленную синхронную связь между клиентом (*service client*), создающим запрос, и сервером (*service server*), отвечающим на запрос. Синхронность в данном случае означает, что если программа отправит запрос, то до получения ответа от сервера программа находится в этой точке и ожидает результат.

Данный способ удобно использовать для периодической передачи данных или когда существует потребность в синхронной связи (режим «запрос – ответ»). Пример такого взаимодействия показан на рис. 3.7.

Сервер отвечает только тогда, когда есть запрос (*service request*) и клиент, который может получить ответ (*service response*). В отличие от модели работы с топиками, модель сервис работает с «одноразовыми» соединениями. Поэтому, когда цикл «запрос – ответ» завершен, соединение между двумя нодами будет прервано.

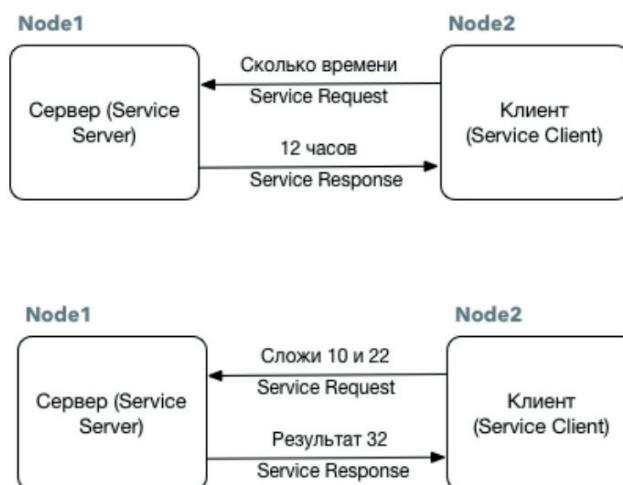


Рис. 3.7. Взаимодействие сервисов в ROS

### 3.5.1. Описание сервисов в ROS

Формат запроса и ответа задается специальным парным сообщением (*message*), в котором есть два сообщения: первое для запроса (*service request*), второе для ответа (*service response*). Файлы с описанием сервисов хранятся в директории *srv* и имеют расширение *.srv*. Подробное описание файла доступно по ссылке: <http://wiki.ros.org/rosbuild/srv>.

В файле описания сервиса первая часть (до разделителя *---*) – это описание сообщения запроса, далее – описание сообщения ответа.

В качестве примера рассмотрим сервис сложения двух чисел *srv/AddTwoInts.srv*. Его структура выглядит следующим образом:

```
uint32 x  
uint32 y  
uint32 sum
```

При таком подходе имя сервиса *AddTwoInts* должно соответствовать имени файла сервиса *AddTwoInts.srv*.

### 3.5.2. Утилита *rosservice*

Аналогично взаимодействию с топиками в *ROS* реализован инструмент взаимодействия с сервисами – *rosservice*. Инструменты утилиты представлены в табл. 3.4.

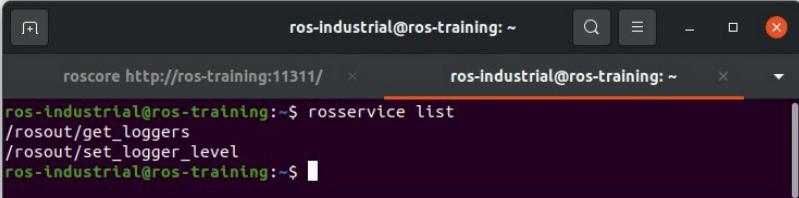
В качестве примера рассмотрим вывод информации о запущенных сервисах в *ROS* по умолчанию. Вызов списка сервисов показан на рис. 3.8.

Для вывода информации из сервиса используется команда *call*, которая позволяет обратиться к сервису и вернуть из него значе-

Таблица 3.4

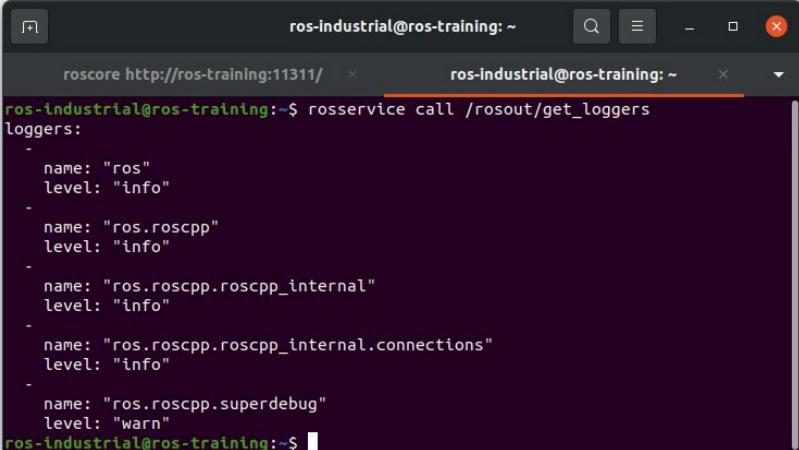
Описание инструментов утилиты *rosservice*

Команда	Описание
<i>rosservice call</i>	Выполнение запроса к серверу
<i>rosservice find</i>	Поиск сервиса по типу
<i>rosservice info</i>	Выводит информацию о сервисе
<i>rosservice list</i>	Выводит список запущенных сервисов
<i>rosservice type</i>	Выводит тип сообщений, используемый сервисом
<i>rosservice uri</i>	Выводит URL сервиса



```
roscore http://ros-training:11311/ x ros-industrial@ros-training: ~ x
ros-industrial@ros-training:~$ rosservice list
/rosout/get_loggers
/rosout/set_logger_level
ros-industrial@ros-training:~$
```

Рис. 3.8. Вывод информации о встроенных сервисах в ROS



```
roscore http://ros-training:11311/ x ros-industrial@ros-training: ~ x
ros-industrial@ros-training:~$ rosservice call /rosout/get_loggers
loggers:
-
  name: "ros"
  level: "info"
-
  name: "ros.roscpp"
  level: "info"
-
  name: "ros.roscpp.roscpp_internal"
  level: "info"
-
  name: "ros.roscpp.roscpp_internal.connections"
  level: "info"
-
  name: "ros.roscpp.superdebug"
  level: "warn"
ros-industrial@ros-training:~$
```

Рис. 3.9. Вывод данных из сервиса *get\_loggers*

ния. На рис. 3.9 продемонстрирован вывод информации из сервиса *get\_loggers*.

Полученная информация необходима при дальнейшем взаимодействии с роботизированной платформой, например, на работе в сервисах хранится системная информация – версия прошивки системной платы.

## 3.6. Основные термины ROS

### 3.6.1. Мастер (*Master*), мастер-нод

Мастер выполняет роль сервера имен для возможности подключения между собой различных нодов. Команда *roscore* запускает сервер мастера, и после этого к нему могут подключиться и заре-

гистрироваться ноды *ROS*. Связь между нодами (обмен сообщениями) невозможна без запущенного мастера.

При запуске *ROS roscore* мастер будет запущен по адресу *URI*, установленному в переменные окружения *ROS\_MASTER\_URI*. По умолчанию адрес использует *IP*-адрес локального ПК и номер порта 11311 [3, с. 102].

### **3.6.2. Нод (*Node*)**

Понятие «нод» относится к наименьшей рабочей единице, используемой в *ROS*. Можно провести аналогию с одной исполняемой программой. *ROS* рекомендует создать один нод для каждой задачи, что позволит легче использовать ее в других проектах. При запуске нод регистрирует информацию о себе в мастере (название нода, типы обрабатываемых сообщений). Зарегистрированный нод может взаимодействовать с другими нодами (получать и отправлять запросы). Важно отметить, что обмен сообщениями между нодами работает без участия мастера (соединение между нодами происходит напрямую). Мастер обеспечивает только единое пространство имен для решения вопроса, куда подключиться к конкретному ноду. Адрес запуска нода берется из переменной окружения *ROS\_HOSTNAME*, которая должна быть определена до запуска. Порт устанавливается на произвольное уникальное значение.

### **3.6.3. Пакет (*Package*)**

Пакет является основной единицей *ROS*. Любое приложение *ROS* оформляется в пакет, в котором определяются конфигурация пакета, ноды, необходимые для работы пакета, зависимости от других пакетов *ROS*.

Работа с пакетами *ROS* очень похожа на работу с пакетами *Linux*. Пакет *ROS* можно как поставить готовым из репозитория пакетов, так и скачать и скомпилировать из исходных кодов.

Поиск доступных пакетов *ROS* возможен на странице <http://wiki.ros.org>.

## **3.7. Стандарты *ROS***

### **3.7.1. Единицы измерений**

Данные, используемые в *ROS*, должны соответствовать единицам СИ – стандарту, наиболее широко используемому в мире.

Это указано в рекомендации *REP-01031*: <http://www.ros.org/reps/rep-0103.html>.

Например, используются длина в метрах, масса в килограммах, время в секундах, ток в амперах, угол в радианах, частота в герцах, сила в ньютонах, мощность в ваттах, напряжение в вольтах и температура в градусах Цельсия.

Все остальные блоки состоят из комбинации вышеупомянутых блоков. Например, скорость перемещения измеряется в метрах в секунду, а скорость вращения – в радианах в секунду.

Рекомендуется соблюдать соответствие единицам СИ, поскольку это позволит другим пользователям снова использовать ваш пакет без необходимости дополнительной конвертации.

### 3.7.2. Координаты X, Y, Z

Оси *x*, *y* и *z* в ROS используют правило правой руки (правило буравчика), как показано на рис. 3.10.

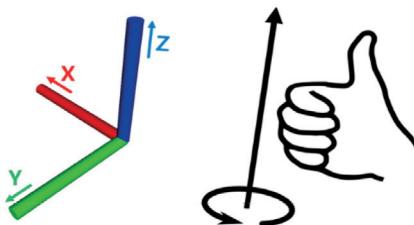


Рис. 3.10. Направление осей в ROS

Направление вперед – это положительное направление оси *X*, которое изображено красным. Направление налево представляет собой положительное направление оси *Y*, которое изображено зеленым. Направление вверх – это положительное направление оси *Z*, а ось обозначена синим. Цветовое кодирование легко запомнить по аббревиатуре *RGB* (стандарт представления цвета *red – green – blue*), которое соответствует осям *X*, *Y*, *Z*.

### 3.7.3. Оси вращения

Следуя правилу буравчика, направление, которое правая рука закручивает, – это положительное направление вращения. Например, если робот вращается с 12 до 9 часов, то, используя радианы

для обозначения угла поворота, получаем, что робот вращается на  $+1.5708$  радиан по оси  $Z$ .

### 3.7.4. Стандарты кодирования

*ROS* рекомендует разработчикам следовать единому стандарту оформления кода (именования и форматирования). Единый формат кодирования уменьшает количество дополнительной работы, которую разработчики часто должны делать при работе с исходным кодом. В частности, это улучшает понимание кода для других участников и облегчает проверку кода [4, с. 61]. В табл. 3.5 представлены основные принципы именования объектов в *ROS*.

Таблица 3.5

#### Принципы именования объектов в *ROS*

Объект	Правило	Пример
Пакет ( <i>Package</i> )	<i>under_scored</i>	<i>rst_ros_package</i>
<i>Topic, Service</i>	<i>under_scored</i>	<i>raw_image</i>
<i>File</i>	<i>under_scored</i>	<i>turtlebot3_fake.cpp</i>
<i>Variable</i>	<i>under_scored</i>	<i>string table_name</i>
<i>Type</i>	<i>CamelCased</i>	<i>typedef int32_t PropertiesNumber</i>
<i>Class</i>	<i>CamelCased</i>	<i>class UrlTable</i>
<i>Structure</i>	<i>CamelCased</i>	<i>struct UrlTableProperties</i>
<i>Function</i>	<i>CamelCased</i>	<i>addTableEntry()</i>
<i>Method</i>	<i>CamelCased</i>	<i>void setNumEntries(int32_t num_entries)</i>
<i>Constant</i>	<i>ALL_CAPITALS</i>	<i>const uint8_t DAYS_IN_A_WEEK = 7</i>
<i>Macro</i>	<i>ALL_CAPITALS</i>	<i>#define PI_ROUNDED 3.0</i>

Данные параметры следует использовать при проектировании объектов в *ROS*.

### 3.8. Создание пакетов ROS

#### 3.8.1. Описание пакета в ROS

Как говорилось ранее, пакет является одной из основных единиц *ROS*. Любое приложение в *ROS* помещается в пакет, в котором

определяются: конфигурация, необходимые программы, зависимости от других пакетов *ROS*, системные зависимости. Пакет представляет собой один проект, в котором происходит работа и выстраивается взаимодействие систем.

Работа с пакетами *ROS* очень похожа на работу с пакетами ОС *Linux*. Пакет *ROS* возможно установить из репозитория пакетов, а также скачать и скомпилировать из исходных кодов.

На текущий момент разработчиками и энтузиастами разработано и опубликовано более 6000 пакетов. Около 3000 пакетов доступны для установки через пакетный менеджер *apt*.

Для того чтобы найти нужный пакет *ROS*, следует обратиться к сайту <http://wiki.ros.org/>.

### 3.8.2. Структура пакета

Пакет *ROS* содержит множество различных файлов. Для того чтобы было проще ориентироваться с файлами любого пакета, сообщество разработчиков рекомендует использовать единообразную файловую структуру пакета, представленную в табл. 3.6.

Таблица 3.6

Файловая структура и описание в *ROS*

Часть структуры	Описание
<i>bin/</i>	Директория, в которой хранятся скомпилированные программы
<i>include/</i>	Директория содержит файлы с заголовками ( <i>headers</i> ) библиотек
<i>launch/</i>	Директория для хранения файлов конфигурации запуска <i>.launch</i>
<i>msg/</i>	Директория для сообщений (для топиков)
<i>src/</i>	Директория для хранения исходников программ
<i>scripts</i>	Директория для хранения исходников скриптов
<i>srv/</i>	Директория для хранения сообщений для использования сервисами ( <i>services</i> )
<i>CMakeLists.txt</i>	Файл формата <i>cmake</i> с инструкциями для установки пакета
<i>package.xml</i>	Файл «манифест» для описания пакета

С точки зрения «системы пакетов» самый важный файл в структуре – это файл *package.xml*. Именно в нем содержится описание пакета, что делает «обычную» директорию с файлами именно пакетом.

Также в этом файле описаны все «внешние» зависимости, необходимые для работы. Если пакет установлен «верно», это гарантирует, что нам не нужно разбираться с дополнительными «установками». Более подробная информация представлена на странице документации: <http://wiki.ros.org/Manifest>.

### 3.8.3. Установка пакета ROS из репозитория

Сообщество разработчиков уже подготовило основные пакеты, которые удобно и быстро возможно установить утилитой *apt*. Пример установки пакета представлен на рис. 3.11.

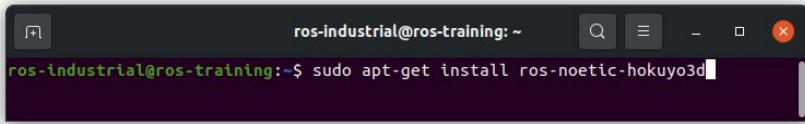
В примере показана установка пакета лазерного дальномера от компании *Hokuyo* для построения 3D-карты пространства. Установка пакетов выполняется с помощью специальной утилиты *apt*, которая встроена в *Ubuntu Linux*. Она также позволяет выполнять установку пакетов не только *ROS*, но других зависимостей и утилит.

Кроме того, данная утилита позволяет просмотреть список всех установленных в *ROS* пакетов. Пример такого запроса представлен на рис. 3.12.

Это не полный список установленных пакетов в *ROS*, а только его часть. Общий список намного больше, и его можно рассмотреть, пролистывая ползунок вниз.

Возникают ситуации, когда необходимо проверить, установлен ли тот или иной компонент *ROS*, но просматривать весь список слишком долго. Утилита *apt* позволяет выполнить поиск конкретного установленного пакета в системе и вывести первичную информацию о нем. Пример такого использования команды представлен на рис. 3.13.

Инструкция *search* сообщает утилите *apt*, что нужно вывести информацию о конкретном пакете. В примере рассмотрен пакет



```
ros-industrial@ros-training: ~$ sudo apt-get install ros-noetic-hokuyo3d
```

Рис. 3.11. Установка пакета ROS из репозитория

```
ros-industrial@ros-training:~$ apt list --installed | grep ros-noetic
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

ros-noetic-actionlib-msgs/now 1.13.1-1focal.20210112.074559 amd64 [installed,upg
radable to: 1.13.1-1focal.20210423.225113]
ros-noetic-actionlib-tutorials/now 0.2.0-1focal.20210112.075151 amd64 [installed
,upgradable to: 0.2.0-1focal.20210922.190120]
ros-noetic-actionlib/now 1.13.2-1focal.20210112.074742 amd64 [installed,upgradab
le to: 1.13.2-1focal.20210922.184944]
ros-noetic-angles/now 1.9.13-1focal.20201015.072755 amd64 [installed,upgradable
to: 1.9.13-1focal.20210727.062146]
ros-noetic-bond-core/now 1.8.6-1focal.20201017.001658 amd64 [installed,upgradab
le to: 1.8.6-1focal.20210922.182438]
ros-noetic-bond/now 1.8.6-1focal.20201015.221826 amd64 [installed,upgradable to:
1.8.6-1focal.20210423.223553]
ros-noetic-bondcpp/now 1.8.6-1focal.20201016.234308 amd64 [installed,upgradable
to: 1.8.6-1focal.20210922.180969]
ros-noetic-bondpy/now 1.8.6-1focal.20201017.001037 amd64 [installed,upgradable t
o: 1.8.6-1focal.20210922.181003]
ros-noetic-camera-calibration-parsers/now 1.12.0-1focal.20210112.082033 amd64 [i
nstalled,upgradable to: 1.12.0-1focal.20210922.192141]
ros-noetic-camera-calibration/now 1.15.3-1focal.20210112.082400 amd64 [installed
,upgradable to: 1.15.3-1focal.20210922.203257]
```

Рис. 3.12. Вывод списка установленных пакетов

```
ros-industrial@ros-training:~$ apt search ros-noetic-rviz
Sorting... Done
Full Text Search... Done
ros-noetic-rviz/focal 1.14.10-1focal.20210928.020110 amd64 [upgradable from: 1.1
4.4-1focal.20210206.032542]
  3D visualization tool for ROS.

ros-noetic-rviz-animated-view-controller/focal 0.2.0-2focal.20210928.023830 amd6
4
  A rviz view controller featuring smooth transitions.

ros-noetic-rviz-animated-view-controller-dbgsym/focal 0.2.0-2focal.20210928.0238
30 amd64
  debug symbols for ros-noetic-rviz-animated-view-controller

ros-noetic-rviz-dbgsym/focal 1.14.10-1focal.20210928.020110 amd64
  debug symbols for ros-noetic-rviz

ros-noetic-rviz imu-plugin/focal 1.2.3-1focal.20210928.023919 amd64
  RVIZ plugin for IMU visualization

ros-noetic-rviz imu-plugin-dbgsym/focal 1.2.3-1focal.20210928.023919 amd64
  debug symbols for ros-noetic-rviz imu-plugin
```

Рис. 3.13. Вывод информации о пакете ROS

*Rviz* – визуализация данных в 3D. Утилита вернула положительный результат выполнения команды, версию пакета, архитектуру ОС и описание функционала пакета.

### 3.8.4. Установка пакетов из исходного кода

Если пакет *ROS* доступен только в виде исходного кода, то существует возможность его установить (для этого чаще используют формулировку «собрать») на свой компьютер и начать его использовать.

Такая операция сложнее, чем использование установки через *apt*, но позволяет устанавливать любые пакеты, а не только те, которые уже были «собраны» заранее.

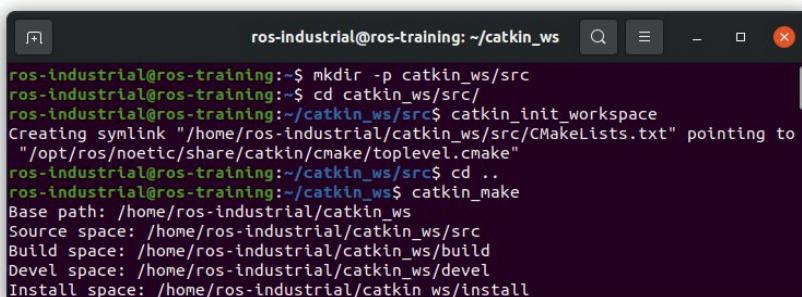
### 3.8.5. Настройка рабочего пространства

Рабочее пространство *ROS* представляет собой место для хранения пакетов пользователя. Пользователь может скачать исходный код пакета и выполнить его сборку или создать свой пакет.

Рекомендация *ROS* – называть рабочее пространство с именем *\_ws* (*work space*). В *ROS* может быть несколько рабочих пространств для работы аналогично тому, как может быть несколько виртуальных окружений для *Python*.

Настройка рабочего окружения представляет собой набор команд терминальной программы и выполняется следующим образом:

1. Необходимо создать директории *catkin\_ws/src*, используя команду *mkdir*. Для создания вложенных директорий используется ключ команды *-p*.
2. Перейти в директорию *catkin\_ws/src*.
3. Выполнить инициализацию рабочего пространства *ROS*, используя команду *catkin\_init\_workspace*.
4. Перейти на уровень директории *catkin\_ws*.
5. Выполнить сборку пакетов рабочего пространства командой *catkin\_make*.



```
ros-industrial@ros-training:~$ mkdir -p catkin_ws/src
ros-industrial@ros-training:~$ cd catkin_ws/src/
ros-industrial@ros-training:~/catkin_ws/src$ catkin_init_workspace
Creating symlink "/home/ros-industrial/catkin_ws/src/CMakeLists.txt" pointing to
"/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
ros-industrial@ros-training:~/catkin_ws/src$ cd ..
ros-industrial@ros-training:~/catkin_ws$ catkin_make
Base path: /home/ros-industrial/catkin_ws
Source space: /home/ros-industrial/catkin_ws/src
Build space: /home/ros-industrial/catkin_ws/build
Devel space: /home/ros-industrial/catkin_ws/devel
Install space: /home/ros-industrial/catkin_ws/install
```

Рис. 3.14. Создание рабочего пространства *ROS*

Данные команды представлены на рис. 3.14.

*Catkin* – набор включает в себя ряд инструментов для работы с пространством *ROS* и выполняет функции, аналогичные программе *CMake*. Папка сборки пакетов в рабочем пространстве появляется набор директорий для взаимодействия с *ROS*. В основном используется папка *src*, куда пользователь помещает свои проекты или пакеты для сборки из исходного кода.

### 3.8.6. Сборка пакета из исходного кода

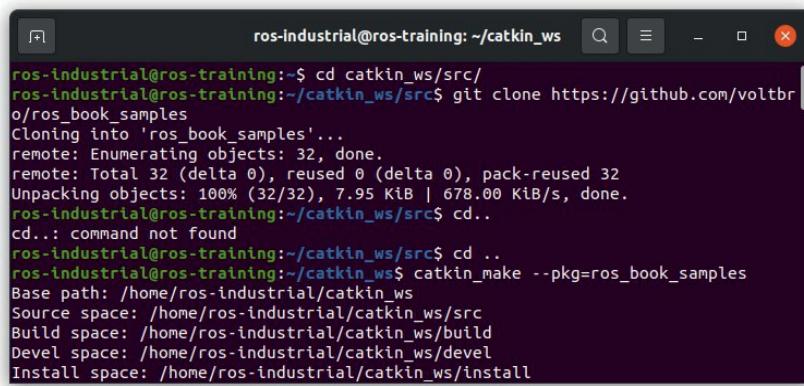
В основном все пакеты, которые пользователь «собирает» вручную, расположены на хостинге проектов *GitHub*.

Для закачки пакета используется стандартная команда *git* и инструмент *clone*. Использование инструмента представлено на рис. 3.15.

В примере используется пакет примеров, скачанный из официального репозитория компании *VoltBro*, изготавлившей сервисного робота, с которым будет происходить взаимодействие.

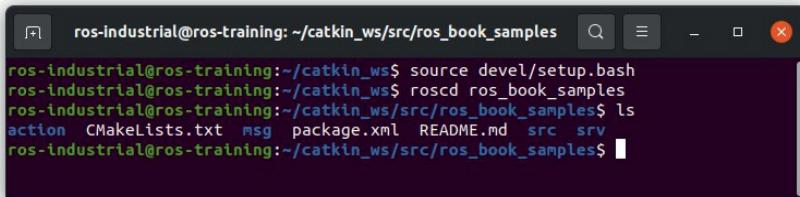
Сообщение об успешной сборке будет выведено в командную строку, если процесс дошел до 100%. После этого следует проверить, существует ли в *ROS* данный пакет. Для этого надо провести операции, продемонстрированные на рис. 3.16.

В данном примере проверка выполняется с использованием команды *ROS roscl*, которая аналогично утилите *cd* позволяет переходить по директориям ОС. По умолчанию собранные пакеты не видны в оболочке терминальной программы – их необходимо про-



```
ros-industrial@ros-training:~$ cd catkin_ws/src/
ros-industrial@ros-training:~/catkin_ws/src$ git clone https://github.com/voltbro/ro
o/ro
os_book_samples
Cloning into 'ro
os_book_samples'...
remote: Enumerating objects: 32, done.
remote: Total 32 (delta 0), reused 0 (delta 0), pack-reused 32
Unpacking objects: 100% (32/32), 7.95 KiB | 678.00 KiB/s, done.
ros-industrial@ros-training:~/catkin_ws/src$ cd..
cd..: command not found
ros-industrial@ros-training:~/catkin_ws/src$ cd ..
ros-industrial@ros-training:~/catkin_ws$ catkin_make --pkg=ro
os_book_samples
Base path: /home/ro
os-industrial/catkin_ws
Source space: /home/ro
os-industrial/catkin_ws/src
Build space: /home/ro
os-industrial/catkin_ws/build
Devel space: /home/ro
os-industrial/catkin_ws/devel
Install space: /home/ro
os-industrial/catkin_ws/install
```

Рис. 3.15. Скачивание и установка пакета *ROS*



```
ros-industrial@ros-training:~/catkin_ws/src/ros_book_samples$ source devel/setup.bash
ros-industrial@ros-training:~/catkin_ws$ rosdep install --from-paths src --ignore-src --os=ubuntu:groovy
ros-industrial@ros-training:~/catkin_ws$ rosdep install --from-paths src --ignore-src --os=ubuntu:groovy
ros-industrial@ros-training:~/catkin_ws$ cd src/ROS_Industrial
ros-industrial@ros-training:~/catkin_ws/src/ROS_Industrial$ ls
action CMakeLists.txt msg package.xml README.md src srv
ros-industrial@ros-training:~/catkin_ws/src/ROS_Industrial$
```

Рис. 3.16. Проверка установленного пакета

писать в оболочку. Для этого используется команда *source*, которая запускает подготовленный список изменений, внесенный в файл *setup.bash*.

### 3.9. Создание пакета ROS

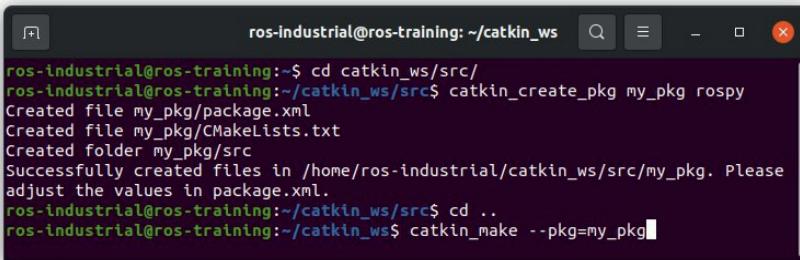
#### 3.9.1. Модуль *catkin*

*Catkin* – это модуль, который служит для «сборки» пакетов. Сборкой называют ряд определенных действий по установке пакета в систему *ROS*. Это могут быть операции копирования файлов, компиляция программ (например, для языка *C*), генерация сообщений.

Для описания необходимой операции по установке используется стандарт *CMake* (*Cross Platform Make*), а сама последовательность операций описана в файле *CMakeLists.txt*.

#### 3.9.2. Создание пользовательского пакета

Создание своего пакета чем-то напоминает сборку пакета из исходников с той лишь разницей, что сам пользователь выступает в качестве источника исходного кода. Пакет создается внутри рабо-



```
ros-industrial@ros-training:~/catkin_ws$ cd catkin_ws/src/
ros-industrial@ros-training:~/catkin_ws/src$ catkin_create_pkg my_pkg rospy
Created file my_pkg/package.xml
Created file my_pkg/CMakeLists.txt
Created folder my_pkg/src
Successfully created files in /home/ros-industrial/catkin_ws/src/my_pkg. Please
adjust the values in package.xml.
ros-industrial@ros-training:~/catkin_ws/src$ cd ..
ros-industrial@ros-training:~/catkin_ws$ catkin_make --pkg=my_pkg
```

Рис. 3.17. Создание пользовательского пакета

чего пространства в директории *src* [5]. При создании пакета важно понимать, какие зависимости и сообщения будут использоваться в нем. Пример создания пользовательского пакета представлен на рис. 3.17.

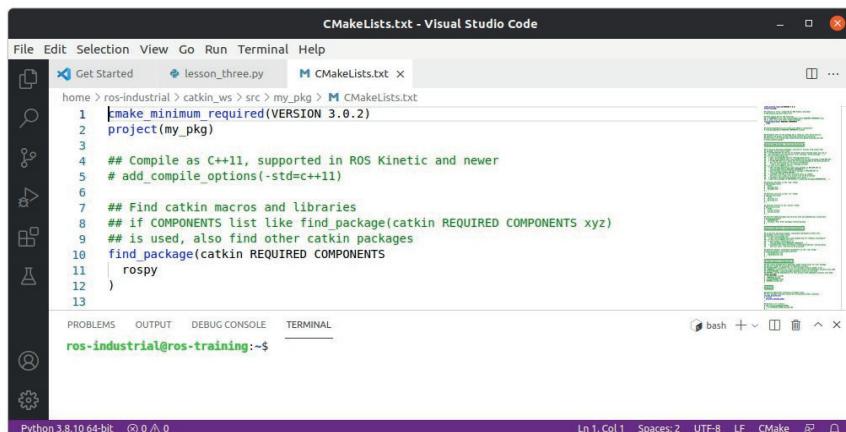
Создание пакета включает в себя ряд простых шагов. Нет необходимости создавать отдельную папку для проекта, так как это делает специальная утилита *catkin\_create\_pkg*. Важно помнить, что создание пакета выполняется из директории *src*.

В примере показано, что пользователь создает пакет с именем *my\_pkg*. После имени пакета идет список зависимостей, которые разделяются пробелами.

### 3.9.3. Работа с зависимостями

Обычно при создании пакета пользователь знает, какие пакеты *ROS* ему понадобятся. Существует возможность редактирования зависимостей пакета. Данная операция выполняется внесением списка зависимостей в файл *CMakeLists.txt*. Данный файл лучше всего открывать, используя среду разработки *Visual Studio Code*, так как в ней есть подсветка необходимого синтаксиса. Вывод данных файла *CMakeLists.txt* представлен на рис. 3.18.

Данный файл является файлом сборки пакета *ROS*, в нем достаточно много полей, которые описывают принципы взаимодействия зависимостей файлов и директорий пакета. Файл наполнен комментариями, которые сообщают пользователю функционал того

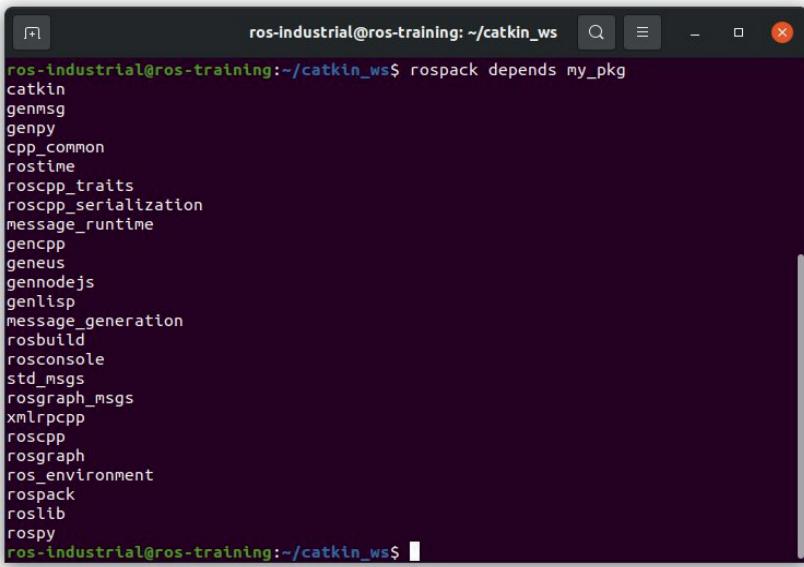


The screenshot shows the Visual Studio Code interface with the title bar "CMakeLists.txt - Visual Studio Code". The main editor area displays the following code:

```
1  cmake_minimum_required(VERSION 3.0.2)
2  project(my_pkg)
3
4  ## Compile as C++11, supported in ROS Kinetic and newer
5  # add_compile_options(-std=c++11)
6
7  ## Find catkin macros and libraries
8  ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9  ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   rospy
12 )
13
```

Below the editor, the status bar shows "ros-industrial@ros-training:~\$". The bottom right corner of the status bar has icons for bash, terminal, and file operations.

Рис. 3.18. Содержимое файла *CMakeLists.txt*



```
ros-industrial@ros-training:~/catkin_ws$ rosdep depends my_pkg
catkin
genmsg
genpy
cpp_common
rostime
roscpp_traits
roscpp_serialization
message_runtime
gen cpp
gen eus
gen nodejs
gen lisp
message_generation
rosbuild
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
roscpp
rosgraph
ros_environment
rosdep
roslib
rospy
ros-industrial@ros-training:~/catkin_ws$
```

Рис. 3.19. Вывод всех зависимостей пакета *my\_pkg*

или иного блока. Для внесения зависимостей используется блок *find\_package*. Утилиты *catkin* также во время создания «подтягивают» необходимые зависимости. В этом можно убедиться, если запросить зависимости пакета, созданного пользователем. Пример вывода всех зависимостей пакета представлен на рис. 3.19.

Для вывода зависимостей пакета используется утилита *rosdep* с инструкцией *depends*. Данные зависимости необходимы для корректной работы пакета в *ROS* и подключаются в момент, когда происходит обращение к ним. Это еще один плюс работы с *ROS*, который позволяет эффективно вести разработку, не обращая внимания на дополнение списка зависимостей пакетов.

### 3.9.4. Библиотека *rospy*

В предыдущем подразделе при создании пакета использовалась зависимость *rospy*. *Rospy* – это клиентская библиотека *Python* для *ROS*, которая позволяет взаимодействовать с топиками и сервисами с помощью языка программирования *Python*. Многие утилиты *Python* написаны с использованием данной библиотеки, например *rostopic* и *rosservice*.

### 3.10. Написание издателя на языке Python

Одним из простых примеров взаимодействия языка программирования *Python* и *ROS* является написание кода программы издателя. *Python* использует библиотеку *rospy* в качестве модуля, в котором описаны методы взаимодействия и работы с *ROS*.

Создание программы начинается с создания файла, у которого установлено расширение *.py*. Для удобства рекомендуется использовать среду разработки *Visual Studio Code*. Пример программы издателя представлен на рис. 3.20.

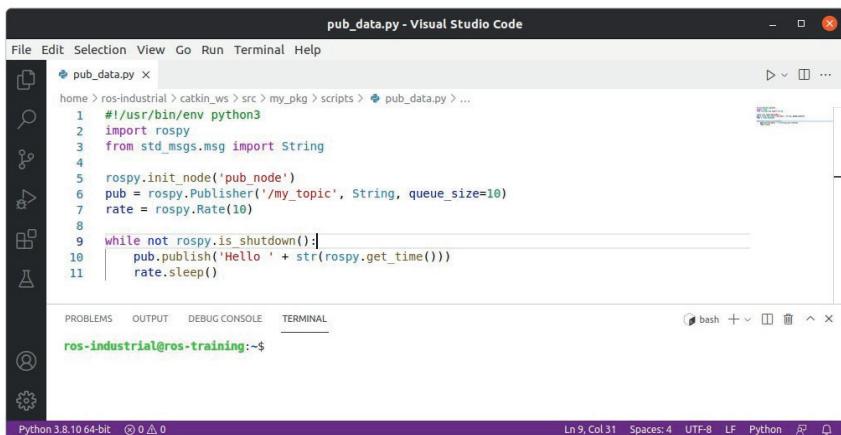
Для удобного взаимодействия с пакетом в *ROS* и разделения отдельных файлов по директориям рекомендуется для файлов с расширением *.py* создать отдельную директорию внутри пакета и назвать ее *scripts*.

Код программы использует последовательность шагов, что позволяет запустить код не только напрямую из окна *VSCode*, но и напрямую из терминала *Linux*.

Принцип написания программы издателя для взаимодействия *Python* с *ROS* состоит из следующих шагов:

1. Импорт необходимых модулей. Импортируемые модули *rospy* и *string* позволяют обеспечить соответствие кода библиотекам *ROS* для передачи данных через стандартные сообщения.

2. Инициализация нода. Важной частью любого нода (узла) является его инициализация. Для того чтобы *ROS* поняла, что запущен не просто скрипт программы на языке *Python*, а полноценный



The screenshot shows the Visual Studio Code interface with a Python file named 'pub\_data.py' open. The code is a ROS node that publishes a string message every 10 seconds. The code is as follows:

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String

rospy.init_node('pub_node')
pub = rospy.Publisher('my_topic', String, queue_size=10)
rate = rospy.Rate(10)

while not rospy.is_shutdown():
    pub.publish('Hello ' + str(rospy.get_time()))
    rate.sleep()
```

The interface includes a top menu bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a tab bar showing 'pub\_data.py - Visual Studio Code'. Below the editor are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab shows the command 'ros-industrial@ros-training:~\$'. At the bottom, there's a status bar with 'Python 3.8.10 64-bit', 'Ln 9, Col 31', 'Spaces: 4', 'UTF-8', 'LF', 'Python', and other icons.

Рис. 3.20. Пример кода издателя на языке Python

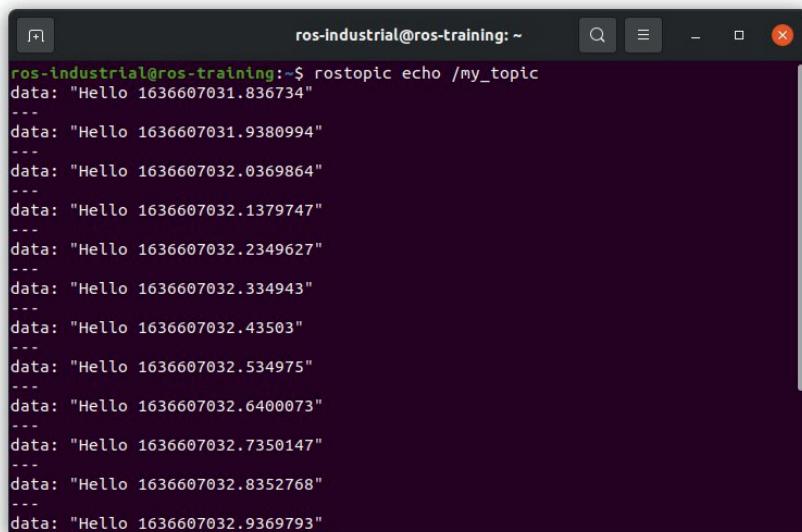
узел системы, у которого есть возможности внутрипроцессного взаимодействия, необходимо произвести инициализацию узла и присвоить ему имя. Через это имя будет происходить обращение внутри *ROS*.

3. Инициализация издателя. Необходимо сообщить программе, что происходит взаимодействие в качестве издателя и объявить топик, в который узел будет публиковать сообщения. В модуле *rospy* реализован класс *publisher*, который выполняет данный функционал. Для корректной работы необходимо при создании объекта данного класса указать:

- а) имя топика, в который публикуются данные (начинается с символа «/»);
- б) тип сообщения топика (в примере – */my\_topic*);
- с) размер очереди на отправку сообщений (*queue\_size=10*).

4. Объявление значения частоты публикации сообщений. Данный параметр устанавливает частоту (Гц), с которой сообщения публикуются в топик.

5. Цикл отправки сообщений. В цикле в качестве условия указано условие вида *rospy.is\_shutdown()*, метод проверки на факт наличия запущенного мастера в системе. Необходимо, чтобы на стадии запуска узла *roscore* был активен и запущенный узел мог обмени-



```
ros-industrial@ros-training:~$ rostopic echo /my_topic
data: "Hello 1636607031.836734"
---
data: "Hello 1636607031.9380994"
---
data: "Hello 1636607032.0369864"
---
data: "Hello 1636607032.1379747"
---
data: "Hello 1636607032.2349627"
---
data: "Hello 1636607032.334943"
---
data: "Hello 1636607032.43503"
---
data: "Hello 1636607032.534975"
---
data: "Hello 1636607032.6400073"
---
data: "Hello 1636607032.7350147"
---
data: "Hello 1636607032.8352768"
---
data: "Hello 1636607032.9369793"
```

Рис. 3.21. Вывод информации из топика */my\_topic*

ваться данными. Также данный метод реагирует на закрытие нода в терминале с помощью комбинаций клавиш *Ctrl + C*.

6. Публикация сообщения. *Publish* – метод, который отправляет содержимое сообщения в топик */my\_topic*. Кроме того, одновременно с сообщением передается время работы запущенного узла.

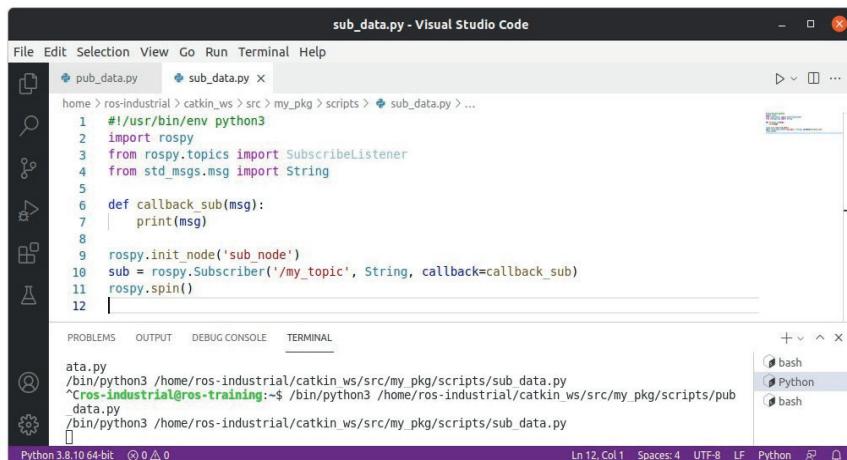
7. Пауза между отправкой. Для того чтобы выдерживать заданную частоту публикации сообщений, в каждой итерации цикла выполняется ожидание времени окончания отправки сообщения.

Запуск узла возможен из терминала или через окно *VSCode*. В момент запуска происходит инициализация всех объектов и в списке топиков появляется новый */my\_topic*. Пример вывода топиков показан на рис. 3.21.

Запросить список всех топиков, запущенных в системе, можно с помощью команды *rostopic list*. При выводе сообщения типа *std\_msgs* формируют массив *data*, в который помещают необходимое сообщение.

### 3.11. Написание подписчика на языке Python

Написание подписчика на языке *Python* похоже на написание издателя, кроме одной важной части – функции обратной связи. В концепции *ROS* узел подписчик «цепляется» к топику и получает сообщения, которые передает издатель. На рис. 3.22 представлен код подписчика на языке *Python*.



The screenshot shows a Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** Two files are open: "pub\_data.py" and "sub\_data.py". The "sub\_data.py" file contains the following Python code:

```
1 #!/usr/bin/env python3
2 import rospy
3 from rospy.topics import SubscribeListener
4 from std_msgs.msg import String
5
6 def callback_sub(msg):
7     print(msg)
8
9 rospy.init_node('sub_node')
10 sub = rospy.Subscriber('/my_topic', String, callback=callback_sub)
11 rospy.spin()
```
- Terminal:** Shows command-line output:

```
ata.py
/bin/python3 /home/ros-industrial/catkin_ws/src/my_pkg/scripts/sub_data.py
^Cros-Industrial@ros-training:~$ /bin/python3 /home/ros-industrial/catkin_ws/src/my_pkg/scripts/pub_data.py
/bin/python3 /home/ros-industrial/catkin_ws/src/my_pkg/scripts/sub_data.py
[]
```
- Status Bar:** Python 3.8.10 64-bit, Ln 12, Col 1, Spaces: 4, UTF-8, LF, Python.

Рис. 3.22. Код подписчика на языке *Python*

Написание кода издателя состоит из следующих шагов:

1. Импорт необходимых модулей. Так же, как и в примере с издателем, используются модули *rospy* и *string* для взаимодействия с получаемыми из топика сообщениями.

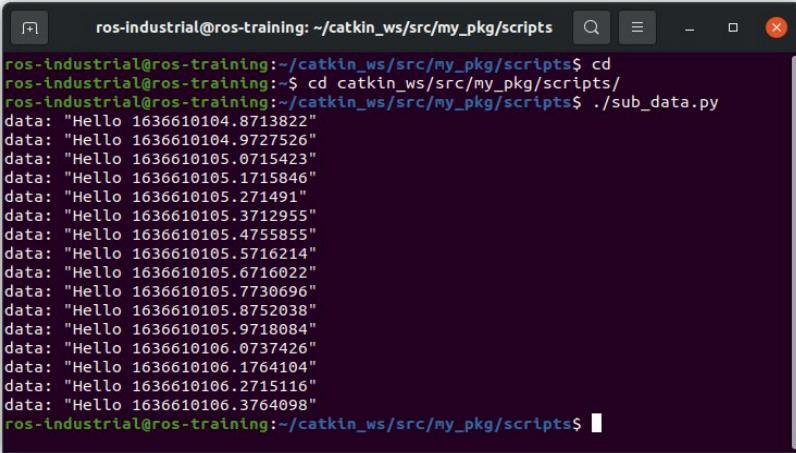
2. Объявление функции обработчика данных. Функция обработчика необходима для того, чтобы данные были получены лишь в момент, когда они приходят. Это значит, если узел издатель не запущен, а узел подписчик работает, то данные в топик не поступают, а следовательно, узел может выполнять иные задачи, которые прописаны в его коде.

3. Инициализация нода. Создание объекта, который содержит имя узла подписчика.

4. Инициализация подписчика. Создание экземпляра класса подписчика (*subscriber*). Создание данного объекта позволяет «сказать» системе *ROS*, что наш нод выступает в качестве подписчика. Конструктор данного экземпляра класса содержит следующие поля:

- a) имя топика, на который подписан узел (*/me\_topic*);
- b) тип сообщения;
- c) имя функции обработчика данных.

5. *rospy.spin()* – строка кода, которая запускает нод и не позволяет программе завершиться. В отличии от кода издателя код подписчика работает иным образом, и ему нет необходимости писать бесконечный цикл. Завершить программу можно через сочетание клавиш *Ctrl + C*. Запуск кода подписчика представлен на рис. 3.23.



The screenshot shows a terminal window titled "ros-industrial@ros-training: ~/catkin\_ws/src/my\_pkg/scripts". The command run was "cd catkin\_ws/src/my\_pkg/scripts/" followed by "./sub\_data.py". The output consists of multiple lines of text, each starting with "data: " followed by a string of characters and numbers, representing ROS messages being received.

```
ros-industrial@ros-training:~/catkin_ws/src/my_pkg/scripts$ cd
ros-industrial@ros-training:~/catkin_ws/src/my_pkg/scripts/
ros-industrial@ros-training:~/catkin_ws/src/my_pkg/scripts$ ./sub_data.py
data: "Hello 1636610104.8713822"
data: "Hello 1636610104.9727526"
data: "Hello 1636610105.0715423"
data: "Hello 1636610105.1715846"
data: "Hello 1636610105.271491"
data: "Hello 1636610105.3712955"
data: "Hello 1636610105.4755855"
data: "Hello 1636610105.5716214"
data: "Hello 1636610105.6716022"
data: "Hello 1636610105.7730696"
data: "Hello 1636610105.8752038"
data: "Hello 1636610105.9718084"
data: "Hello 1636610106.0737426"
data: "Hello 1636610106.1764194"
data: "Hello 1636610106.2715116"
data: "Hello 1636610106.3764098"
ros-industrial@ros-training:~/catkin_ws/src/my_pkg/scripts$
```

Рис. 3.23. Запуск кода программы подписчика

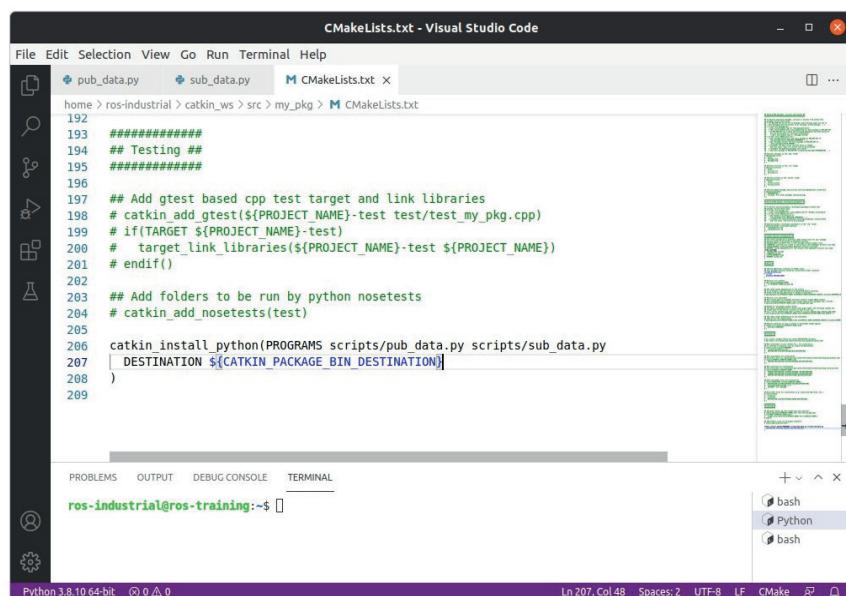
В коде программы возможно существование нескольких объектов издателей и подписчиков, которые взаимодействуют с другими узлами, запущенными в системе. Возможно объединение кодов подписчика и издателя.

### 3.12. Сборка пакетов

На текущий момент запуск программ выполнялся средствами языка программирования *Python* без сборки пакета. Сборка пакета позволяет зафиксировать в рабочем окружении *ROS* зависимости пользовательского пакета, а также ноды, которые были написаны пользователем.

Пакет пользователя не знает о существовании узлов, так как в его файле *CMakeLists.txt* не указана данная информация. Для внесения информации необходимо дополнить файл содержимым, представленным на рис. 3.24.

Строки о сборке скриптов вносятся в конец файла. В примере внесены файлы *pub\_data.py*, содержащий код издателя, и *sub\_data.py*, содержащий код подписчика. При написании имен файлов необходимо указывать директорию, в которой лежат скрипты, тогда при



```
File Edit Selection View Go Run Terminal Help
pub_data.py sub_data.py CMakeLists.txt
home > ros-industrial > catkin_ws > src > my_pkg > CMakeLists.txt
192
193 ######
194 ## Testing ##
195 #####
196
197 ## Add gtest based cpp test target and link libraries
198 # catkin_add_gtest(${PROJECT_NAME}-test test/test_my_pkg.cpp)
199 # if(TARGET ${PROJECT_NAME}-test)
200 #   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
201 # endif()
202
203 ## Add folders to be run by python nosetests
204 # catkin_add_nosetests(test)
205
206 catkin_install_python(PROGRAMS scripts/pub_data.py scripts/sub_data.py
207   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
208 )
209

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ []
+ ^ x
bash Python bash
Python 3.8.10 64-bit 0 △ 0 Ln 207, Col 48 Spaces:2 UTF-8 LF CMake ⌂ ⌂
```

Рис. 3.24. Дополнение файла *CMakeLists.txt*

```
ros-industrial@ros-training:~/catkin_ws$ catkin_make --pkg=my_pkg
Base path: /home/ros-industrial/catkin_ws
Source space: /home/ros-industrial/catkin_ws/src
Build space: /home/ros-industrial/catkin_ws/build
Devel space: /home/ros-industrial/catkin_ws/devel
Install space: /home/ros-industrial/catkin_ws/install
#####
##### Running command: "make cmake_check_build_system" in "/home/ros-industrial/catkin_ws/build"
#####
##### Running command: "make -j2 -l2" in "/home/ros-industrial/catkin_ws/build/my_pkg"
#####
ros-industrial@ros-training:~/catkin_ws$
```

Рис. 3.25. Сборка пакета *my\_pkg*

сборке пакета утилита *catkin* не выдаст ошибку об отсутствии файлов. Перечисление отдельных узлов происходит через пробел.

Сборка пакетов выполняется с помощью команды *catkin\_make*, выполнение которой показано на рис. 3.25.

Команда собрала единственный пакет, указанный в ее аргументах. Кроме того, сборка пакета прошла успешно, так как не возникло непредвиденных ошибок.

### 3.13. Запуск пакетов

Управление запуском пакетов и узлов выполняется встроенными утилитами. Таких утилит две: *rosrun* и *roslaunch*.

#### 3.13.1. Утилита *rosrun*

Данная утилита выполняет роль команды, которая запускает только один нод в выбранном пакете. Пример запуска узла изданетя представлен на рис. 3.26.

```
ros-industrial@ros-training:~/catkin_ws$ rosrun my_pkg
pub_data.py  sub_data.py
ros-industrial@ros-training:~/catkin_ws$ rosrun my_pkg pub_data.py
```

Рис. 3.26. Запуск узла из пакета *my\_pkg*

Из примера видно, что утилита *rosrun* выводит два узла, которые есть в пакете *my\_pkg*. Это позволяет сделать вывод, что проделанные в предыдущем пункте операции верны и сборка прошла успешно.

Для запуска нескольких узлов требуется снова использовать утилиту *rosrun*. При таком подходе каждый новый узел необходимо запускать в новом окне или в скрытом режиме. Первый подход превращает экран в большой набор открытых окон терминала, а второй не является удобным, когда нужно остановить узел. Для решения данной проблемы в *ROS* реализована утилита *roslaunch*.

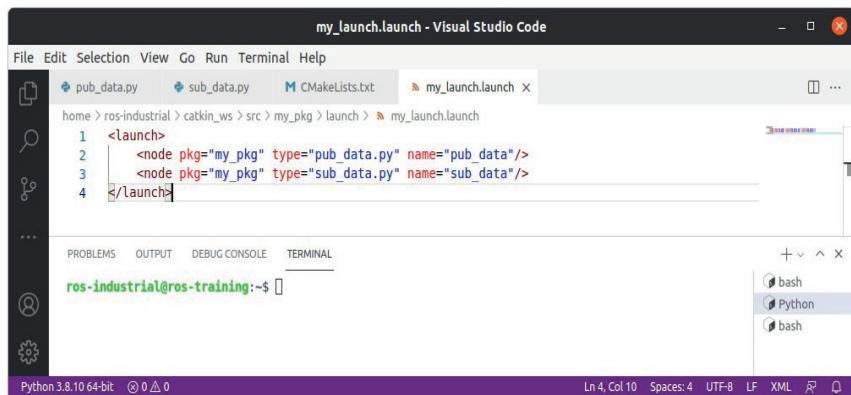
### 3.13.2. Утилита roslaunch

Утилита позволяет объединить несколько узлов (нодов) в одном файле и запускать их одновременно. Кроме того, возможно дополнить настройками исполняемые файлы узлов, передавая в них параметры по умолчанию.

Файлы для утилиты *roslaunch* имеют расширение *.launch* и используют синтаксис формата *XML*. Создание файлов *.launch* удобно выполнять в среде разработки *VSCode*. Пример такого файла представлен на рис. 3.27.

*Рекомендуется создать в пакете отдельную директорию с именем *launch*, в которую будут помещены все файлы с данным расширением.*

*Перед запуском файла рекомендуется выполнить повторную сборку пакета.*



```
my_launch.launch - Visual Studio Code
File Edit Selection View Go Run Terminal Help
pub_data.py sub_data.py CMakeLists.txt my_launch.launch ...
home > ros-industrial > catkin_ws > src > my_pkg > launch > my_launch.launch
1 <launch>
2   <node pkg="my_pkg" type="pub_data.py" name="pub_data"/>
3   <node pkg="my_pkg" type="sub_data.py" name="sub_data"/>
4 </launch>

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$ 
+ ^ x
bash Python bash
Ln 4, Col 10 Spaces: 4 UTF-8 LF XML ⌂
Python 3.8.10 64-bit ⌂ 0 ⌂ 0
Ln 4, Col 10 Spaces: 4 UTF-8 LF XML ⌂
```

Рис. 3.27. Пример файла *.launch*

Код файла *.launch* строится по следующим правилам:

1. Код начинается со слова *launch* в угловых скобках. Так как синтаксис файла соответствует синтаксису *XML*, то каждый отдельный объект помещается в теги, ограниченные угловыми скобками. Заканчивается тег *launch* строкой */launch*.

2. Тег запуска узла. Указание на запуск каждого узла выполняется с помощью тега *node*, после которого идет описание структуры запускаемого узла:

a) *pkg* – имя пакета, который будет запускаться;

b) *type* – название узла (нода) для запуска;

c) *name* – исполняемое имя. В большинстве случаев совпадает с именем узла. Использование разных имен позволяет создавать несколько экземпляров одного и того же узла

Запуск файла *.launch* выполняется утилитой *roslaunch*. После имени утилиты вводится имя пакета. Последним шагом является указания названия файла *.launch*. Пример запуска утилиты представлен на рис. 3.28.

Данный файл *.launch* запустил два узла, прописанных в его тегах, с именами *pub\_data* и *sub\_data*. Если вывести список всех за-

```
ros-industrial@ros-training:~/catkin_ws$ roslaunch my_pkg my_launch.launch
... logging to /home/ros-industrial/.ros/log/a8a58ac6-429e-11ec-8063-b5275d8a945
c/roslaunch-ros-training-41137.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ros-training:46483/
SUMMARY
=====
PARAMETERS
  * /rosdistro: noetic
  * /rosversion: 1.15.9

NODES
/
  pub_data (my_pkg/pub_data.py)
  sub_data (my_pkg/sub_data.py)

ROS_MASTER_URI=http://localhost:11311

process[pub_data-1]: started with pid [41153]
process[sub_data-2]: started with pid [41154]
```

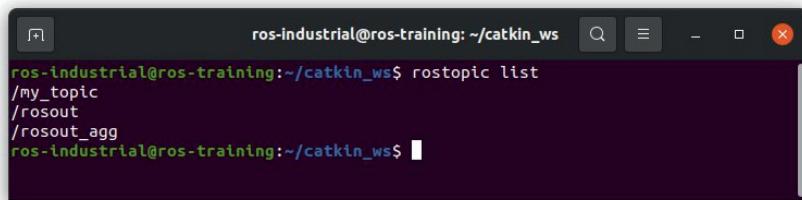
Рис. 3.28. Запуск файла *.launch*

пущенных топиков командой `rostopic list`, то можно заметить, что издатель передает данные в топик. Список запущенных топиков представлен на рис. 3.29.

Инструкция `echo` выведет информацию из топика `/my_topic` в окно терминала.

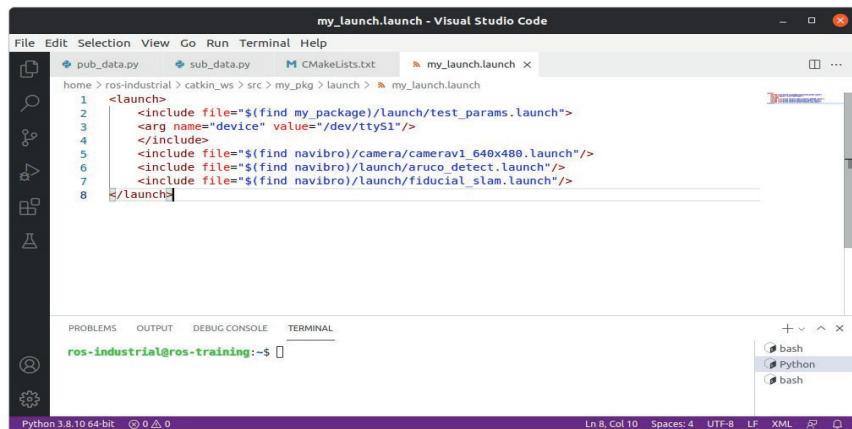
В реальных проектах запускаются десятки нодов. Конфигурировать каждый из них в одном файле не всегда удобно. К тому же обычно сторонние пакеты уже содержат подходящие файлы `.launch`. Поэтому существует механизм `include`, который позволяет подключать другие файлы запуска. Пример программы представлен на рис. 3.30.

В этом примере выполняется запуск файла `test_params.launch`, который находится в пакете `navibro`. Выполним настройку данного файла через устройство `/dev/ttyS1`, а также подключим три других файла `.launch` из другого пакета.



```
ros-industrial@ros-training:~/catkin_ws$ rostopic list
/my_topic
/rosout
/rosout_agg
ros-industrial@ros-training:~/catkin_ws$
```

Рис. 3.29. Вывод списка топиков



```
my_launch.launch - Visual Studio Code
File Edit Selection View Go Run Terminal Help
pub_data.py sub_data.py CMakeLists.txt my_launch.launch
home > ros-industrial > catkin_ws > src > my_pkg > launch > my_launch.launch
1  <launch>
2    |<include file="$(find my_package)/launch/test_params.launch">
3    |  <arg name="device" value="/dev/ttyS1"/>
4    |</include>
5    |<include file="$(find navibro)/camera/camerav1_640x480.launch"/>
6    |<include file="$(find navibro)/launch/aruco_detect.launch"/>
7    |<include file="$(find navibro)/launch/fiducial_slam.launch"/>
8  </launch>

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ros-industrial@ros-training:~$
```

Рис. 3.30. Запуск нескольких файлов `.launch`

### **3.14. Практические задания**

**Задание 1.** Написать программу «Издатель», которая в топик */numbers* отправляет случайные числа от 1 до 99 с частотой 5 Гц (*num\_pub.py*).

**Задание 2.** Написать программу (*base\_test.py*), которая:

- 1) подпишется на топик */numbers*;
- 2) для пяти последовательно полученных цифр рассчитает среднее арифметическое. После расчета среднего арифметического программа должна начать рассчитывать последовательность заново;
- 3) программа публикует результат вычисления в топик */result*;
- 4) при получении максимального числа из ранее полученных опубликует данные в топик */max\_val*.

## **4. ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРНЫХ ПЛАТ**

### **4.1. Основная информация о системной плате робота**

Одной из основных частей всех роботов является система сбора данных и управления периферией. Такие системы реализуются с использованием микроконтроллерных систем. На рынке представлено множество микроконтроллеров от разных производителей и компаний, но одни из самых популярных микроконтроллеров, которые нашли применение во многих областях техники: от любительских устройств до 3D-принтеров и станков, – это микроконтроллеры компании *Microchip Technology* семейства *AVR*.

*Ранее микроконтроллеры выпускались компанией Atmel. В 2016 г. компания Atmel была поглощена компанией Microchip Technology, которая продолжает выпускать часть продуктов под наименование Atmel. Многие микроконтроллеры сохранили наименование Atmel: ATmegaXX/ATXXSAMXX и др. Часть новых микроконтроллеров имеют наименование Microchip AtmegaXX/ATXXSAMXX. (XX – идентификаторы микроконтроллеров, состоящие из цифр и букв.)*

Данное семейство представляет восьмибитные микроконтроллеры, обладающие ядром *RISC* и низким энергопотреблением. У микроконтроллеров достаточно много различных характеризующих их параметров, таких как количество таймеров, количество входов/выходов, количество АЦП (аналого-цифровых преобразователей) и ЦАП (цифроаналоговых преобразователей), объем памяти (флеш-память, ОЗУ, EEPROM и др.), наличие и количество интерфейсов (*CAN, UART, SPI, I2C* и др.).

Данное пособие не ставит задачу научить разбираться во всех семействах микроконтроллеров и уметь программировать каждое. Оно дает основные навыки, необходимые для написания кода под конкретный микроконтроллер и микроконтроллерную плату.

В качестве микроконтроллерной платы выступает популярная плата *Arduino Mega2560*, построенная на базе микроконтроллера *Atmega2560*. Краткие характеристики микроконтроллерной платы представлены в табл. 4.1.

Представленный микроконтроллер обладает как достаточным количеством интерфейсов ввода/вывода данных, так и возможностями для хранения прошивки и данных.

*ATmega2560 – один из самых распространенных микроконтроллеров в мире. Он стал основой для большого количества устройств и приборов, но наибольшее распространение он получил в кругах де-*

Таблица 4.1

**Краткие характеристики микроконтроллерной платы  
*Arduino Mega2560***

Наименование	Значение
Микроконтроллер	<i>ATmega2560</i>
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7–12 В
Входное напряжение (предельное)	6–20 В
Цифровые входы/выходы	54 (14 из которых могут работать так же, как выходы ШИМ)
Аналоговые входы	16
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3.3 В	50 мА
Флеш-память	256 Кб (из которых 8 Кб используются для загрузчика)
ОЗУ	8 Кб
Энергонезависимая память (EEPROM)	4 Кб
Тактовая частота	16 МГц

шевых и эффективных 3D-принтеров и ЧПУ-станков, что позволило популяризировать 3D-печать среди массового сегмента DIY-мейкеров. На текущий 2021 г. он также остается одним из главных микроконтроллеров для плат управления 3D-принтерами, но постепенно уходит на второй план по сравнению с решениями на 32-битных МК системах (таких, как *STM32F405* и др.).

Системная плата робота представлена на рис. 4.1 и 4.2.

Назначение кнопок на плате:

*Restart* – нажатие кнопки перезапускает программу МК *STM32*. Происходит чтение *rosparams*, сбрасываются значения датчика *IMU*, одометрии, текущих скоростей колес. Кнопку необходимо использовать при изменении *rosparams* на стороне *Raspberry*. Пере- запуск идет около 30 секунд.

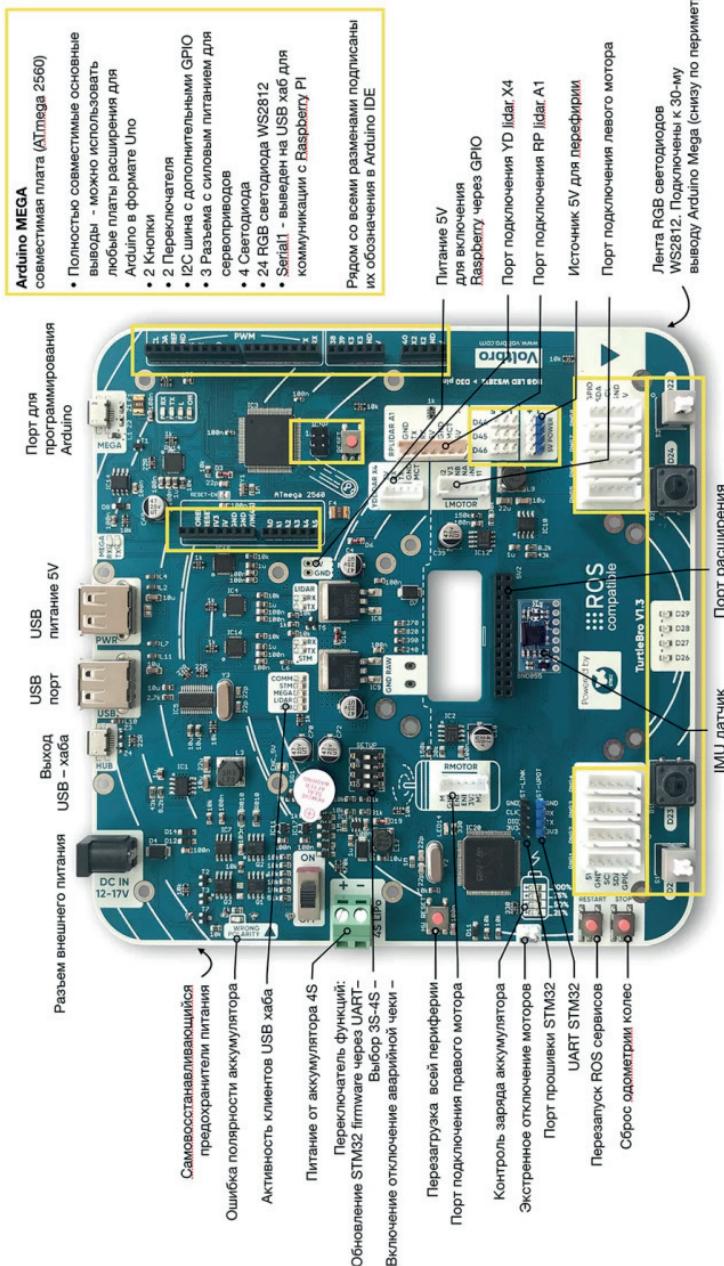
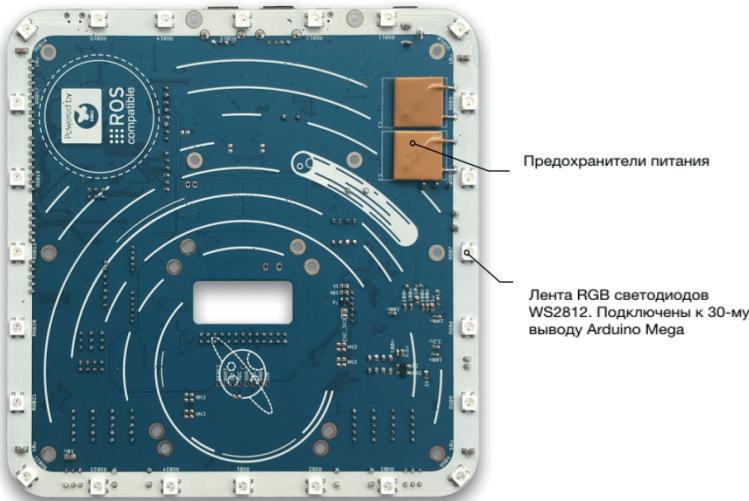


Рис. 4.1. Системная плата (вид сверху)

**Плата управления  
TurtleBro Board V1.3**



*Рис. 4.2. Системная плата (вид снизу)*

*Stop* – при нажатии сбрасывается значение *cmd\_vel* (робот останавливается), обнуляются значения одометрии и *IMU*-датчика. Удобно пользоваться этой кнопкой для установки нулевого положения робота на полигоне.

*Hw\_reset* – кнопка сброса МК *STM32*. Осуществляет полный перезапуск робота, включая *Raspberry*.

*Reset* – кнопка сброса *Arduino*. Осуществляет полный перезапуск *Arduino*.

*EMRG* – выходы для подключения чеки экстренной остановки моторов. Для работы в нормальном режиме контакты должны быть замкнуты.

Прямое взаимодействие с роботом описано в следующем разделе, из которого вы узнаете, как применять полученные знания на практике.

## 4.2. Среда разработки Arduino IDE

Для написания кода под микропроцессорную плату с платформой *Arduino* необходим набор инструментов разработчика. Существует множество инструментов для написания кода и прошивки

микропроцессорных плат, но одним из самых доступных и простых является среда разработки *Arduino IDE* – интегрированная среда разработки для *Windows*, *MacOS* и *Linux*, разработанная на *C* и *C++*, предназначенная для создания и загрузки программ на *Arduino*-совместимые платы, а также на платы других производителей.

Установка *Arduino IDE* для ОС *Linux* может быть выполнена как через терминал (существует несколько способов установки), так и с использованием файла *.deb*.

Рассмотрим один из вариантов установки *IDE* через терминал.

1. Необходимо скачать архив среды разработки на виртуальную машину с официального сайта компании *Arduino*, ссылка для скачивания: <https://www.arduino.cc/en/software>.

## Downloads

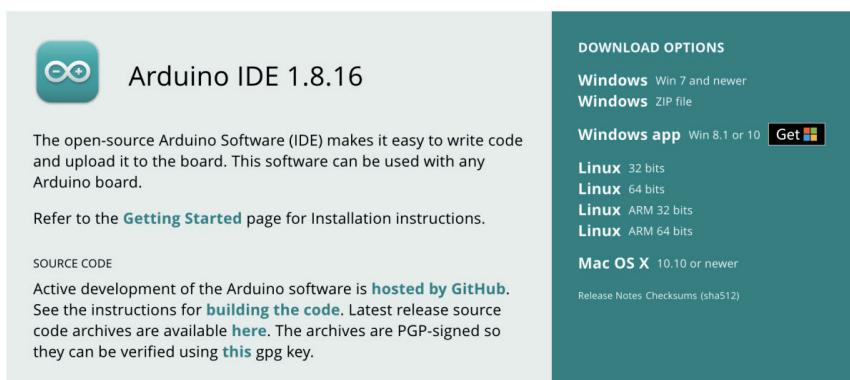


Рис. 4.3. Окно загрузки *Arduino IDE*

A terminal window titled "ros-industrial@ros-training: ~/Downloads/arduino-1.8.16". The user runs several commands to extract the tarball, change directory, and run the install script with sudo. The output shows the progress of the installation, including adding desktop shortcuts and menu items. The command "done!" indicates the process is complete.

Рис. 4.4. Окно установки *Arduino IDE*

Для ОС *Ubuntu Linux*, на которой работает виртуальная машина, необходимо выбрать пункт *Linux 64 bits*.

2. Открыть терминал *Linux*.

3. Перейти в папку *downloads*, куда по умолчанию происходит закачка файлов.

4. Распаковать файл. В терминал по умолчанию встроен архиватор *tar*, который позволяет архивировать и разархивировать файлы. Для извлечения содержимого файлов используется команда *tar -xf* название архива.

5. Перейти в распакованную папку.

6. Запустить скрипт установки *./install.sh* от суперпользователя.

Выполнив данные операции, вы увидите, что в ОС появилась среда разработки *Arduino IDE*.

#### 4.3. Описание среды разработки

Среда разработки для микроконтроллерных плат *Arduino IDE* позволяет писать код и загружать его на микроконтроллеры *Arduino/ESP8266/ESP32* и др. Окно программы представлено на рис. 4.5.

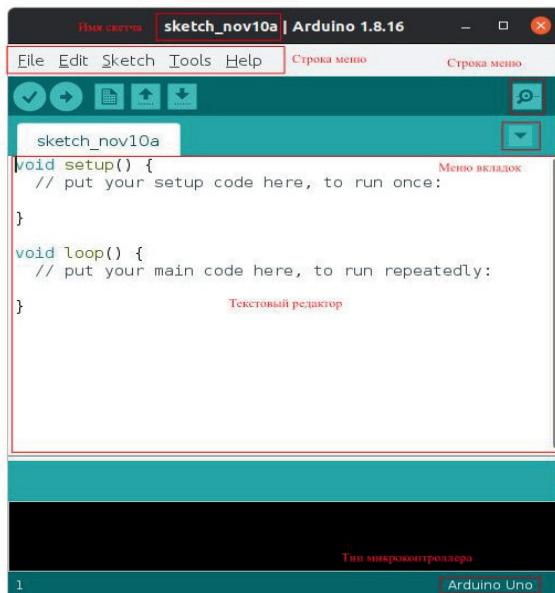


Рис. 4.5. Окно программы *Arduino IDE*



Рис. 4.6. Кнопки управления скетчом

Среда разработки позволяет вводить код программы в текстовом редакторе, обладает возможностью подсветки синтаксиса, создания и сохранения файлов, проверки кода и загрузки его на микроконтроллер. На рис. 4.6 представлены кнопки управления скетчом.

Данные кнопки позволяют ускорить работу со скетчом, а именно на панели расположены кнопки слева направо:

- 1) проверка скетча на ошибки;
- 2) загрузка скетча на плату;
- 3) создать новый скетч;
- 4) открыть скетч;
- 5) сохранить скетч.

В платформе *Arduino* программы называются скетчами. Скетч содержит в себе набор библиотек, функций, переменных и других объектов, которые доступны пользователю для работы.

Перед загрузкой скетча в плату необходимо проверить правильность выбора платы порта, к которому она подключена, в меню *tools*.

#### 4.4. Написание простой программы в Arduino IDE

При изучении какого-либо языка первой программой, которую пишет пользователь, является программа *HelloWorld*. Так как *Arduino* является платформой для взаимодействия с реальными объектами (датчиками, моторами, светодиодами), первая программа не отличается от той же программы, но направлена на управление светодиодом. Пример данной программы представлен на рис. 4.7.

Программа для *Arduino* содержит две обязательные процедуры: *void setup()* и *void loop()*. В скетче всегда должны быть данные процедуры в единственном экземпляре. Процедура *setup* выполняется один раз при запуске микроконтроллера. Обычно она используется для конфигурации портов микроконтроллера и других настроек. После выполнения *setup* запускается процедура *loop*, которая выполняется в бесконечном цикле. Именно это используется в данном примере, чтобы светодиод мигал постоянно.

The screenshot shows the Arduino IDE interface with the title bar "Blink | Arduino 1.8.16". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu is a toolbar with icons for upload, download, and other functions. A status bar at the bottom shows the number "1" and the board type "Arduino Uno". The main area displays the "Blink" sketch:

```
// the setup function runs once when you press reset c
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);      // turn the LED on
    delay(1000);                         // wait for a sec
    digitalWrite(LED_BUILTIN, LOW);       // turn the LED off
    delay(1000);                         // wait for a sec
}
```

Рис. 4.7. Программа управления светодиодом

*Arduino* использует синтаксис языка *C++*, что накладывает определенные требования к написанию кода. Первое: «{}» фигурные скобки обозначают тело оператора в языке *C++* и требуют, чтобы на каждую открытую скобку была закрыта, иначе это вызовет ошибку. Второе: в конце каждой строки по правилам языка *C++* пользователь ставит символ «;» окончания строки.

Представленный в примере код содержит три функции, наиболее распространенные в языке *Arduino*:

1) *pinMode(pin, mode)* – настройка режима работы пина. У данной функции два аргумента. На место аргумента *pin* ставиться номер пина, которым планирует управлять пользователь. В примере показано использование не номера пина, а имени, которому соответствует некий пин. По умолчанию в платформе *Arduino* *LED\_BUILTIN* значение 13, так как на платах 13-й пин соединен со светодиодом аппаратно;

2) *digitalWrite(pin, value)* – отправляет цифровой сигнал в пин;

3) *delay(value)* – функция задержки. Принимает значение в миллисекундах.

Пин – это отдельный контакт на плате *Arduino*, которым может управлять пользователь: включать и выключать пин, считывать сигнал, подключать датчики и т. п. Данное пособие не предполагает подробного ознакомления с платформой *Arduino*. Более детально изучить работу с платой *Arduino* можно по ссылке: <http://wiki.amperka.ru>.

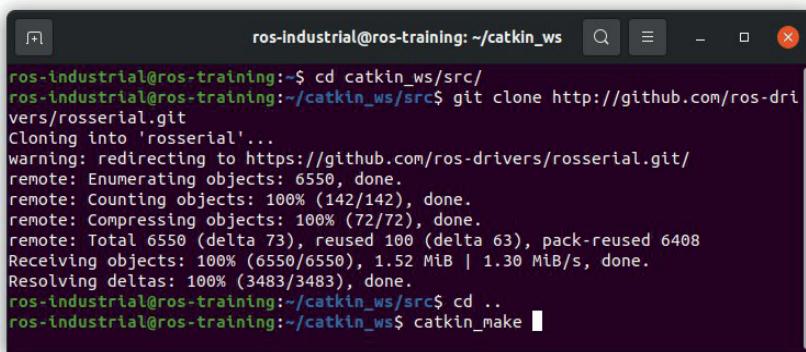
#### 4.5. *Arduino* и ROS

Взаимодействие платформы *Arduino* и *ROS* позволяет соединять реальные датчики и системы в одну общую систему. *Arduino* становится издателем или подписчиком топиков и позволяет упаковать данные от внешних компонентов в вид сообщения, которое понимает *ROS*. Микропроцессорная плата превращается в отдельный нод, который собирает, фильтрует данные, выполняет вывод данных на периферию, получает команды от других узлов.

Для реализации данного подхода для платы *Arduino* реализована библиотека *rosserial\_arduino*, которая позволяет выполнять все данные функции.

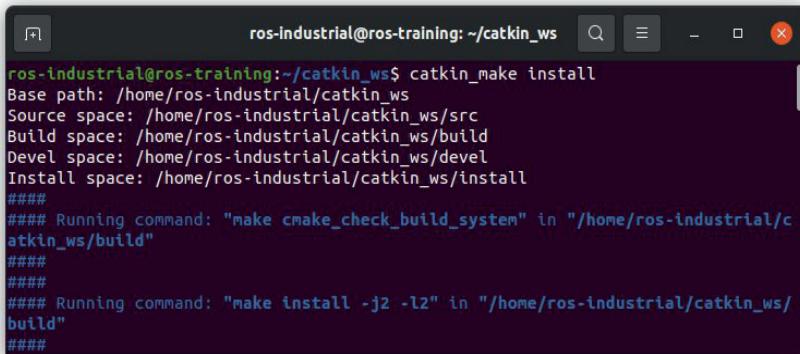
Установка библиотеки *rosserial* выполняется с помощью команд терминала, представленных на рис. 4.8.

Данный пакет лучше всего собирать из исходников, так как в них содержится наиболее новая версия. Сборка происходит по стандартному алгоритму, с использованием утилиты *catkin*. Дополнительно необходимо провести установку модулей из пакета сборки библиотек, так как по умолчанию в систему *ROS* попала информация только о протоколах связи с *Arduino*, но не сборки



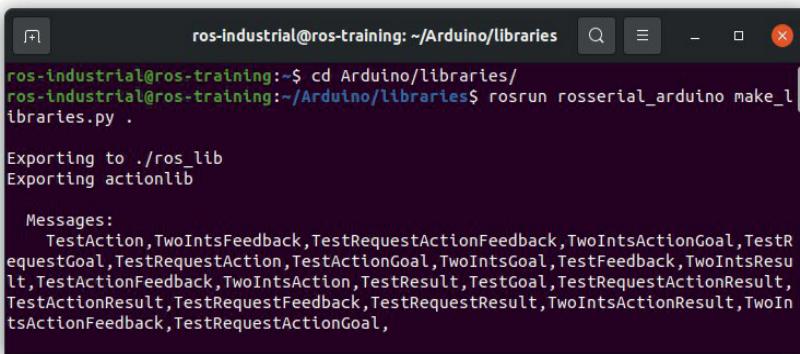
```
ros-industrial@ros-training:~/catkin_ws$ cd catkin_ws/src/
ros-industrial@ros-training:~/catkin_ws/src$ git clone http://github.com/ros-drivers/rosserial.git
Cloning into 'rosserial'...
warning: redirecting to https://github.com/ros-drivers/rosserial.git/
remote: Enumerating objects: 6550, done.
remote: Counting objects: 100% (142/142), done.
remote: Compressing objects: 100% (72/72), done.
remote: Total 6550 (delta 73), reused 100 (delta 63), pack-reused 6408
Receiving objects: 100% (6550/6550), 1.52 MiB | 1.30 MiB/s, done.
Resolving deltas: 100% (3483/3483), done.
ros-industrial@ros-training:~/catkin_ws/src$ cd ..
ros-industrial@ros-training:~/catkin_ws$ catkin_make
```

Рис. 4.8. Сборка пакета *rosserial*



```
ros-industrial@ros-training:~/catkin_ws$ catkin_make install
Base path: /home/ros-industrial/catkin_ws
Source space: /home/ros-industrial/catkin_ws/src
Build space: /home/ros-industrial/catkin_ws/build
Devel space: /home/ros-industrial/catkin_ws/devel
Install space: /home/ros-industrial/catkin_ws/install
#####
##### Running command: "make cmake_check_build_system" in "/home/ros-industrial/catkin_ws/build"
#####
#####
##### Running command: "make install -j2 -l2" in "/home/ros-industrial/catkin_ws/build"
#####
#####
```

Рис. 4.9. Установка модуля в систему



```
ros-industrial@ros-training:~/Arduino/libraries$ cd Arduino/libraries/
ros-industrial@ros-training:~/Arduino/libraries$ rosrun rosserial_arduino make_libraries.py .

Exporting to ./ros_lib
Exporting actionlib

Messages:
  TestAction,TwoIntsFeedback,TestRequestActionFeedback,TwoIntsActionGoal,TestRequestGoal,TestRequestAction,TestActionGoal,TwoIntsGoal,TestFeedback,TwoIntsResult,TestActionResult,TwoIntsAction,TestResult,TestGoal,TestRequestActionResult,TestActionResult,TestRequestFeedback,TestRequestResult,TwoIntsActionResult,TwoIntsActionFeedback,TestRequestActionGoal,
```

Рис. 4.10. Сборка библиотек для МКП Arduino

элементов компиляции библиотек для микропроцессорных плат. Установка пакета представлена на рис. 4.9.

После этого появляется возможность открыть интерфейс взаимодействия между микроконтроллерной платой и основной системой *ROS*, а также выполнить сборку библиотек.

Сборку библиотек лучше всего организовать в директории *Arduino*. В ней специально для внешних библиотек создается директория *libraries*. На рис. 4.10 представлен процесс сборки библиотек для плат *Arduino*.

Когда вы проделаете данные операции, в папке *libraries* появится папка *roslib*, содержащая библиотеки и примеры использования *ROS* и плат *Arduino*.

## 4.6. Скрипт издателя ROS

Принцип взаимодействия интерфейсов *ROS* не изменяется с появлением звена в виде МКП *Arduino*. Логика работы системы остается прежней, а следовательно, плата *Arduino* может выступать источником данных. Плата «собирает» данные от внешней периферии и упаковывает их в сообщения формата, понятного в *ROS*. Пример простого издателя, которым выступает плата *Arduino*, представлен на рис. 4.11.

Код состоит из следующих основных блоков:

1. Объявление библиотек. Как и в примере с издателем для *Python*, скетч начинается с объявления необходимых библиотек.



The screenshot shows the Arduino IDE interface with the title bar "HelloWorld | Arduino 1.8.16". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for upload, download, and search. The main code editor window contains the following C++ code:

```
#include <ros.h>
#include <std_msgs/String.h>

ros::NodeHandle nh;

std_msgs::String str_msg;
ros::Publisher chatter("/chatter", &str_msg);

char hello[13] = "hello world!";

void setup()
{
    nh.initNode();
    nh.advertise(chatter);
}

void loop()
{
    str_msg.data = hello;
    chatter.publish( &str_msg );
    nh.spinOnce();
    delay(1000);
}
```

The status bar at the bottom displays "Done compiling." and memory usage information: "Sketch uses 9086 bytes (28%) of program storage space. Global variables use 1362 bytes (66%) of dynamic memor".

Рис. 4.11. Скетч издателя для платы *Arduino*

В них прописывается логика работы с данными, а также происходит вызов классов для взаимодействия с объектами издателя.

2. Создание объекта класса для обращения к библиотеке *ros.h*. Через данный объект происходит связь с ядром *ROS*.

3. Создание вспомогательных объектов. В примере выполняется публикация сообщения типа *string*. Для того чтобы взаимодействовать с ним, объявляется объект для хранения сообщения данного типа.

4. Создание объекта издателя. Объект *chatter* является объектом, на который ссылается скетч. Так, при подключении к ядру *ROS* «видит» наличие издателя в системе. В качестве аргументов данный конструктор получает имя топика издателя и тип сообщения. В тип передается не сам объект, а адрес на область памяти, где он расположен. Это особенность написания языка *C++*.

5. Объявление дополнительных переменных. В качестве примера платы публикует сообщения *Hello world!* в топик */chatter*. Даные записаны в массив на 13 элементов.

6. Процедура настройки *void setup()*. Синтаксис для плат *Arduino* требует наличия данной процедуры, чтобы произвести настройку работы платы. В данной процедуре выполняются следующие операции:

a) инициализация нода. Инициализация выполняется один раз, поэтому данный метод вызывается в процедуре настройки;

b) инициализация издателя. Необходимо произвести инициализацию издателя, чтобы система могла публиковать сообщения.

7. Процедура *void loop()*. Аналог бесконечного цикла, который постоянно шлет данные в топик */chatter*. Данная процедура состоит из следующих частей:

a) присваивание атрибутам значений. Прежде чем отправлять значение переменной с текстом “Hello, World!”, необходимо записать его в атрибут *data*. Таким образом происходит упаковка текста в сообщение типа *string*;

b) публикация сообщения. Метод *publish* отправляет сообщение в *ROS*. После этого появляется топик */chatter*, из которого возможно вызвать текст сообщения;

c) *nh.spinOnce()*. Установка паузы между отправкой сообщений;

d) задержка работы программы, чтобы данные отправлялись не слишком часто.

Представленная программа готова для загрузки в МКП *Arduino*. Загрузка скетча выполняется с помощью кнопки «прошивка». В случае успешной загрузки в меню *Arduino IDE* появится сообщение, представленное на рис. 4.12.

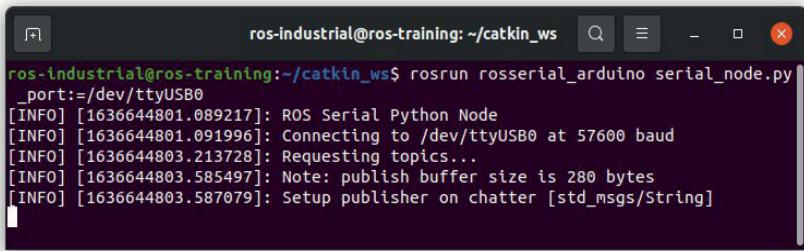


Рис. 4.12. Сообщение об успешной загрузке программы

Плата ожидает связи с *ROS*, чтобы начать слать данные. Для обеспечения передачи данных необходимо открыть канал связи между платой и ядром. Данные от микроконтроллера передаются через стандартный последовательный порт. Его реализация может иметь разные физические формы: соединение с помощью кабеля, *Bluetooth*, радиомодуль и другой физический уровень связи контроллера и системы.

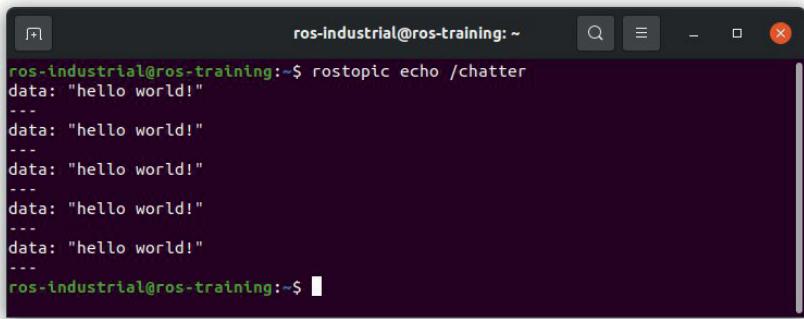
Связь и открытие канала выполняется с использованием библиотеки *rosserial\_arduino*. Запуск такого пакета представлен на рис. 4.13.

Для запуска данного узла необходимо указать пакет *rosserial\_arduino*, имя узла, порт, к которому подключена плата *Arduino*. Данный порт следует посмотреть в меню *tools* среды разработки *Arduino IDE*. Пример вывода данных представлен на рис. 4.14.



```
ros-industrial@ros-training:~/catkin_ws$ rosrun rosserial_arduino serial_node.py
/_port:=/dev/ttyUSB0
[INFO] [1636644801.089217]: ROS Serial Python Node
[INFO] [1636644801.091996]: Connecting to /dev/ttyUSB0 at 57600 baud
[INFO] [1636644803.213728]: Requesting topics...
[INFO] [1636644803.585497]: Note: publish buffer size is 280 bytes
[INFO] [1636644803.587079]: Setup publisher on chatter [std_msgs/String]
```

Рис. 4.13. Подключение к плате Arduino



```
ros-industrial@ros-training:~$ rostopic echo /chatter
data: "hello world!"
---
ros-industrial@ros-training:~$
```

Рис. 4.14. Вывод данных из топика /chatter

Данные в топик передаются раз в секунду, что не позволяет переполнить буфер данных. Следует помнить о тонкостях работы интерфейса последовательного порта и понимать, что у него есть ограничения на количество передаваемой информации. Поэтому не следует передавать данные с высокой частотой, это может привести к переполнению буфера и рассинхронизации платы *Arduino* и *ROS*.

Перед каждой новой прошивкой нужно закрывать под *rosserial\_arduino*, так как он не отпускает последовательный порт, что вызывает ошибку при новой загрузке программы в плату *Arduino*.

#### 4.7. Скрипт подписчика ROS

Подписчик, реализованный на плате *Arduino*, управляет полезной нагрузкой, подключенной к плате. Такой нагрузкой могут выступать двигатели, экраны, средства индикации, устройства управления и многое другое. В качестве примера рассмотрим

скетч, представленный на рис. 4.15, позволяющий управлять светодиодом на плате [7].

Представленный в примере скетч имеет следующие блоки кода:

1. Объявление библиотек. Так же, как и с издателем, скетч программы начинается с объявления библиотек.

2. Создание объекта класса для обращения к библиотеке *ros.h*.

Через данный объект происходит связь с ядром *ROS*.

3. Создание функции *callback*. Аналогично скрипту, написанному на языке *Python*, для обработки сообщений необходима функция *callback*, которая вызывается в случае, если данные приходят на плату. В остальное время платы *Arduino* выполняет другие задачи.



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Blink | Arduino 1.8.16
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Open, Save, Upload, and others.
- Sketch Name:** Blink
- Code Area:** Contains C++ code for a ROS node that controls an LED.

```
#include <ros.h>
#include <std_msgs/Bool.h>

ros::NodeHandle nh;

void messageCb(const std_msgs::Bool &signal_msg){
    digitalWrite(LED_BUILTIN, signal_msg.data);
}

ros::Subscriber<std_msgs::Bool> sub("on_off_led", &messageCb);

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT);
    nh.initNode();
    nh.subscribe(sub);
}

void loop()
{
    nh.spinOnce();
    delay(1);
}
```
- Status Bar:** Done uploading.
- Output Window:** Shows memory usage statistics:

```
Sketch uses 9066 bytes (29%) of program storage space. Maximum is
Global variables use 1361 bytes (66%) of dynamic memory, leaving
```
- Bottom Status:** 10 Arduino Nano, ATmega328P (Old Bootloader) on /dev/ttyUSB0

Рис. 4.15. Скетч подписчика для платы *Arduino*

4. Создание объекта подписчика. Данный объект позволяет наделить плату *Arduino* возможностью подписываться на топик и получать сообщения от *ROS* для управления полезной нагрузкой. Конструктор данного класса включает в себя тип объекта, имя топика, на который подписывается нод, адрес на область памяти функции обработчика.

5. Процедура настройки *void setup()*. В данной процедуре происходит:

а) настройка pinов. С помощью функции *pinMode* перенастраиваем pin светодиода на выход;

б) инициализация нода. Инициализация выполняется один раз, поэтому данный метод вызывается в процедуре настройки;

в) инициализация подписчика. Необходимо произвести инициализацию подписчика, чтобы система могла получать сообщения по последовательному порту.

6. Процедура *void loop()*. Аналог бесконечного цикла, который запрашивает сообщения из топика */on\_off\_led*. Данная процедура состоит из следующих частей:

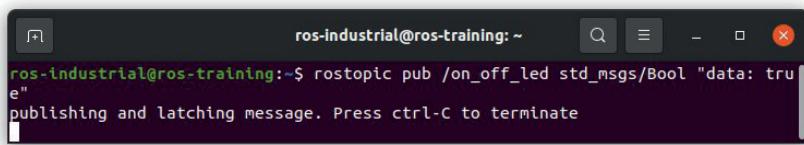
а) *nh.spinOnce()*. Установка паузы между отправкой сообщений;

б) задержка работы программы, чтобы данные отправлялись не слишком часто.

Запуск нода начинается с загрузки кода скетча на плату *Arduino* и подключения платы в терминале с использованием нода *rosserial\_arduino*.

Публикация данных выполняется утилитой *rostopic* с инструкцией *pub*, как показано на рис. 4.16.

В момент публикации сообщения типа *bool* (значений *true* или *false*) плата *Arduino* использует полученное сообщение, извлекает из него данные (поле *data*) и подставляет это значение в функцию *digitalWrite*. Данные операции позволяют управлять светодиодом, установленным на плате *Arduino*, но ничего не мешает управлять другим pinом на плате.



```
ros-industrial@ros-training:~$ rostopic pub /on_off_led std_msgs/Bool "data: true"
publishing and latching message. Press ctrl-C to terminate
```

Рис. 4.16. Публикация сообщения в топик */on\_off\_led*

#### **4.8. Практические задания**

Задание 1. Создать пакет в рабочем пространстве *catkin\_ws servo\_control* с зависимостями *rospy*.

Задание 2. Написать на языке *Python* нод *manual\_control*, публикующий сообщения угла поворота сервомашинки в топик */servo\_pub*. Угол вводится с клавиатуры и ограничен значениями от 0 до 180. Сообщения используют тип *int*.

Задание 3. Написать скетч подписчика для платы *Arduino*. Плата должна подписаться на топик */servo\_pub* и передавать полученные углы в сервомашинку, подключенную к pinu D44.

## **5. УПРАВЛЕНИЕ РОБОТИЗИРОВАННОЙ ПЛАТФОРМОЙ TURTLEBRO**

Настоящий раздел пособия посвящен прямому взаимодействию с роботизированной платформой и дает навыки и компетенции по прямой работе с ней, а также позволяет получить базовые навыки, необходимые для выполнения демонстрационного экзамена.

Большинство роботизированных платформ построено по принципу:

- 1) управляющее устройство;
- 2) системы ввода (датчики, сенсоры, кнопки и др.);
- 3) системы вывода (моторы, индикация, звуковые системы и др.).

Наш мобильный робот представляет собой разработку компании *VoltBro*, *TurtleBro* соответствует перечисленным принципам и обладает следующими системами:

1. Система питания. Обеспечивает энергоснабжение робота, контроль разряда АКБ, опционально функции заряда АКБ.

2. Приводы колес. Отвечают за корректное движение робота. Привод колеса должен правильно отрабатывать заданную скорость, выдавать одометрию. Важно отметить, что в концепции *ROS* «железо» должно хорошо работать, поэтому важно на этапе выбора или создания привода тщательно проверить его характеристики.

3. Инерциальный датчик (*IMU*). Для получения и корректировки данных о положении робота во всех современных мобильных роботах используются инерциальные датчики, такие как акселерометр и гироскоп. Часто эти два датчика реализуются на одной микросхеме. Многие производители микросхем выпускают специализированные датчики для роботов со встроенной фильтрацией данных.

4. Лазерный дальномер (*Lidar*). Одним из самых популярных на сегодняшний день датчиков для мобильного робота является врачающийся лазерный дальномер, сканирующий пространство вокруг себя. В результате его работы робот может «видеть» препятствия, строить карту, локализовывать (определять в пространстве) себя на карте. В зависимости от характеристик лазерные дальномеры могут стоить от ста до нескольких тысяч долларов.

5. Камера. Практически любой современный робот оборудован видеокамерой, ее можно использовать как просто для наблюдения, так и для реализации алгоритмов управления на основе машинного зрения. Для роботов также производится ряд специализированных

камер, позволяющих кроме изображения получать еще и карту расстояний до каждой точки (*depth*-камеры).

Именно такими устройствами обладает семейство роботов *Turtlebot*, созданное специально для изучения *ROS*. На сегодняшний день несколько производителей в мире выпускают таких роботов, в том числе есть и российский вариант – *TurtleBro*. Именно на таком роботе построена компетенция *FutureSkills 2.0* «Эксплуатация сервисных роботов».

### 5.1. Raspberry Pi

Компьютеры *Raspberry*, идущие в комплекте с роботами, поставляются с предустановленными ОС *Raspberry Pi OS Lite* (<https://www.raspberrypi.org/downloads/raspbian/>), *ROS Noetic* и всеми необходимыми системными пакетами. Обновление образа ОС возможно через скачивание и полную перезапись *SD*-карты. Для работы необходима карта размером 16 Гб. Образ можно использовать для следующих моделей *Raspberry*:

- *Raspberry 3 Model B*;
- *Raspberry 3 Model B+*;
- *Raspberry 4*.

Архив доступных образов: <https://yadi.sk/d/U0G80JoXs9eqcA>.

Рекомендуем выбирать самую последнюю версию.

Проще всего загрузить образ на *SD*-карту с помощью программы *Etcher* (<https://www.balena.io/etcher/>). Программа обладает поддержкой всех основных ОС.

По умолчанию имя робота установлено *turtlebro01*, рекомендуется сразу заменить его на имя согласно номеру платы *turtlebroNN*. Для этого необходимо отредактировать файлы */etc/hosts* и */etc/hostname*, расположенные на роботе, и переименовать *turtlebro01* → *turtlebroNN*. Удобнее всего это сделать на компьютере с ОС *Linux*, подключив *SD*-карту, или уже на включенном роботе, а потом перезагрузить его. Версию образа можно посмотреть в файле */boot/version* на *SD*-карте.

На карте установлены основные необходимые *ROS* пакеты:

```
rosinstall_generator actionlib actionlib_msgs amcl angles base_
local_planner bond bondcpp camera_calibration_parsers camera_
info_manager carrot_planner catkin class_loader clear_costmap_
recovery cmake_modules costmap_2d cpp_common cpp_package_
demo cv_bridge cv_camera diagnostic_msgs dwa_local_planner
dynamic_reconfigure fake_localization gencpp geneus genlisp genmsg
```

*genodejs genpy geometry\_msgs global\_planner gmapping image\_transport joint\_state\_publisher kdl\_parser laser\_geometry map\_msgs map\_server message\_filters message\_generation message\_runtime mk move\_base move\_base\_msgs move\_slow\_and\_clear nav\_core nav\_msgs navfn nodelet openslam\_gmapping orocos\_kdl pluginlib python\_orocos\_kdl python\_qt\_binding robot\_state\_publisher ros\_environment rosapi rosauth rosbag rosbag\_migration\_rule rosbag\_storage rosbash rosboost\_cfg rosbridge\_library rosbridge\_msgs rosbridge\_server rosbuild rosconsole rosconsole\_bridge roscpp rosCPP\_serialization rosCPP\_traits rosCPP\_tutorials roscreate rosgraph rosgraph\_msgs rosLang rosLaunch rosLib rosLint rosLisp rosLz4 rosmake rosMaster rosMsg rosNode rosOut rosPack rosParam rosPy rosPy\_tutorials rosSerial\_arduino rosSerial\_client rosSerial\_msgs rosSerial\_python rosService roSTest roSTime roSTopic roSUnit roSTwF rotate\_recovery rPLIDAR\_ros sensor\_msgs smclib std\_msgs std\_srvs stereo\_msgs tf tf2 tf2\_geometry\_msgs tf2\_kdl tf2\_msgs tf2\_py tf2\_ros tf2\_sensor\_msgs topic\_tools trajectory\_msgs urDF urDF\_parser\_plugin uvc\_camera visualization\_msgs voxel\_grid xmlrpcpp.*

Установлены основные пакеты *turtlebro turtlebro\_turtlebro\_navigation*.

## 5.2. Настройка подключения к новой сети Wi-Fi через SD-карту

На SD-карте, содержащей готовый образ системы для запуска на роботе, есть два раздела разного размера. Обычно они называются *system* и *boot*, но иногда система может назвать их по-другому при подключении к компьютеру. Обычно при подключении SD-карты к компьютеру с *Windows* он покажет, что есть два *usb*-накопителя. Один из них можно будет открыть – это *boot*. Второй накопитель компьютер предложит отформатировать – это *root*.

Раздел *system* содержит стандартный набор директорий файловой системы *Linux* и занимает основной объем SD-карты (подробнее по ссылке: <https://ru.wikipedia.org/wiki/FHS>).

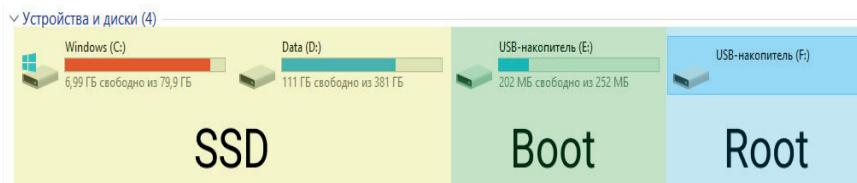
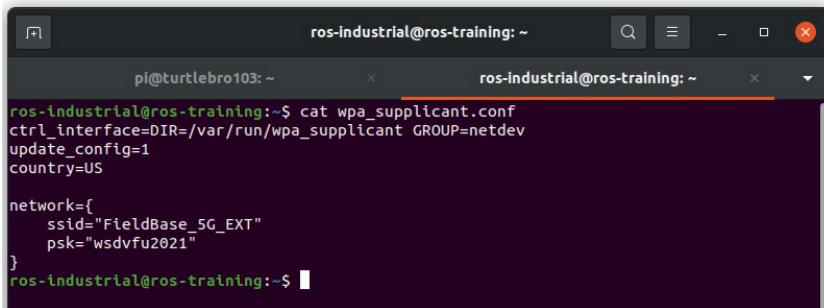


Рис. 5.1. Файловая система



```
pi@turtlebro103: ~          ros-industrial@ros-training: ~
ros-industrial@ros-training:~$ cat wpa_supplicant.conf
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
    ssid="FieldBase_5G_EXT"
    psk="wsdvvfu2021"
}
ros-industrial@ros-training:~$
```

Рис. 5.2. Общий вид файла конфигурации

Раздел *boot* небольшой и содержит настройки запуска *Raspberry* (подробнее по ссылке: [https://www.raspberrypi.org/documentation/configuration/boot\\_folder.md](https://www.raspberrypi.org/documentation/configuration/boot_folder.md)).

Если на этапе загрузки *Raspberry* найдет файл *wpa\_supplicant.conf* в разделе *boot*, то этот файл будет перемещен в */etc/wpa\_supplicant/wpa\_supplicant.conf* и таким образом станет конфигурационным файлом подключения к Wi-Fi-сетям.

Чтобы сконфигурировать *Raspberry* в этом режиме, подключите *SD*-карту *Raspberry* к вашему компьютеру. Создайте на *SD*-карте файл */boot/wpa\_supplicant.conf* с содержанием, представленным на рис. 5.2.

Использование символа кавычек «» в файле с настройками обязательно.

Удостовериться в правильной настройке сетевой конфигурации можно, зайдя на веб-сервер, запущенный на роботе: <http://192.168.0.100:8080> (указав *IP*-адрес вашего робота). На этой странице будут доступны основные данные робота и изображение, получаемое с камеры. Роботом можно управлять кнопками *A*, *W*, *S*, *D*.

### 5.3. Ввод робота в эксплуатацию

Начиная с этого пункта и до конца раздела будут приводиться методические указания по выполнению заданий демонстрационного экзамена.

#### 5.3.1. Проверка базовой информации робота

1. Продемонстрировать название дистрибутива *Linux*; команда: *lsb\_release -a* пункт *Distributor ID*.

2. Продемонстрировать версию интерпретатора *Python*; *python -version* и *python3 -version*.
3. Продемонстрировать версию библиотеки *rospy*; *pip3 freeze | grep rospy*.
4. Продемонстрировать версию дистрибутива *ROS*; *rosversion -d*.
5. Продемонстрировать версию пакета *turtlebro*; *rosversion turtlebro*.
6. Продемонстрировать версию прошивки микроконтроллера материнской платы; *rosservice call /board\_info «request: {}» → firmware\_version*.
7. Продемонстрировать серийный номер системной платы робота (*mcu\_id*); *rosservice call /board\_info «request: {}» → mcu\_id*.
8. Продемонстрировать размер оперативной памяти (*Kb*); *grep MemTotal /proc/meminfo*.
9. Продемонстрировать полное пространство на *SD*-карте (*Gb*); *df -h | grep /dev/root* → первое значение.
10. Продемонстрировать версию образа ОС, установленной на *Raspberry Pi*. *uname -r*.

### 5.3.2. Проверка ROS-процессов

Необходимо проверить наличие топиков и сервисов, указанных в инструкции, в списках всех топиков и сервисов, запущенных на роботе. После проверки сделать отметку в сервисной книжке. Список топиков и сервисов можно найти в документации. Пакет *turtlebro* → параметры и настройка через *launch*.

Команда для вывода всех топиков – *rosservice list*.

Сверьте вручную данные из документации и вывод команд. Если все верно, то не забудьте внести эти данные в сервисную книжку.

### 5.3.3. Проверка работы камеры

Необходимо проверить наличие камеры, установить разрешение и продемонстрировать изменения, показать трансляцию в веб-интерфейсе.

Так как в *Linux* все является файлом, то и подключенные устройства можно найти в одном из каталогов. Таким каталогом является */dev. DEV* – каталог в системах типа *UNIX*, содержащий так называемые *специальные файлы* – интерфейсы работы с драйверами ядра. Если воспользоваться командой *ls -la* и вывести всю информацию о файлах, которые располагаются в этой папке, то будет видно, что большинство из них имеют тип *c (character)* или *b (block) device*.

```
pi@turtlebro103: ~
adata are not supported on this platform. In the latter case, you should configu
re Tepi with --disable-gvfs-metadata.
ros-industrial@ubuntu:~$ ssh pi@192.168.31.42
pi@192.168.31.42's password:
Linux turtlebro103 5.10.52-v7l+ #1441 SMP Tue Aug 3 18:11:56 BST 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Nov  8 04:17:50 2021 from 192.168.31.216
pi@turtlebro103:~$ ls -la /dev/video*
crw-rw---- 1 root video 81, 7 Nov  8 07:17 /dev/video0
crw-rw---- 1 root video 81, 8 Nov  8 07:17 /dev/video1
crw-rw---- 1 root video 81, 1 Nov  8 07:17 /dev/video10
crw-rw---- 1 root video 81, 5 Nov  8 07:17 /dev/video11
crw-rw---- 1 root video 81, 6 Nov  8 07:17 /dev/video12
crw-rw---- 1 root video 81, 0 Nov  8 07:17 /dev/video13
crw-rw---- 1 root video 81, 2 Nov  8 07:17 /dev/video14
crw-rw---- 1 root video 81, 3 Nov  8 07:17 /dev/video15
crw-rw---- 1 root video 81, 4 Nov  8 07:17 /dev/video16
```

Рис. 5.3. Список файлов

```
pi@turtlebro103: ~
$ v4l2-ctl --list-dev
bcm2835-codec-decode (platform:bcm2835-codec):
/dev/video10
/dev/video11
/dev/video12

bcm2835-isp (platform:bcm2835-isp):
/dev/video13
/dev/video14
/dev/video15
/dev/video16

USB 2.0 PC Camera: PC Camera (usb-0000:01:00.0-1.2):
/dev/video0
/dev/video1
```

Рис. 5.4. Список камер

Чтобы найти все подключенные камеры, воспользуйтесь командой `ls -la /dev/video*`. После ее выполнения в окне терминала появится список из девяти файлов, далее необходимо определить, какой файл соответствует физической камере робота.

Команда `v4l2-ctl -list-devices` выведет информацию о том, к каким интерфейсам подключены камеры, что из этого виртуальная камера или интерфейс, а что – физическая.

Теперь отчетливо видно, что только две камеры из девяти являются настоящими и подключены по *USB*, но на роботе установлена только одна. Далее необходимо определить, какая из них реальная,

## Мета данные камеры

## Данные о картинке

```
pi@turtlebro103:~ $ sudo v4l2-ctl --device=/dev/video0 --all
Driver Info:
  Driver name      : uvcvideo
  Card type        : USB 2.0 PC Camera: PC Camera
  Bus info         : usb-0000:01:00.0-1.2
  Driver version   : 5.10.52
  Capabilities    : 0x84a00001
    Video Capture
    Metadata Capture
    Streaming
    Extended Pix Format
    Device Capabilities
  Device Caps     : 0x04200001
    Video Capture
    Streaming
    Extended Pix Format
Media Driver Info:
  Driver name      : uvcvideo
  Model            : USB 2.0 PC Camera: PC Camera
  Serial           :
  Bus info         : usb-0000:01:00.0-1.2
  Media version    : 5.10.52
  Hardware revision: 0x00000004 (4)
  Driver version   : 5.10.52
Interface Info:
  ID               : 0x03000002
  Type             : V4L Video
Entity Info:
  ID               : 0x00000001 (1)
  Name             : USB 2.0 PC Camera: PC Camera
  Function         : V4L2 I/O
  Flags            : default
  Pad 0x01000007  : 0: Sink
  Link 0x02000010: from remote pad 0x10000a of entity 'Processing 2': Data, Enabled, Immutable
Priority: 2
Video input : 0 (Camera 1: ok)
Format Video (Continues):
  Width/Height     : 640/480
  Pixel Format    : 'I420' (Motion-JPEG)
  Field            : None
  Bytes per Line  : 0
  Size Image       : 921600
  Colorspace       : sRGB
  Transfer Function: Rec. 709
  YCbCr/HSV Encoding: ITU-R 601
  Quantization    : Default (maps to Full Range)
  Flags            :
Crop Capability Video Capture:
  Bounds           : Left 0, Top 0, Width 640, Height 480
  Default          : Left 0, Top 0, Width 640, Height 480
  Pixel Aspect     : 1/1
Selection: crop_default, Left 0, Top 0, Width 640, Height 480, Flags:
Selection: crop_bounds, Left 0, Top 0, Width 640, Height 480, Flags:
Streaming Parameters Video Capture:
  Capabilities     : timperframe
  Frames per second: 10.000 (10/1)
  Read buffers     : 0
    brightness 0x00980900 (int)   : min=-255 max=255 step=1 default=0 value=20
    contrast 0x00980901 (int)    : min=0 max=30 step=1 default=16 value=16
    saturation 0x00980902 (int)  : min=0 max=127 step=1 default=36 value=60
    hue 0x00980903 (int)        : min=-16000 max=16000 step=100 default=0 value=0
white_balance_temperature_auto 0x00980904 (bool)   : default=1 value=1
  gamma 0x00980910 (int)       : min=20 max=250 step=1 default=100 value=100
  power_line_frequency 0x00980918 (menu)   : min=0 max=2 default=1 value=1
  white_balance_temperature 0x00980919 (int)   : min=2500 max=7000 step=1 default=5000 value=5000 flags=inactive
  sharpness 0x0098091b (int)    : min=0 max=15 step=1 default=2 value=2
  backlight_compensation 0x0098091c (int)   : min=0 max=2 step=1 default=1 value=1
```

Рис. 5.5. Данные камеры

информацию экспериментально: отсоедините *usb*-кабель камеры от микрокомпьютера и запустите команду *ls -la /dev/video\**.

*Video0* и *video1* исчезли, значит, эти файлы отвечают за реальную камеру. Присоедините обратно *usb*-кабель камеры к компьютеру и перезагрузите робота командой *sudo reboot*. Запустите еще раз команду вывода файлов, камера должна будет появиться в списке.

Теперь необходимо установить максимальное разрешение камеры в сервисе *set\_camera\_info* и продемонстрировать, что новые значения корректно установлены. В работе есть два узла, которые используют камеру */webserver* и */uvc\_camera*, из-за этого они не могут быть запущены вместе, обрабатывать видеопоток может только один из них. Изначально всегда включен */webserver*. Это можно проверить, введя в браузере вот этот адрес: <http://turtlebro103.local:8080/>, далее откроется веб-приложение с видео с камеры. Невозможно получить какие-либо данные из этого узла, кроме самой картинки, поэтому запустим второй узел, чтобы получить данные в чистом виде.

Для этого необходимо открыть файл *turtlebro.launch*. В нем записано, какие узлы нужно роботу запускать при старте.

Здесь есть две важные записи: *run\_turtlebro\_web* и *run\_camera\_ros*. Первая запускает */webserver*, вторая – */uvc\_camera*. Необходимо открыть этот файл текстовым редактором и заменить *true* на *false*, *false* на *true* в этих строчках. После чего перезагружаем робота.

```
Last login: Tue Nov  9 09:28:27 2021 from 192.168.31.161
pi@turtlebro103:~ $ ls -la /dev/video*
crw-rw---- 1 root video 81, 0 Nov  9 09:17 /dev/video0
crw-rw---- 1 root video 81, 1 Nov  9 09:17 /dev/video1
crw-rw---- 1 root video 81, 3 Nov  9 09:17 /dev/video10
crw-rw---- 1 root video 81, 7 Nov  9 09:17 /dev/video11
crw-rw---- 1 root video 81, 8 Nov  9 09:17 /dev/video12
crw-rw---- 1 root video 81, 2 Nov  9 09:17 /dev/video13
crw-rw---- 1 root video 81, 4 Nov  9 09:17 /dev/video14
crw-rw---- 1 root video 81, 5 Nov  9 09:17 /dev/video15
crw-rw---- 1 root video 81, 6 Nov  9 09:17 /dev/video16
pi@turtlebro103:~ $ ls -la /dev/video*
crw-rw---- 1 root video 81, 3 Nov  9 09:17 /dev/video10
crw-rw---- 1 root video 81, 7 Nov  9 09:17 /dev/video11
crw-rw---- 1 root video 81, 8 Nov  9 09:17 /dev/video12
crw-rw---- 1 root video 81, 2 Nov  9 09:17 /dev/video13
crw-rw---- 1 root video 81, 4 Nov  9 09:17 /dev/video14
crw-rw---- 1 root video 81, 5 Nov  9 09:17 /dev/video15
crw-rw---- 1 root video 81, 6 Nov  9 09:17 /dev/video16
pi@turtlebro103:~ $
```

Рис. 5.6. Список камер

```
pi@turtlebro103:/etc/ros/turtlebro.d $ cat turtlebro.launch
<launch>

<arg name="run_rosserial" default="true"/>
<arg name="run_ydlidar" default="false"/>
<arg name="run_rplidar" default="true"/>

<arg name="run_turtlebro_web" default="true"/>
<arg name="run_camera_ros" default="false"/>

<arg name="run_simple_odom" default="true"/>

<include file="$(find turtlebro)/launch/rosserial.launch" if="$(arg run_rosserial)"/>
<include file="$(find turtlebro)/launch/robot_model.launch"/>

<include file="$(find turtlebro)/launch/camera_ros.launch" if="$(arg run_camera_ros)"/>

<include file="$(find turtlebro)/launch/ydlidar.launch" if="$(arg run_ydlidar)"/>
<include file="$(find turtlebro)/launch/rplidar.launch" if="$(arg run_rplidar)"/>

<include file="$(find turtlebro_web)/launch/turtlebro_web.launch" if="$(arg run_turtlebro_web)"/>
<include file="$(find turtlebro)/launch/simple_odom.launch" if="$(arg run_simple_odom)"/>

</launch>
pi@turtlebro103:/etc/ros/turtlebro.d $
```

Рис. 5.7. Содержимое *turtlebro.launch*

Теперь при переходе по ссылке не будет открываться веб-страница, значит, все сделано верно. В списке топиков должны появиться */camera\_info* и *front\_camera/compressed*. Первый отправляет данные о камере, второй – о картинке и сам кадр изображения. Запустим первый командой *rostopic echo /camera\_info*. Возвращаются данные о камере. Если приходят нулевые значения, значит, не сработала программа установки значений, необходимо сделать это вручную.

Ранее запускалась команда для определения, какая камера передает данные, она как раз и вернула данные о картинке. Отсюда берутся данные о разрешении, они нужны по заданию.

Существует сервис */set\_camera\_info*, который настраивает данные о картинке. Введем команду *rosservice call /set\_camera\_info* и нажмем *Tab*. В результате появятся данные, которые сервис ожидает как входные. Необходимо стрелкой влево дойти до пунктов *"height"* и *"width"* и заменить на соответствующие значения. После чего можно опять вызвать команду вывода данных из топика */camera\_info*, чтобы увидеть изменения. Также необходимо показать, что данные поступают, для этого нужно вызвать топик */front\_camera/compressed* командой *rostopic echo /front\_camera/compressed*.

По заданию нужно продемонстрировать работу камеры в веб-интерфейсе. Для этого нужно повторить действия по замене переменных в файле *turtlebro.launch*.

### 5.3.4. Проверка одометрии колес

Необходимо проверить корректность работы одометрии.

В данном подразделе будет рассмотрен случай, когда одометрия работает корректно и настраивать базу робота не нужно. Но может возникнуть ситуация, когда робот не едет по прямой, а движется по дуге, тогда необходимо искать проблему и устранять ее. Возможные причины: на роботе стоят разные двигатели, редуктор одного из двигателей повредился. В этих случаях следует уведомить судью о поломке и попросить заменить робота или двигатели. Проверить это можно, если одно из колес проскальзывает или если маркировки на двигателях разнятся. Еще одна причина – неверные настройки одометрии. Для этого необходимо посмотреть данные одометрии. Требуется изменить конфигурационные данные робота: расстояния между колесами, параметры колес.

Основная задача – проверить, корректно ли робот принимает и исполняет команды, связанные с перемещением. Если ему отправлена команда двигаться вперед со скоростью  $V$ , выполняет ли он ее правильно. Есть несколько способов проверить это: воспользуемся ручным вводом данных напрямую в топик. Передача скорости напрямую в топик движения позволит точнее всего отследить корректность настройки платформы.

Сперва запустите топик с одометрией, чтобы определить, где сейчас по данным информационно-управляющей системы он располагается *rostopic echo /odom*. Если вы включили робота и еще никуда не перемещались, то все должно быть по 0. Нулевые значения гарантируют в будущем высокую точность навигации, что поможет точнее строить карты и перемещаться по ним. Если вам вернулись не нулевые значения в позиции робота, то необходимо обнулить одометрию. Для этого запустите сервис *rosservice call /reset*. Одометрия обнулится, это можно будет проверить, запустив вывод сообщений из топика */odom*.

Для следующих тестов рекомендуется разместить робота на подставке, чтобы колеса не касались пола. Также следует открыть второе окно терминала, чтобы данные одометрии постоянно были на экране.

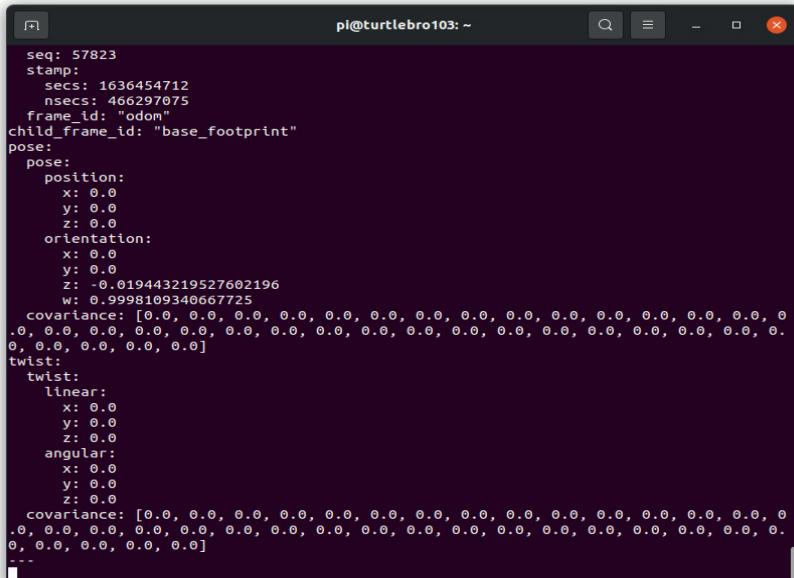
Существует топик */cmd\_vel*, который позволяет напрямую передавать скорость движения робота, а также скорость вращения. Топик принимает на вход шесть значений: линейная скорость в м/с по трем осям и угловая скорость по трем осям. Наш робот имеет только одну пару колес, поэтому перемещаться может только по од-

ной оси  $X$ . Также он может поворачиваться только по одной оси  $Z$ . Вы можете проверить это поочередно, отправив данные на каждую ось. Данный подход будет работать только при наличии данных на оси  $X$  для *linear* и оси  $Y$  для *angular*.

Теперь можно эмпирически узнать максимальную линейную скорость робота и максимальную угловую скорость. Найдем ее экспериментально, запустив топик с одометрией, а также топик *rostopic pub /cmd\_vel geometry\_msgs/Twist*. После ввода команды нажмите *Tab*, чтобы появился шаблон входных данных. Введите для *linear X* значение 0.1 и меняйте его с шагом 0.1.

Значение скорости доходит до 0.25 м/с и более не возрастает. Это предельная скорость робота при движении вперед. Повторим для отрицательных значений, чтобы узнать максимальную скорость перемещения назад.

Скорость робота назад тоже не более 0.25 м/с. Для того чтобы робот начал вращение, необходимо вводить значения в поле *angular Z*. Вращение по часовой стрелке – отрицательные значения, против – положительные. Максимальная угловая скорость для робота – 2.5 рад/с. Теперь проверим, верно ли робот исполняет команды.

A screenshot of a terminal window titled "pi@turtlebro103: ~". The window displays a JSON-like structure representing odometry data. The data includes fields for sequence number (seq), stamp (secs and nsecs), frame\_id ("odom"), child\_frame\_id ("base\_footprint"), pose (position and orientation), covariance, and twist (linear and angular velocities). The terminal interface shows standard Linux window controls at the top.

```
seq: 57823
stamp:
  secs: 1636454712
  nsecs: 466297075
frame_id: "odom"
child_frame_id: "base_footprint"
pose:
  pose:
    position:
      x: 0.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.019443219527602196
      w: 0.9998109349667725
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0, 0.0]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0, 0.0]
```

Рис. 5.8. Вывод данных одометрии в терминал

Сбросим одометрию и отправим роботу команду данными линейной скорости 0.1 м/с, в топике с одометрией должны расти значения в строке *position X* и только в ней, через 5 с строка должна показывать 1 м. Затем установите скорость -0.1 м/с, данные должны вернуться в 0. Если данные отображаются корректно, то поставьте робота на поле и повторите тест.

Не забудьте внести эти данные в сервисную книжку.

### **5.3.5. Проверка контроля напряжения аккумуляторной батареи**

Для отображения состояния аккумуляторной батареи существует топик, вызываемый командой *rostopic echo /bat*. Он выводит информацию о напряжении батареи. С помощью добавления программы *grep* можно выводить только информацию о напряжении *rostopic echo /bat | grep voltage*. Также информация об аккумуляторе выводится на индикаторе на материнской плате робота. Проверить состояние каждой отдельно взятой ячейки можно с помощью мультиметра.

Не забудьте внести эти данные в сервисную книжку.

### **5.3.6. Проверка лидара**

Лидар – технология измерения расстояний путем излучения света и замера времени возвращения этого отраженного света на ресивер. На роботе лидар расположен сверху, он постоянно вращается, чтобы обеспечить покрытие 360 градусов. Для работы с ним существует узел */rplidarNode* и созданный им топик */scan*. Топик возвращает массив значений – расстояние до объекта. Всего в массиве 360 значений, следовательно, элемент хранит расстояние до объекта на *i* градусе. Но считывать такие данные неудобно, поэтому существует программа для визуализации – *Rviz*. Она позволяет визуализировать в графическом интерфейсе переданные в топики данные. С ее помощью можно отобразить местоположение робота и управлять им.

*Rviz* не может быть запущен на компьютере без визуальной ОС, также он довольно требователен к аппаратному обеспечению, поэтому запускается не на роботе, а на стороннем компьютере с установленным *ROS*. Для этого необходимо прописать в компьютере переменные среды, которые будут хранить IP-адрес робота.

Необходимо открыть файл *~/.bashrc*. В нем прописаны команды, которые запускаются сразу после открытия терминала. В самом низу файла экспортовать две переменные:

```
export ROS_MASTER_URI=http://192.168.31.250:11311/
export ROS_HOSTNAME=192.168.31.100
```

Где 192.168.31.250 – это *IP* робота; 192.168.31.100 – это *IP* вашего компьютера. После ввода данных и сохранения изменений в терминале нужно прописать следующую команду: *source ~/.bashrc*. После этого изменения будут применены и можно будет запустить *Rviz*. Но об этом чуть позже.

Сперва необходимо проверить, что топик существует и данные с него поступают. Запустим топик */scan* и проверим, что данные приходят и меняются: *rostopic echo /scan*. Поднесите руку к лазерному дальномеру и проверьте, изменились ли выходные данные. Если все работает корректно, то необходимо на отдельном компьютере запустить *Rviz*, чтобы визуализировать выходные данные. *Rviz* – это узел, поэтому для его запуска необходимо прописать команду на компьютере: *rosrun rviz rviz*. После ввода команды откроется окно программы. Разберемся, что в ней есть, какие блоки за что отвечают.

Поле визуализации все данные, которые будут подгружены в *Rviz*, будут отображаться на нем. Они могут быть разного типа: модель робота, область точек, вектор. По полю можно перемещаться с помощью камеры, также через него будут отправляться команды роботу: выставление точек по перемещению, выставление вектора движения, смена одометрии.

В список отображаемых данных можно добавить данные из любого топика. Если до этого только в командной строке вызывался топик и было видно, как большое количество данных «пролетает по терминалу», то сейчас эти данные можно будет визуализировать. Вспомним задачу по проверке камеры, можно добавить сюда топик */front\_camera/compressed*, увидеть картинку с камеры. Чтобы добавить данные, нужно нажать на кнопку *add* и выбрать топик или шаблон для визуализации. Чтобы его удалить, нужно выбрать пункт и нажать *remove*. При этом каждый элемент можно скрывать, снимая галочку с соответствующего чекбокса напротив названия.

Вы можете изменять каждый элемент, для этого нужно развернуть его, нажав на имя, и поменять нужные данные.

Наконец, панель инструментов. Там расположены инструменты, которыми можно пользоваться на поле: измерение расстояния, смена ракурса и т. д. Они понадобятся во время построения карты полигона и перемещения робота по ней. Разобравшись с тем, как устроен *Rviz*, вернемся к задаче, как отобразить данные с лидара.

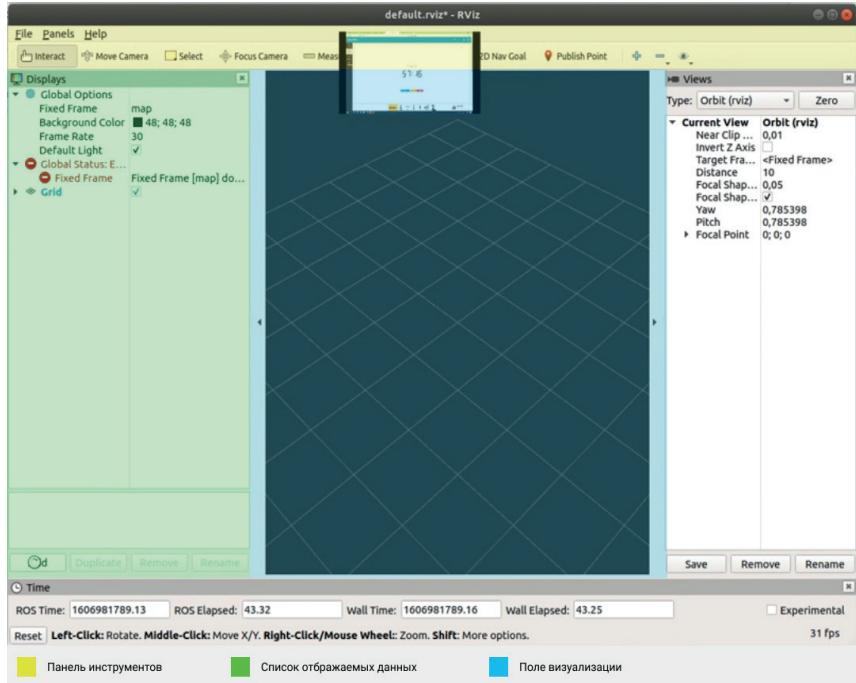


Рис. 5.9. Окно Rviz

Добавим в список данных новый топик: `add → by topic → /scan → LaserScan`. Теперь в списке должен появиться `LaserScan`. При этом данные не будут отображаться, возможно, сам элемент списка будет подсвечен красным. Так произошло, потому что в глобальных настройках указано, что данные поля привязываются к `map`. `LaserScan` никак с ним не связан, поэтому происходят ошибки. Поменяем конфигурацию, настроим отображение относительно робота: `Displays → Global Options → Fixed Frame → base_link`. На карте должна появиться область красных точек. Проверьте, что все верно, поднесите к лидару руку. Картинка должна поменяться.

### 5.3.7. Проверка IMU-датчика

*IMU* – это гироскопическое устройство, предназначенное для стабилизации отдельных предметов или приборов, а также для определения угловых отклонений предметов.

Для получения данных с этого датчика нужно вызвать топик */imu* командой *rostopic echo /imu*, в результате будут отображаться данные типа *sensor\_msgs/Imu*. Проверить корректность этих данных довольно сложно, но возможно проверить, что это не шумы и существует зависимость положения или перемещения робота и отображения данных. Откройте веб-интерфейс робота и терминал с запущенным топиком. Попробуйте переместить робота вперед или начать вращение, соответствующие данные должны меняться.

Также необходимо проверить, что частота обновления данных соответствует документации. Для этого нет прямого способа, необходимо вручную посчитать, сколько различных данных было отправлено за 1 с. Можно запустить топик, оставить на пару секунд, остановить его и вручную посчитать, какое количество данных содержит в строке *secs* одно и то же значение. Этот процесс возможно автоматизировать, введя последовательно команды:

```
rostopic echo /imu > imu_output – отключите командой Ctrl + C через пару секунд
```

```
export curr_second="secs: second" – вместо second значение secs из файла imu_output
```

```
grep imu_output | wc -l
```

В результате будет выведено число – количество данных, содержащих в качестве параметра *secs* указанное выше время. Согласно документации, у вас должно получиться 20. После того как вы удостоверились, что данные приходят корректно, нужно их визуализировать. В связи с тем, что *Rviz* по умолчанию некорректно отображает данные *IMU*, необходимо поставить соответствующий плагин для корректного отображения данных с *IMU*. Установка плагина осуществляется командой *sudo apt install ros-noetic-rviz-imu-plugin*.

При добавлении визуализации данных в *Rviz* необходимо выбрать тип сообщения *rviz\_imu\_plugin* → *Imu*, а также топик */imu*.

### 5.3.8. Проверка микроконтроллера *Atmega*

Необходимо проверить работу микроконтроллера *Atmega* и его коммуникацию с *ROS*, проверить работоспособность светодиодной ленты и кнопок *D22-D25*.

Загрузить код в *Arduino* можно двумя способами: через компьютер и среду *IDE* или через *Raspberry Pi*. Первый способ понятен и прост, если вы уже работали с *Arduino*, поэтому обсудим второй. Для примера будем использовать базовый скетч *blink* – он включает и выключает встроенный светодиод.

Откройте в среде файл, проверьте, что он успешно компилируется: *CTRL + R* или Скетч → Проверить / Компилировать. Если среда не нашла никаких ошибок, то действуем дальше. Нужно преобразовать код в бинарный файл *CTRL + ALT + S* или Скетч → Экспорт бинарного файла. Не забудьте указать, на какую плату вы собираетесь загружать файл. После чего необходимо отправить этот файл на робота. Воспользуйтесь возможностью файлового менеджера *Linux* показывать файловую систему удаленных машин. После воспользуйтесь командой: *avrdude -v -v -p atmega2560 -c wiring -P /dev/serial/by-id/usb-Silicon\_Labs\_CP2102\_USB\_to\_UART\_Bridge\_Controller\_0001-if00-port0 -b 115200 -D -U flash:w:sketch\_mar24a.ino.mega.hex:i*

Где после *flash:w:* идет название файла. Если консоль вывела команду об успешной загрузке, то все готово. Теперь необходимо загрузить тестовую прошивку и проверить, что все работает корректно.

Идея *ROS* – микросервисность. Это можно было увидеть при настройке камеры и при проверке каждого модуля. Вместо того чтобы запускать одну большую программу и там искать блок кода, который отвечает за модуль, производится вызов нужного топика с данными. Когда меняется способ работы с данными, меняется файл запуска, после перезагрузки видно, что появляются другие узлы. Это связано с тем, что каждый модуль является полноценной, а главное, независимой программой, которая имеет для связи интерфейс взаимодействия – сервисы и узлы.

Далее это будет отчетливо видно. В коде будут вызываться топики и сервисы, а не импортироваться программы. Поэтому для того, чтобы следовать принципам *ROS*, нужно добавить в код *Arduino* интерфейс взаимодействия. Программа будет узлом, который выполняет свою работу или постоянно, или после получения команды. Если открыть тестовый код, то сразу будут видны эти принципы и знакомые слова, а после загрузки скетча в *nodelist* и *topiclist* появятся элементы с именами из кода. Для выполнения задания их следует запустить, чтобы удостовериться в корректности работы платы.

## 5.4. Настройка и запуск робота

### 5.4.1. Автономная навигация

Для того чтобы робот мог перемещаться самостоятельно, без передачи параметров человеком, как это делалось ранее, необходимо написать программу, которая будет манипулировать топиками.

Так как используется образовательный робот, то такой код уже написан, необходимо загрузить его на робота, правильно настроить и запустить.

Перейдите в папку `~/catkin_ws/src` и запустите следующие две команды: `git clone https://github.com/voltbro/turtlebro_navigation` `git export ROVER_MODEL=turtlebro`.

Последнюю команду рекомендовано записать в файл `~/bashrc`. Скачаем код с навигацией, чтобы прочитать код, перейдите по ссылке. Там же находится код-документация по настройке. Если код успешно загрузился, то сбрасываем одометрию и запускаем `Rviz`.

Добавьте на поле визуализации *LaserScan* модель робота. Запустите узел автономной навигации в папке, куда вы ее скачали, командой `roslaunch turtlebro_navigation turtlebro_slam_navigation.launch open_rviz:=0`.

На этом этапе могут возникнуть проблемы, может не произойти загрузка карты. В терминале с запущенным узлом будет «висеть» строка *requesting the map*. В этом случае стоит перезагрузить робота. В *Rviz* добавляем последний элемент – *Map*, установите в поле *Topic* значение */map*. Теперь нам нужно построить карту. Можно пользоваться веб-версией для этого, но мы рекомендуем инструмент в *Rviz 2D Nav Goal*. С помощью этого инструмента достаточно указывать точку, куда должен приехать робот и в какую сторону он будет развернут после окончания движения. Для этого нужна очень точная карта, так как после через растровый редактор будет производиться выделение границы карты вручную.

После того, как карта достроена, не завершая процесс навигации, в отдельном окне запустите команду `rosrun map_server map_saver -f тутар`. Карта сохранилась в файл *тутар*. Карта лежит в директории, в которой была запущена команда. На самом деле сохранилось два файла: *тутар.pgm* и *тутар.pgtx*. Первый хранит карту, а второй – метаданные. Скачайте эти файлы на свой компьютер, теперь необходимо их изменить. Завершайте процесс навигации. Откройте карту в любом растровом редакторе и уберите все «артефакты» и «выбросы». Карта содержит три цвета: черный, белый, серый. Черный – стенка, белый – разрешенная для перемещения поверхность, серая – неизвестная область. Необходимо закрыть все стенки, удалить внутри стен неизвестные области, оставив только белый цвет. Вне стен должен быть только серый цвет.

На заднем плане представлена исходная карта, на переднем плане – переработанная. В качестве примера исходная карта была

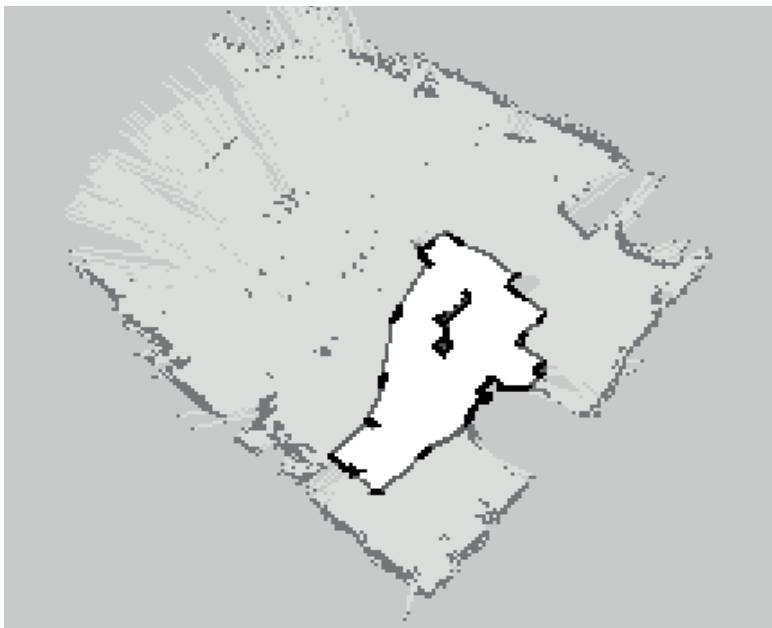


Рис. 5.10. Карта полигона

ограничена вручную и выделена отдельная область. Теперь робот не будет пересекать границы мнимой карты. Также поменяем название карты на *map*, поэтому нужно поменять название второго файла, в нем же нужно заменить строчку *image: mymap.pgm* на новое название. Как только карта будет закончена, необходимо загрузить эти файлы обратно на робота в папку: *~/catkin\_ws/src/turtlebro\_navigation/maps*. Там уже будут файлы, удаляем их и загружаем новые.

Теперь запустим навигацию по карте. Сбрасываем одометрию и запускаем следующую команду: *roslaunch turtlebro\_navigation turtlebro\_map\_navigation.launch*. Запущена навигация по карте, откройте *Rviz*, добавьте туда *MAP* и посмотрите результат. Ничего не изменилось, потому что необходимо пересобрать пакет, запустите команду: *sudo ./src/catkin/bin/catkin\_make\_isolated -install -DCMAKE\_BUILD\_TYPE=Release -install-space /opt/ros/noetic -DPYTHON\_EXECUTABLE=/usr/bin/python3 -pkg=turtlebro\_navigation* [7]. Теперь робот может перемещаться по карте, даже если убрать все стенки. У него есть виртуальная карта, и за ее пределы он не поедет.

#### **5.4.2. Запуск функции патрулирования**

Необходимо запустить патрулирование, чтобы робот мог перемещаться по точкам постоянно без внешнего вмешательства. Воспользуемся пакетом *turtlebro\_patrol*, чтобы решить эту задачу. В папке *~/catkin\_ws/src/* запустите следующие команды: *git clone https://github.com/voltbro/turtlebro\_patrol/cd ..*

```
catkin_make -pkg turtlebro_patrol
```

Пакет установлен и собран, научимся его запускать. Патрулирование работает по координатам на карте, которая была собрана в прошлом задании. Поэтому сперва определим эти координаты и запишем в отдельный файл. Сбросим одометрию и запустим все узлы запуска навигации по карте из предыдущего задания. После запустим *Rviz* и добавим все необходимые данные на поле. Для того чтобы понимать, какую координату имеет точка, воспользуемся инструментом *Publish Point*. При наведении на карту в правом нижнем углу появятся координаты, их стоит записывать на листе или сразу в файл, чтобы в дальнейшем выстраивать маршрут. Выберите две точки на карте, расстояние между ними должно быть 1.5 м, и запишите их координаты. Координаты в *Rviz* сохраняются в метрах, 1.0 – это 1 м от стартовой точки. Нулевая координата – место, откуда робот начал построение карты.

После того как были выбраны и записаны точки, представьте мысленно этот маршрут. Посмотрите, может ли робот приехать в выбранную точку, может ли он там сделать полный оборот. Робот может объехать препятствие, если он располагается по пути к точке, но он не может остановиться на точке, которая расположена слишком близко к стенке или является стеной. Также нужно определить, в какую сторону будет смотреть робот после остановки. Ноль градусов – это стартовая позиция робота при построении карты; 90 градусов – поворот робота на 90 градусов от стартовой позиции по часовой стрелке. Очень важно правильно выбрать точку, чтобы робот смог доехать до нее, а затем развернуться на нужный градус. Выбор угла поворота должен соответствовать направлению движения робота. Не потому, что робот не сможет сам развернуться, а потому, что так эффективнее. Алгоритм выстраивает маршрут, исходя из того, что робот поедет камерой вперед, лучше, если ему помочь и самому установить нужную точку.

Итак, выберите две точки по правилам выше. Именно по ним робот будет совершать патрулирование, бесконечное перемещение по этому маршруту. Теперь необходимо открыть файл *~/catkin\_ws/*

*src/turtlebro\_patrol/data/goal.xml* и записать в него точки патрулирования. Там уже есть пример с двумя точками. Новые координаты вводятся аналогично примеру.

Перед запуском патрулирования не нужно запускать другие узлы, даже узел навигации – в пакет уже встроен запуск всех нужных модулей, и в этом заключается проблема. Нужно поменять узлы запуска, потому что изначально патрулирование производится по *slam*, а необходимо по готовой карте *map*. Заходим в файл и меняем в нем узел запуска на *map* вместо *slam* ~/*catkin\_ws/src/turtlebro\_patrol/launch/patrol.launch*

Осталось пересобрать пакет и запустить патрулирование: *cd ~/  
catkin\_ws/*

```
catkin_make -pkg turtlebro_patrol
```

Сбрасываем одометрию и запускаем узел *roslaunch turtlebro\_patrol patrol.launch*

Ждем появления надписи *odom received*, если такой надписи нет, то что-то пошло не так, перезагрузите робота. Если надпись есть, то открываем еще один терминал и смотрим, какие топики появились. Должен появиться топик */patrol\_control*, подробнее о командах топика в документации. Откройте *Rviz*, добавьте те же данные, что и для навигации. После можно начинать навигацию. Существует пять основных команд, которые отправляются в топик патрулирования: *start, pause, resume, home, shutdown*. Команды говорят сами за себя, необходимо для запуска патрулирования ввести эту строчку в терминал с командой *start rostopic pub /patrol\_control std\_msgs/String "data: 'next'"*

Подождите пару секунд, робот должен начать патрулирование, если хотите вернуть робота на стартовую позицию или остановить патрулирование, то отправьте соответствующую команду, не забыв закрыть прошлую.

Бывает так, что робот не может проехать до точки, – будет видна красная надпись в терминале с запущенным узлом патрулирования. В этом случае необходимо поменять точки, а лучше поставить вспомогательные. Проследите, чтобы робот выполнил три и более циклов успешно, не забудьте воспользоваться командой «пауза» в процессе.

#### *5.4.3. Модификация параметров пакета навигации*

В следующих заданиях, выполняемых на роботе, необходимо выполнить работы с дополнительным оборудованием, что внесет изменения в размеры роботизированной платформы. Нам нужно

уметь менять размеры робота, чтобы навигация была точной, иначе робот станет больше, а его цифровая копия – нет. Там, где цифровой робот проезжает, физический будет застревать, будет ошибаться одометрия и будут накапливаться ошибки. Поэтому следует научиться менять размеры робота.

У робота есть огромное количество параметров, их все можно менять и настраивать через команду *rosparam* или через файлы *.launch*, если известно, где лежит файл *.launch* для определенного параметра. В данном случае нужен параметр размера, который задается координатами робота, его локальными координатами. Считается, что нулевая позиция – это центр робота, значит, его передний, задний и другой края – это отступы на какое-то количество сантиметров влево, вправо и т. д. Откроем текущие размеры робота следующей командой: *rosparam get /move\_base/local\_costmap/footprint*

Появится список из четырех массивов, состоящий из двух точек: *X*, *Y*. Каждый массив – это координата угла робота, левый передний – первая точка. Следовательно, если необходимо поменять размер робота, то нужно понять, в каком месте и какие точки поменяются. Нужно сделать робота на 5 см длиннее, потому что на нем впереди будет установлен датчик. С помощью команды добавляем 5 см к первой и второй координате *Y*: *rosparam set /move\_base/local\_costmap/footprint '[0.06,0.15],[-0.14,0.15],[-0.14,-0.1],[0.06,-0.1]'*

Теперь робот имеет другие размеры, с 5 см не особо заметно, но теперь робот не сможет близко подъезжать к стенкам. Запустите навигацию в *Rviz* с помощью инструмента *2D Nav Goal*.

## 5.5. Установка дополнительного оборудования

### 5.5.1. Тестирование оборудования

Необходимо проверить работоспособность тепловизора и лампы.

В пункте 5.3.8 уже производилась загрузка программного кода на *Arduino*, и сейчас необходимо выполнить аналогичные действия.

Начнем с проверки работы тепловизора. Программа считывает температуру и отправляет раз в секунду пакет из 64 значений в последовательный порт на скорости 9600 бод. Если вы используете метод подключения к *Arduino* через кабель к своему компьютеру, то нет никакой проблемы считать данные. После загрузки кода нужно выбрать монитор порта *Ctrl + Shift + M* или *Инструменты*

→ *Монитор порта*. Затем выберите скорость считывания данных и ждите вывод данных.

При удаленной загрузке файлов этот способ не сработает. Необходимо запустить команду в терминале, на роботе `screen /dev/ttyUSB0 9600` или `cat /dev/ttyS0`.

Чтобы проверить работоспособность устройства, поднесите свою руку к датчику, выводимые данные должны начать изменяться. В случае ошибок: проверьте код, проверьте подключение по документации, проверьте порты, попросите эксперта проверить устройство на тестовом стенде.

**Проверка лампы.** Код проверки лампы гораздо проще: он включает и выключает устройство раз в секунду. Достаточно правильно подключить устройство и загрузить код на плату.

### 5.5.2. Подключение оборудования к ROS

Необходимо подключить установленные устройства к *ROS*, чтобы было возможно управлять ими через единый интерфейс. Сначала загрузим код на *Arduino*, он создает узел и три топики: *amg88xx\_pixels*, *limit\_switch*, *alarm\_led*. Программный код можно скачать в репозитории *turtlebro\_overheat\_sensor* по ссылке: *turtlebro\_overheat\_sensor/src/arduino/amg88xx\_main/*. После скачиваем репозиторий на робота в папку *~/catkin\_ws/src* и выполняем следующие команды.

```
cd ~/ros_catkin_ws/src
git clone https://github.com/voltbro/turtlebro_overheat_sensor.git
cd ~/ros_catkin_ws
sudo ./src/catkin/bin/catkin_make_isolated --install
-DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/melodic
-pkg=turtlebro_overheat_sensor
```

Если все сделано правильно, то после запуска следующей команды у вас появятся топики, которые были описаны в *Arduino roslaunch turtlebro\_overheat\_sensor heat\_sensor.launch*

Запустите топик *alarm\_led* следующей командой и проверьте, что все работает корректно, включите и выключите лампу: *rostopic pub /alarm\_led std\_msgs/Bool «data: true»* Не забудьте также запустить оставшиеся два топика, чтобы проверить корректность работы кода. Также необходимо провести испытания с нагревательным элементом. С запущенным узлом поставьте робота на испытательное поле и, используя веб-интерфейс, подведите его к устройству. В результате должна загореться лампа.

Стандартные пороговые значения для включения лампы не подходят для соревнований, поэтому нужно их поменять. Файл конфигурации находится в директории `~/ros_catkin_ws/src/turtlebro_overheat_sensor/launch/heat_sensor.launch`. В нем необходимо заменить пороговые значения температуры с 55 на 45. Пересоберите пакет и проведите повторные испытания.

## 5.6. Проведение рабочих испытаний модернизированного робота

### 5.6.1. Стендовые испытания

Необходимо, чтобы робот корректно «отработал» один «неисправный» нагревательный элемент, используя совместную работу пакетов `turtlebro_patrol` и `turtlebro_overheat_sensor`. Необходимо последовательно настроить и запустить два пакета, продемонстрировав, что возможно начать масштабировать результат.

Запустите пакет навигации и выберите две точки: стартовая – необязательно 0,0 и точка с нагревательным элементом. Перед тем как выбирать вторую точку, рекомендовано включить пакет `turtlebro_overheat_sensor` и с помощью веб-интерфейса подвести робота к устройству. Посмотрите, когда робот точно обнаруживает неисправность, удобно ли ему туда подъехать, может ли он там развернуться. Это очень важные вопросы: от расположения точки зависит время выполнения задания, а также сам факт определения неисправности. Возможно неправильно указать угол поворота, и робот не сможет навестись на устройство. Если выбрано две точки, то следуйте указаниям подраздела 5.4.2 и поменяйте точки.

Сбросьте одометрию, запустите последовательно два пакета и отправьте в топик патрулирования команду «начать». Робот должен выполнить минимум три повторения. Во время теста меняйте нагревательный элемент, чтобы избежать ошибок ложного срабатывания.

### 5.6.2. Испытания на полигоне

Это последнее задание из стандартных. Оно нацелено на проверку умения работать со всеми модулями одновременно. Так как задание объемное и выполнение его полностью занимает время, то стоит следить за зарядом аккумулятора и поставить его на зарядку, если он меньше 50%. При низком заряде робот может некорректно выполнять задачу.

Необходимо составить маршрут движения робота, который включает все нагревательные элементы, в случае нахождения не-

исправного робот отправляет команду с данными о времени, месте и температуре неисправного прибора.

Начнем с создания точек. Так как это патрулирование, то необязательно включать стартовую точку. Робот может начинать с любого места, при старте узла он отправится в первую точку самостоятельно. Через веб-интерфейс доведите робота до устройств, чтобы определить, видит ли робот неисправность, попросите поменять устройства, чтобы проверить точку, где стоит холодный прибор. Если все точки отмечены, то поставьте робота на стартовую позицию,бросьте одометрию. Используя инструмент *2D Nav Goal*, в *Rviz* при включенном пакете *turtlebot\_overheat\_sensor* вручную укажите роботу место и угол поворота, который был записан ранее. Повторите так шесть раз. Если робот успешно подъезжает к устройству, видит неисправность и едет к следующему, а главное, стабильно выполняет это раз за разом, то загружайте точки в файл и пересоберите пакет. Если возникают проблемы, то меняйте точки.

Небольшой совет: если на поле три нагревательных элемента, это не значит, что должно быть указано три точки. Программа сама выстраивает маршрут, но если расстояние до точки небольшое, а путь прост – он прямой, в противном случае можно ставить дополнительные точки, чтобы робот развернулся и выровнялся.

Если все работает корректно, то необходимо поставить робота на стартовую позицию, сбросить одометрию, запустить последовательно пакеты и показать работу. Не забудьте вызывать топик *heat\_sensor\_output*, чтобы показать корректность работы.

### 5.6.3. Создание пакетного файла *.launch*

Для того чтобы постоянно не вызывать два пакета, необходимо написать файл *.launch*, который объединил бы пакеты. Создадим такой файл на основе *heat\_sensor.launch*, назовем его *heat\_sensor\_patrol.launch*

Пакетный файл описывает, что включает в себя пакет и как его правильно запускать, поэтому не обязательно, чтобы используемые пакеты лежали в одной папке. Для примера посмотрите, как устроен *patrol.launch*. Самыми первыми строчками включим пакет навигации по карте и пакет патрулирования, все остальное оставим неизменным. Как только файл будет готов, сохраните документ, пересоберите пакет и запустите его, используя наш новый файл. Все должно работать.

## **5.7. Безопасное выключение (сохранение файлов на *SD*-карту)**

Если вы редактировали файлы на роботе и хотите действительно убедиться, что при выключении робота они не потеряются, то, прежде чем выключить робота, необходимо выполнить команду *sudo sync*. Данная команда запишет все «закешированные» файлы на *SD*-карту. Только после этого можно выключить робота выключателем или перезагрузить его.

## **6. ДЕМОНСТРАЦИОННЫЕ ЗАДАНИЯ С ОПИСАНИЕМ РАБОТ**

### **Модуль 1. Ввод робота в эксплуатацию**

#### ***Описание модуля***

Участнику необходимо выполнить приемку нового робота. Перед началом выполнения задания модуля участник получает доступ к роботу и сопроводительную документацию. После этого участник должен самостоятельно выполнить задания модуля.

Инструкция использования удаленного полигона и подключения к роботу: <https://www.notion.so/vbtodo/VPN-2adea9c7404c468ba346123ae261893f>.

#### ***1.1. Проверка базовой информации робота***

Участнику необходимо получить базовую информацию о конфигурации / установленном программном обеспечении на роботе и внести ее в сервисную книжку (внесение информации в сервисную книжку происходит во время сдачи модуля). Для этого необходимо:

- 1) продемонстрировать версию дистрибутива *ROS*;
- 2) продемонстрировать версию пакета *turtlebro*;
- 3) продемонстрировать версию прошивки микроконтроллера материнской платы;
- 4) продемонстрировать серийный номер системной платы робота (*mcu\_id*);
- 5) продемонстрировать размер оперативной памяти (*Kb*);
- 6) продемонстрировать полное пространство на *SD*-карте (*Gb*);
- 7) продемонстрировать версию образа ОС, установленной на *Raspberry Pi*.

#### ***1.2. Проверка процессов ROS***

Необходимо проверить наличие топиков и сервисов, указанных в инструкции, в списках всех топиков и сервисов, запущенных на роботе. После проверки сделать отметку в сервисной книжке. Для этого необходимо:

- 1) продемонстрировать получение списка топиков на роботе;
- 2) продемонстрировать сверку списка топиков на роботе и списка топиков, указанных в инструкции;
- 3) продемонстрировать получение списка сервисов на роботе;
- 4) продемонстрировать список сервисов, указанных в инструкции.

### ***1.3. Проверка работы камеры***

Необходимо проверить работу камеры робота, полученные результаты занести в сервисную книжку:

- 1) продемонстрировать наличие подключенной камеры в `/dev/...`;
- 2) продемонстрировать получение данных о максимальных разрешениях камеры;
- 3) продемонстрировать работу камеры через веб-интерфейс.

### ***1.4. Проверка одометрии колес***

Необходимо проверить наличие и корректность получаемой одометрии:

- 1) продемонстрировать сброс одометрии;
- 2) продемонстрировать запуск робота на выполнение вращения вправо по оси  $Z$  и корректность отображения получаемой одометрии в соответствии с осями направления робота;
- 3) продемонстрировать запуск робота на выполнение вращения влево по оси  $Z$  и корректность отображения получаемой одометрии в соответствии с осями направления робота;
- 4) занести результаты в сервисную книжку.

### ***1.5. Проверка контроля напряжения аккумуляторной батареи***

Необходимо выполнить следующие работы:

- 1) продемонстрировать вывод напряжения батареи в соответствующем топике;
- 2) сделать запись о проверке корректности вывода напряжения батареи в сервисной книжке.

### ***1.6. Проверка IMU-датчика***

Необходимо проверить работу IMU-датчика:

- 1) продемонстрировать данные IMU-датчика в консоли;
- 2) продемонстрировать проверку соответствия частоты обновления данных IMU с документацией на робота;
- 3) продемонстрировать корректную работу IMU-датчика в *Rviz*;
- 4) сделать запись о проверке корректности работы IMU-датчика в сервисной книжке.

### ***1.7. Проверка МК Atmega***

Необходимо проверить работу микроконтроллера *Atmega* и связи с *ROS*:

- 1) необходимо загрузить тестовую прошивку *TB-RGBline-test.ino.mega.hex* из *git*-репозитория;
- 2) продемонстрировать процесс загрузки тестовой прошивки на робота;
- 3) продемонстрировать проверку работоспособности светодиодной ленты;
- 4) сделать запись о проверке корректности работы светодиодной ленты в сервисной книжке.

## **Модуль 2. Настройка и запуск робота**

### ***Описание модуля***

После приемки необходимо выполнить настройку робота для выполнения задач предприятия. Для этого необходимо настроить робота для выполнения задачи автономного патрулирования. Во время выполнения задачи можно обращаться к экспертам для установки робота на полигоне в начальную позицию и подключения к автономному питанию.

#### ***2.1. Автономная навигация***

Необходимо произвести следующие действия:

- 1) продемонстрировать запуск на роботе программного обеспечения для автономной навигации в режиме *SLAM* (на роботе установлен пакет): [https://github.com/voltbro/turtlebro\\_navigation](https://github.com/voltbro/turtlebro_navigation);
- 2) продемонстрировать запуск *Rviz* и движение робота при помощи указания целей в *Rviz* на примере не менее двух целей через внешнюю веб-камеру;
- 3) продемонстрировать процесс создания карты полигона при помощи *SLAM*-навигации;
- 4) продемонстрировать созданную карту в *Rviz*;
- 5) продемонстрировать сохранение карты на робота, копирование файла-изображения карты с робота в домашнюю директорию ПК участника.
- 6) загрузить файл-изображение карты в облачное хранилище для проверки;
- 7) продемонстрировать сохраненное в домашней директории ПК изображение при помощи любой программы отображения изображений. Построенная карта не должна иметь неизвестных зон и должна быть готова для навигации.

## **2.2. Запуск функции патрулирования**

Необходимо настроить робота для осуществления патрулирования по полигону:

- 1) продемонстрировать установку на робота пакета *turtlebro\_patrol*: [https://github.com/voltbro/turtlebro\\_patrol/](https://github.com/voltbro/turtlebro_patrol/);
- 2) продемонстрировать проверку установки всех необходимых *ROS* пакетов для *turtlebro\_patrol* на роботе;
- 3) продемонстрировать файл с координатами задания маршрута патрулирования по двум точкам: (точка старта  $\rightarrow$  1,5 м вперед по оси *X* от точки старта);
- 4) продемонстрировать запуск пакета *turtlebro\_patrol*;
- 5) продемонстрировать работу патрулирования (не менее трех циклов), включая выполнение задания:
  - a) начало патрулирования;
  - b) пауза в патрулировании;
  - c) возобновление патрулирования;
  - d) остановка патрулирования и завершение работы пакета *turtlebro\_patrol*.

## **2.3. Модификация параметров пакета навигации**

На передней части робота установлено дополнительное оборудование. В результате этого размеры робота изменились (передний край робота переместился на 5 см вперед по оси *X*):

- 1) необходимо продемонстрировать процесс увеличения размеров робота (*footprint*) в пакете автономной навигации;
- 2) продемонстрировать новые размеры через *rosparams*;
- 3) продемонстрировать корректную работу навигации с измененными параметрами на примере движения до не менее двух точек, задаваемых из *Rviz*.

Для дополнительной информации обратитесь к курсу автономной навигации по ссылке: <http://learn.voltbro.ru/data/ros-navigation>.

## **Модуль 3. Сервисная диагностика и обслуживание робота**

### ***Описание модуля***

В этом модуле необходимо провести регулярное сервисное обслуживание робота.

#### ***3.1. Проверка сервисных пакетов***

Для функционирования роботу необходимо наличие дополнительных *ROS*-пакетов, содержащих сервисные скрипты, конфигу-

рационные файлы и другие служебные данные. Эти пакеты производитель размещает в *git*-репозиториях:

*https://github.com/voltbro/ws service pkg 1*

*https://github.com/voltbro/ws service pkg 3*

Необходимо:

1) продемонстрировать обновление исходного кода для пакета *ws\_service\_pkg\_1* до последней версии, соответствующей версии в репозитории;

2) продемонстрировать установку в *ROS* обновленного пакета;

3) продемонстрировать запуск диагностического скрипта в пакете *ws\_service\_pkg\_1*;

4) записать параметр конфигурации (*configuration checksum*), полученный в результате работы скрипта пакета в сервисную книжку.

### **3.2. Обновление версии сервисных пакетов**

По требованию производителя некоторые сервисные пакеты по результатам прошлой проверки необходимо откатить к определенной прошлой версии, которая хранится в одном из прошлых коммитов репозитория. Необходимо посмотреть в локальном репозитории список коммитов и выбрать тот, в котором версия пакета соответствует требуемой. Необходимо произвести следующие действия:

1) продемонстрировать получение списка всех коммитов пакета *ws\_service\_pkg\_3*;

2) продемонстрировать обновление пакета *ws\_service\_pkg\_3* к версии 2.8.2;

3) продемонстрировать установку в *ROS* пакета *ws\_service\_pkg3* версии 2.8.2;

4) продемонстрировать запуск диагностического скрипта в пакете *ws\_service\_pkg\_3*;

5) записать параметр конфигурации (*configuration checksum*), полученный в результате работы скрипта в сервисную книжку.

## **Модуль 4. Установка и настройка дополнительного оборудования**

### ***Описание модуля***

Участнику необходимо оснастить роботов оборудованием для контроля противопожарной обстановки, настроить его и ввести в эксплуатацию. В дистанционном формате демонстрационного экзамена эксперты перед началом модуля устанавливают на робо-

тов дополнительное оборудование и предоставляют документацию к нему.

Участнику необходимо выполнить установку программного обеспечения для использования дополнительного оборудования, сорвать и настроить пакет для *ROS* для обработки данных с оборудования.

Оборудование: тепловизор *AMG8833* (подключен по *I2C*) и сигнальная лампа (подключена к pinу *A12*).

Дополнительные данные: примеры программного обеспечения для использования тепловизора и документация к тепловизору.

Участник должен самостоятельно устанавливать и конфигурировать как базовое программное обеспечение робота, так и программное обеспечение дополнительного оборудования.

#### ***4.1. Тестирование дополнительного оборудования***

Необходимо разработать и продемонстрировать исходный код тестовой прошивки для проверки подключения дополнительного оборудования, продемонстрировать его загрузку на микроконтроллер (МК) робота и корректную работу.

1) Прошивка. Реализует мигание сигнальной лампой с интервалом в 1 с.

2) Загрузить файл с исходным кодом прошивки в облачное хранилище.

При демонстрации работы необходимо показать исходный код программы с краткими комментариями голосом, процесс загрузки на робота.

#### ***4.2. Подключение оборудования к ROS***

В репозитории *GitHub* находится пакет *ROS turtlebro\_overheat\_sensor* для работы с дополнительным оборудованием. Для того чтобы получать данные от дополнительного оборудования и вывести их в *ROS*, необходимо также загрузить на контроллер расширения соответствующую прошивку. Ее можно найти в том же репозитории по адресу: `src/arduino/amg88xx_main/amg88xx_main.ino`. Инструкцию по удаленной загрузке скетча *Arduino* можно найти в инструкции к роботу.

Необходимо:

1) продемонстрировать установку пакета, работающего с дополнительным оборудованием: [https://github.com/voltbro/turtlebro\\_overheat\\_sensor](https://github.com/voltbro/turtlebro_overheat_sensor).

2) продемонстрировать работу с дополнительным оборудованием через *ROS*:

а) работа сигнальной лампы через публикацию в соответствующий топик *ROS*;

б) работа тепловизора через подписку на сообщения соответствующего топика *ROS*;

3) продемонстрировать модернизацию настроек пакета *turtlebro\_overheat\_sensor*. По умолчанию робот реагирует на температуру 55 градусов. Необходимо изменить граничное значение работы датчика на 45 градусов:

а) продемонстрировать внесенные изменения в конфигурационный файл;

б) продемонстрировать запуск пакета *turtlebro\_overheat\_sensor*;

с) продемонстрировать корректную работу модернизированного пакета путем подведения робота к нагревательному элементу через веб-интерфейс.

## **ЗАКЛЮЧЕНИЕ**

Полученные в данном пособии сведения помогут овладеть базовыми навыками в области сервисной робототехники, получить набор компетенций для обслуживания, настройки, диагностики сервисных роботов.

Пособие формирует базовые знания в области администрирования ОС *Linux*, программирования на языке *Python*, а также навыки и компетенции, связанные с ОС для роботов – *ROS*. Кроме того, пособие направлено на решение задач по подготовке специалистов, эксплуатантов сервисных роботов к демонстрационному экзамену по стандарту *WorldSkills*. Представленной информации достаточно для успешной сдачи демонстрационного экзамена.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Qiglei M., Gerkey B., Smart W. D.* Programming Robots with ROS: a practical introduction to the Robot Operating System. O'Reilly Media, Inc. 2015. 448 p.
2. *Ramkumar G., Lentin J.* ROS Robotics Projects. Packt Publishing, 2019. 256 p.
3. *Bernardo R. J.* Hands-On ROS for Robotics Programming. Packt Publishing, 2020. 432 p.
4. *Lentin J.* Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy. Apress, 2018. 293 p.
5. *Слабуха Н.* Введение в ROS. URL: <http://docs.voltbro.ru/startng-ros/developing/new-package.html> (дата обращения: 16.10.2021).
6. Пакет автономной навигации. URL: [https://github.com/voltbro/turtlebro\\_navigation](https://github.com/voltbro/turtlebro_navigation) (дата обращения: 16.10.2021).
7. Репозиторий файлов для задач WorldSkills. URL: <https://github.com/voltbro/ws-sro> (дата обращения: 16.10.2021).

## **СВЕДЕНИЯ ОБ АВТОРАХ**

**Ефимов Павел Андреевич** – технический директор-руководитель Центра проектной деятельности Дальневосточного федерального университета, эксперт компетенции «Эксплуатация сервисных роботов».

**Сыч Александр Александрович** – специалист, руководитель направления «Робототехника» Центра проектной деятельности Дальневосточного федерального университета.

**Баранчугов Илья Александрович** – ассистент департамента компьютерно-интегрированных производственных систем Дальневосточного федерального университета.

**Морозова Нина Тихоновна** – кандидат технических наук, доцент департамента автоматики и робототехники Дальневосточного федерального университета.

**Ноткин Борис Сергеевич** – кандидат технических наук, доцент департамента компьютерно-интегрированных производственных систем Дальневосточного федерального университета.

## СОДЕРЖАНИЕ

Предисловие .....	3
Введение.....	5
1. Основы операционной системы Linux .....	6
1.1. Операционная система Linux .....	6
1.2. ОС Linux и виртуальная машина .....	7
1.3. Основы работы в командной строке Linux .....	9
1.4. Команды терминала ОС Linux .....	11
1.4.1. Команда ls.....	12
1.4.2. Команда clear .....	13
1.4.3. Команда pwd.....	14
1.4.4. Команда cd .....	14
1.4.5. Команда mkdir .....	15
1.4.6. Команда mv .....	16
1.4.7. Команда cp .....	17
1.4.8. Команда rm .....	18
1.4.9. Команда touch .....	18
1.4.10. Команда cat .....	18
1.4.11. Команда nano .....	19
1.4.12.  (Вертикальный разделитель).....	20
1.4.13. Команда grep .....	21
1.4.14. Команда scp .....	21
1.5. Запуск программ.....	22
1.6. Удаленное управление операционной системой .....	22
1.7. Пользователи и права пользователей .....	25
1.7.1. Основная информация о пользователях .....	25
1.7.2. Отображение информации о владельце файла и каталога.....	26
1.7.3. Команда chrow .....	27
1.7.4. Права доступа.....	28
1.7.5. Команда chmod .....	29
1.7.6. Команда sudo.....	30
1.7.7. Команда apt.....	31
1.8. Практические задания.....	31
2. Программирование на языке Python .....	32
2.1. История Python .....	32
2.2. Зачем создан Python .....	32
2.3. Особенности программирования на Python .....	33
2.4. Приложения на Python .....	34
2.4.1. Веб-приложения .....	34
2.4.2. Научные и цифровые вычисления .....	34

2.4.3. Создание прототипов программного обеспечения .....	34
2.5. Инструменты для разработки .....	36
2.6. Интерпретатор Python .....	37
2.7. Простые программы на языке Python .....	38
2.7.1. Вывод данных на экран .....	38
2.7.2. Синтаксис языка программирования .....	39
2.8. Ввод и вывод данных .....	41
2.8.1. Вывод данных.....	41
2.8.2. Ввод данных .....	42
2.9. Математические операторы и библиотеки.....	44
2.9.1. Целые числа и операции над ними .....	44
2.9.2. Действительные числа .....	45
2.9.3. Библиотека math.....	47
2.10. Условия и логические операторы.....	49
2.10.1. Условия .....	49
2.10.2. Логические операторы .....	51
2.11. Коллекции Python.....	51
2.11.1. Строки .....	52
2.11.2. Срезы.....	53
2.11.3. Списки.....	54
2.11.4. Цикл while.....	55
2.11.5. Цикл for.....	56
2.12. Функции языка Python .....	57
2.12.1. Функции .....	57
2.12.2. Стандартные функции в языке Python.....	58
2.13. Основы объектно-ориентированного программирования .....	59
2.13.1. Создание классов .....	59
2.13.2. Использование self в классах.....	60
2.13.3. Метод __init__ .....	60
2.14. Концепции ООП .....	60
2.14.1. Наследование .....	61
2.14.2. Создание классов-наследников .....	61
2.14.3. Инкапсуляция .....	62
2.14.4. Полиморфизм .....	63
2.15. Практические задания .....	64
3. ROS – Robot Operating System .....	66
3.1. Причины использования ROS .....	67
3.2. Положительные качества ROS .....	67
3.3. Введение в ROS.....	68
3.4. Сообщения .....	70
3.4.1. Типы данных в ROS .....	70

3.4.2. Получение информации о сообщениях .....	71
3.4.3. Topic .....	72
3.4.4. Утилита rostopic .....	73
3.5. Сервисы ROS .....	75
3.5.1. Описание сервисов в ROS.....	76
3.5.2. Утилита rosservice.....	76
3.6. Основные термины ROS .....	77
3.6.1. Мастер (Master), мастер-нод.....	77
3.6.2. Нод (Node).....	78
3.6.3. Пакет (Package).....	78
3.7. Стандарты ROS.....	78
3.7.1. Единицы измерений .....	78
3.7.2. Координаты X, Y, Z .....	79
3.7.3. Оси вращения .....	79
3.7.4. Стандарты кодирования.....	80
3.8. Создание пакетов ROS .....	80
3.8.1. Описание пакета в ROS .....	80
3.8.2. Структура пакета .....	81
3.8.3. Установка пакета ROS из репозитория .....	82
3.8.4. Установка пакетов из исходного кода.....	84
3.8.5. Настройка рабочего пространства .....	84
3.8.6. Сборка пакета из исходного кода .....	85
3.9. Создание пакета ROS .....	86
3.9.1. Модуль catkin .....	86
3.9.2. Создание пользовательского пакета .....	86
3.9.3. Работа с зависимостями .....	87
3.9.4. Библиотека rospy .....	88
3.10. Написание издателя на языке Python.....	89
3.11. Написание подписчика на языке Python .....	91
3.12. Сборка пакетов .....	93
3.13. Запуск пакетов .....	94
3.13.1. Утилита rosrun.....	94
3.13.2. Утилита roslaunch .....	95
3.14. Практические задания .....	98
4. Программирование микроконтроллерных плат.....	99
4.1. Основная информация о системной плате робота .....	99
4.2. Среда разработки Arduino IDE .....	102
4.3. Описание среды разработки .....	104
4.4. Написание простой программы в Arduino IDE .....	105
4.5. Arduino и ROS .....	107
4.6. Скрипт издателя ROS .....	109
4.7. Скрипт подписчика ROS .....	112

4.8. Практические задания.....	115
<b>5. Управление роботизированной платформой TurtleBro .....</b>	<b>116</b>
5.1. Raspberry Pi .....	117
5.2. Настройка подключения к новой сети Wi-Fi через SD-карту.....	118
5.3. Ввод робота в эксплуатацию .....	119
5.3.1. Проверка базовой информации робота .....	119
5.3.2. Проверка ROS-процессов .....	120
5.3.3. Проверка работы камеры.....	120
5.3.4. Проверка одометрии колес.....	125
5.3.5. Проверка контроля напряжения аккумуляторной батареи .....	127
5.3.6. Проверка лидара .....	127
5.3.7. Проверка IMU-датчика .....	129
5.3.8. Проверка микроконтроллера Atmega .....	130
5.4. Настройка и запуск робота .....	131
5.4.1. Автономная навигация .....	131
5.4.2. Запуск функции патрулирования .....	134
5.4.3. Модификация параметров пакета навигации .....	135
5.5. Установка дополнительного оборудования.....	136
5.5.1. Тестирование оборудования .....	136
5.5.2. Подключение оборудования к ROS.....	137
5.6. Проведение рабочих испытаний модернизированного робота.....	138
5.6.1. Стендовые испытания .....	138
5.6.2. Испытания на полигоне .....	138
5.6.3. Создание пакетного файла .launch .....	139
5.7. Безопасное выключение (сохранение файлов на SD-карту) .....	140
<b>6. Демонстрационные задания с описанием работ .....</b>	<b>141</b>
Модуль 1. Ввод робота в эксплуатацию.....	141
1.1. Проверка базовой информации робота.....	141
1.2. Проверка процессов ROS.....	141
1.3. Проверка работы камеры .....	142
1.4. Проверка одометрии колес .....	142
1.5. Проверка контроля напряжения аккумуляторной батареи .....	142
1.6. Проверка IMU-датчика .....	142
1.7. Проверка МК Atmega .....	142
Модуль 2. Настройка и запуск робота.....	143
2.1. Автономная навигация .....	143
2.2. Запуск функции патрулирования.....	144

2.3. Модификация параметров пакета навигации .....	144
Модуль 3. Сервисная диагностика и обслуживание робота .....	144
3.1. Проверка сервисных пакетов .....	144
3.2. Обновление версии сервисных пакетов .....	145
Модуль 4. Установка и настройка дополнительного оборудования.....	145
4.1. Тестирование дополнительного оборудования .....	146
4.2. Подключение оборудования к ROS .....	146
Заключение .....	148
Библиографический список.....	149
Сведения об авторах .....	150

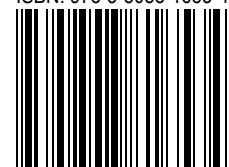
Учебное издание

**Ефимов Павел Андреевич,  
Сыч Александр Александрович,  
Баранчугов Илья Александрович,  
Морозова Нина Тихоновна,  
Ноткин Борис Сергеевич**

## ЭКСПЛУАТАЦИЯ СЕРВИСНЫХ РОБОТОВ

Учебное пособие

ISBN: 978-5-8088-1653-4



9 785808 816534

Редактор *O. Ю. Багиева*  
Компьютерная верстка *Ю. В. Умницыной*

---

Подписано к печати 23.11.21. Формат 60 × 84 1/16.  
Усл. печ. л. 9,0. Уч.-изд. л. 9,3.  
Тираж 100 экз. Заказ № 523.

---

Редакционно-издательский центр ГУАП  
190000, Санкт-Петербург, Б. Морская ул., 67