PROGRAMLAB
INNOVATIVE DIGITAL SYSTEMS

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

ВИРТУАЛЬНЫЙ УЧЕБНЫЙ КОМПЛЕКС «ОТРАБОТКА НАВИГАЦИИ И УПРАВЛЕНИЯ ДВИЖУЩЕГОСЯ ОБЪЕКТА ВНЕ ПОМЕЩЕНИЯ С ИСПОЛЬЗОВАНИЕМ НЕЙРОННЫХ СЕТЕЙ»

PLLLC.RU

ОГЛАВЛЕНИЕ

Общая информация 5
Введение в ROS 6
Назначение ROS 6
Базовые понятия ROS 7
Обмен сообщениями 9
Утилиты 10
Поставляемое ПО 12
Установка необходимых пакетов для ПК. 13
Директория ROS: 14
Первый запуск 14
LIDAR 14
Используемые топики 15
Топики для Lidar 15
Топики для просмотра камер: 15
Разработка и эксплуатация собственного функционала 16
Использование msg и srv 17
Информация о msg 17
Создание нового msg 17
Использование rosmsg 20
Создание srv 20

Использование rossrv 21
Общее для msg и srv 22
Создание Publisher ноды 23
Объяснение кода 24
Написание Subscriber ноды 27
Объяснение кода 28
Создание нод 28
Запуск publisher 28
Запуск subscriber 29
Инструкция по установке и запуску проекта 31
Начало работы с комплексом 34
Режимы работы ПО 41
Работа с комплексом в режиме редактора карт 42
Управление при помощи геймпада 51

Общая информация

Программно-аппаратный комплекс (далее - комплекс) для отработки навигации и управления движущегося объекта вне помещения с использованием нейронных сетей должен позволять проводить практические занятий по решению ряд задач навигации, встающие перед наземными системами автоматического в открытом, преимущественно городском пространстве с рассеянным освещением.

Внешний вид комплекса

Введение в ROS

Robot Operating System (ROS) - это гибкая платформа (фреймворк) для разработки программного обеспечения роботов. Это набор разнообразных инструментов, библиотек и определенных правил, целью которых является упрощение задач разработки ПО роботов.

Создание действительно надежного, универсального программного обеспечения для роботов чрезвычайно сложная задача. С точки зрения робота, проблемы, которые кажутся тривиальными для людей, часто требуют очень сложных технических решений. Часто разработка такого решения не под силу одному человеку.

ROS была создана, чтобы стимулировать совместную разработку программного обеспечения робототехники. Каждая отдельная команда может работать над одной конкретной задачей, но использование единой платформы, позволяет всему сообществу получить и использовать результат работы этой команды для своих проектов.

Назначение ROS

Целью создания ROS является создание "**среды разработки, которая позволяет разработчикам ПО для роботов сотрудничать на глобальном уровне**»

ROS сосредоточена на максимизации повторного использования кода при разработке. Основные характеристики позволяющие это реализовать:

Распределенные процессы: Структура ROS создана в виде минимальных единиц исполняемых процессов (нод), и каждый процесс выполняется изолированно. Взаимодействие разных нод происходит только на уровне обмена сообщениями.

Управление пакетами. Несколько процессов, имеющих общую задачу, объединяются в пакеты. Управление пакетами подразумевает набор утилит, позволяющие автоматически скачивать, устанавливать и удалять пакеты. Пакетный менеджер гарантирует работоспособность и целостность установленных пакетов.

Публичные репозитории и документация: Каждый доступный пакет публикуется в публичном репозитории. Документация пакетов, публикуется в единой системе, которая упрощает поиск необходимых пакетов.

Единое API: При разработке программы, использующей ROS, вы получаете простое и легко встраиваемое API. В примерах программ вы увидите, что использование API не сильно отличается от языка (C ++ или Python). При этом нет разницы при использовании API на каком языке была написана программа.

Базовые понятия ROS

Основные термины

Мастер (Master), Мастер-Нода

Мастер выполняет роль сервера имен для возможности подключения между собой различных нод. Команда roscore запускает сервер мастера, и после этого к нему могут подключиться и зарегистрироваться ноды ROS. Связь между нодами (обмен сообщениями), невозможна без запущенного мастера.

При запуске ROS roscore, мастер будет запущен по адресу URI, установленным в переменной окружения ROS_MASTER_URI. По умолчанию адрес использует IP-адрес локального ПК и номер порта 11311

Нода (Node)

Понятие ноды, относится к наименьшей "рабочей" единицы используемой в ROS. Можно провести аналогию с одной исполняемой программой. ROS рекомендует создать одну ноду для каждой задачи, что позволит легче использовать ее в других проектах.

При запуске нода регистрирует информацию о себе на мастере (название ноды, типы обрабатываемых сообщений). Зарегистрированная нода может взаимодействовать с другими нодами (получать и отправлять запросы). Важно отметить, что обмен сообщениями между нодами работает без участия мастера (соединение между нодами происходит на прямую). Мастер обеспечивает только единое пространство имен для решения вопроса куда подключиться к конкретной ноде. Адрес запуска ноды, берётся из переменной окружения ROS_HOSTNAME, которая должна быть определенна до запуска. Порт устанавливается на произвольное уникальное значение.

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 поставить готовым из репозитория пакетов, так и скачать и скомпилировать из исходных кодов.

Сообщение (Message)

Ноды отправляют и принимают данные между собой, согласно заданного формата. Эти данные называют Сообщения, а описание Типом Сообщения.

Сообщения могут быть как простых типов (integer, float, boolean), так и могут состоять из сложных структур, содержащих вложенные сообщения и массивы сообщений).

Например, для сообщения с координатами обьекта (XYZ) есть существующий тип сообщения geometry_msgs/Point.msg который описывается:

Топик (Topic), модель Издатель и Подписчик

Топик (Topic), это один из видов обмена сообщениями, который буквально похож на тему в разговоре. Нода издателя (publisher) сначала регистрирует свою тему на мастере, а затем начинает публикацию сообщений в эту тему (топик). Узлы подписчиков, которые хотят получать информацию из этой темы (топика) при помощи мастера получают адрес этой темы и далее получают сообщения из этого топика.

Издатель (Publisher)

Издателем называется процесс, который рассылает сообщения в рамках созданного топика для других нод. Одна нода может содержать несколько издателей, публикующих данные в разные топики.

Подписчик (Subscriber)

Подписчиком (Subscriber) называется процесс, который получает сообщения из определенного топика (Topic). Подписчик (Subscriber) регистрируется на Мастере (Master), указывая какие топики Подписчик (Subscriber) хочет получать. После этого Издатель (Publisher) начинает отправлять сообщения подписчику. Связь с топиком для подписчика является асинхронной (издатель публикует сообщения в независимости от статуса подписчиков).

Этот тип взаимодействия удобно применять для работы с датчиками, которые непрерывно передают полученные значения.

Сервис (Service), Сервис Клиент и Сервис Сервер

Сервис — это модель коммуникации, работающая по принципу синхроной двунаправленной связи между клиентом (Service Client), который запрашивает данные и сервером (Service Server), который отвечает на запросы.

Сервис Сервер (Service Server)

«Сервис Сервер» — это узел коммуникации (процесс), который получает запрос, обрабатывает данные и передает обратно ответ. Запрос и ответ представляют собой обычное Сообщение (Message).

Сервис Клиент (Service Client)

Сервис Клиент — это узел коммуникации (процесс), который создает запрос на Сервис Сервере (Service Server) и получает ответ после выполнения запроса.

Данная модель взаимодействия применяется для удаленного выполнения различных операций в рамках разных нод.

Обмен сообщениями

Как мы обсудили ранее, ключевая концепция ROS предполагает создание множества независимых нод, которые взаимодействуют друг с другом. В этом главе мы подробно рассмотрим способы коммуникации между нодами.

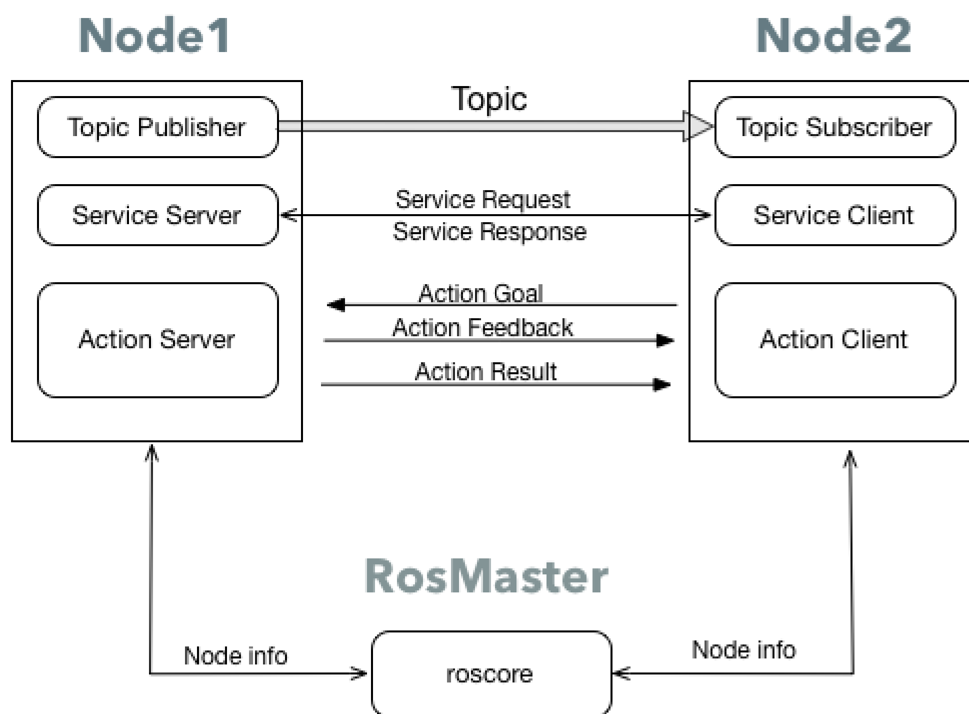
Существуют три основных способа (концепции) коммуникации:

Топик (Theme), который обеспечивает синхронную однонаправленную передачу / прием сообщений;

Сервис (Service), который обеспечивает синхронное двунаправленное взаимодействие: запрос / ответ сообщения;

Действие (Action), которое обеспечивает асинхронное двунаправленное взаимодействие с шагами: цель - результат - обратная связь.

Диаграмму коммуникации можно изобразить схемой:



Утилиты

Утилита для визуализации Rviz

Rviz – это инструмент с открытым исходным кодом, предназначенный для визуализации процессов и отладки алгоритмов робототехнической системы.

Rviz является частью пакета ROS Noetic, поэтому дополнительная установка не требуется, но если Rviz нет в перечне пакетов, то его можно установить отдельно, для этого используйте следующую команду:

1	sudo apt-get install ros-noetic-rviz

Инструмент rviz позволяет в реальном времени визуализировать на 3D-сцене все компоненты робототехнической системы — системы координат, движущиеся части, показания датчиков, изображения с камер.

Настроенный роутер идет в поставке с комплексом, аппаратный комплекс автоматически подключится к нему при запуске роутера.

Для запуска визуализации в реальном времени, необходимо подключиться к нему по Wi-Fi и запустить rviz, указав соответствующий ROS_MASTER_URI:

1	ROS_MASTER_URI=http://192.168.0.100:11311

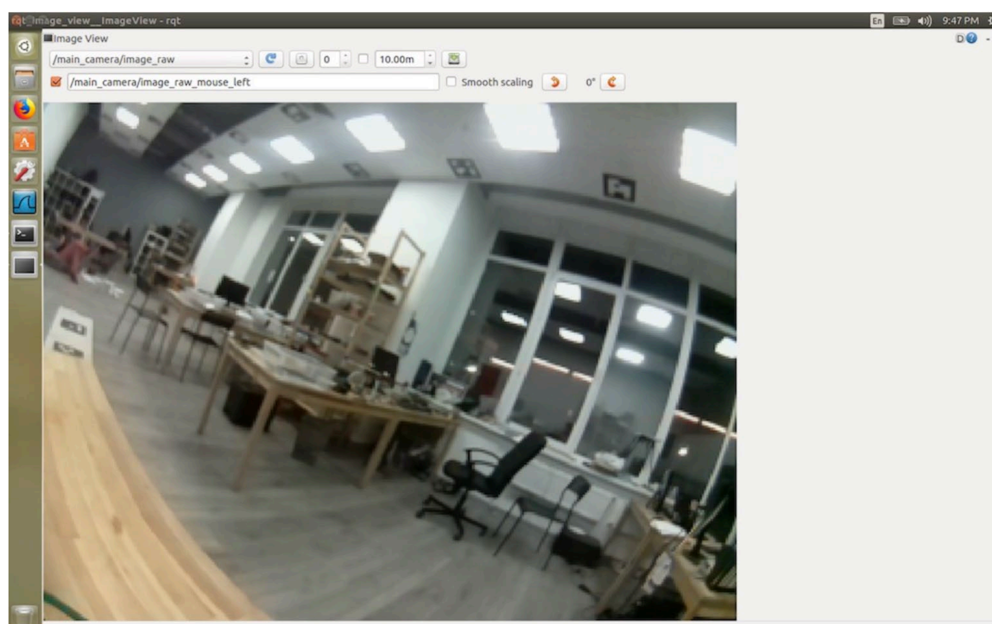
Утилита для просмотра изображения rqt_image_view

Для того, чтобы начать работу с утилитой подключитесь к Wi-Fi сети Клевера и запустите rqt_image_view с указанием его IP-адреса:

ROS_MASTER_URI=http:// 192.168.0.100:11311rqt_image_view

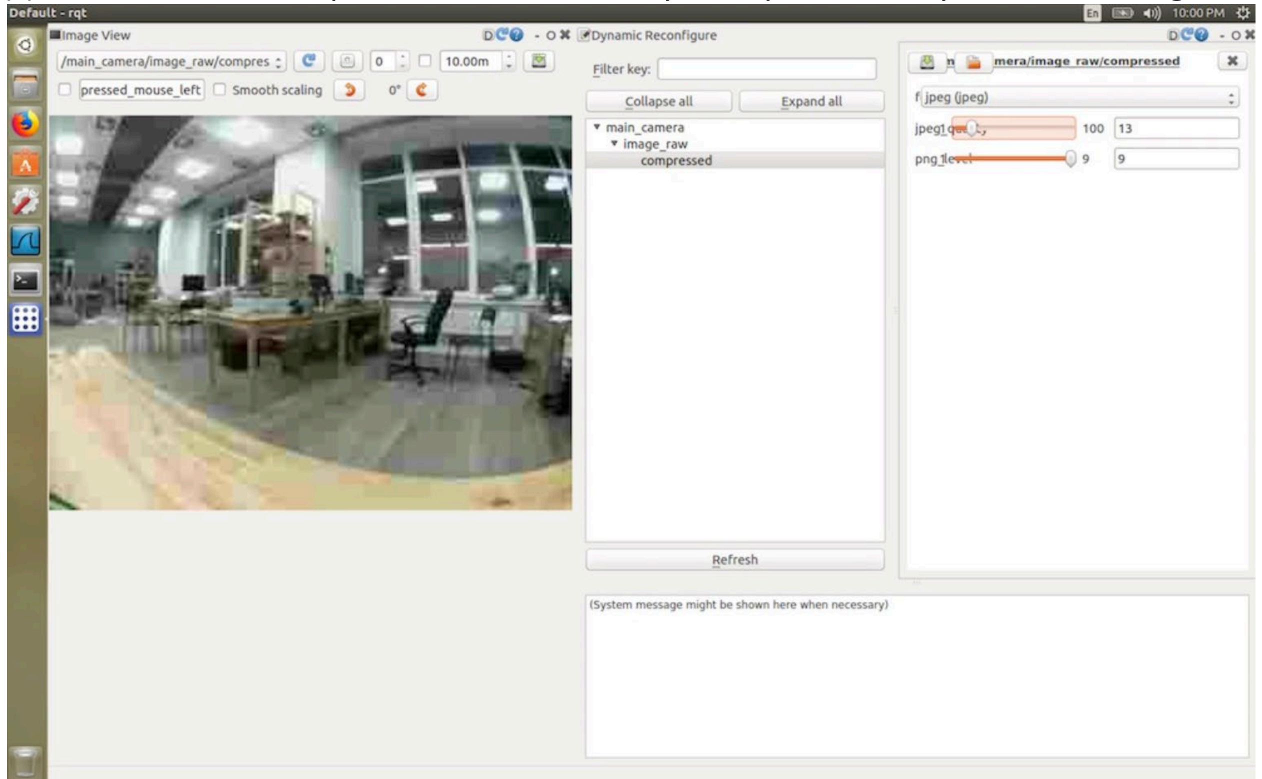
Выберите топик для просмотра, например, /main_camera/image_raw:

Результат визуализации коптера и камеры представлен ниже:



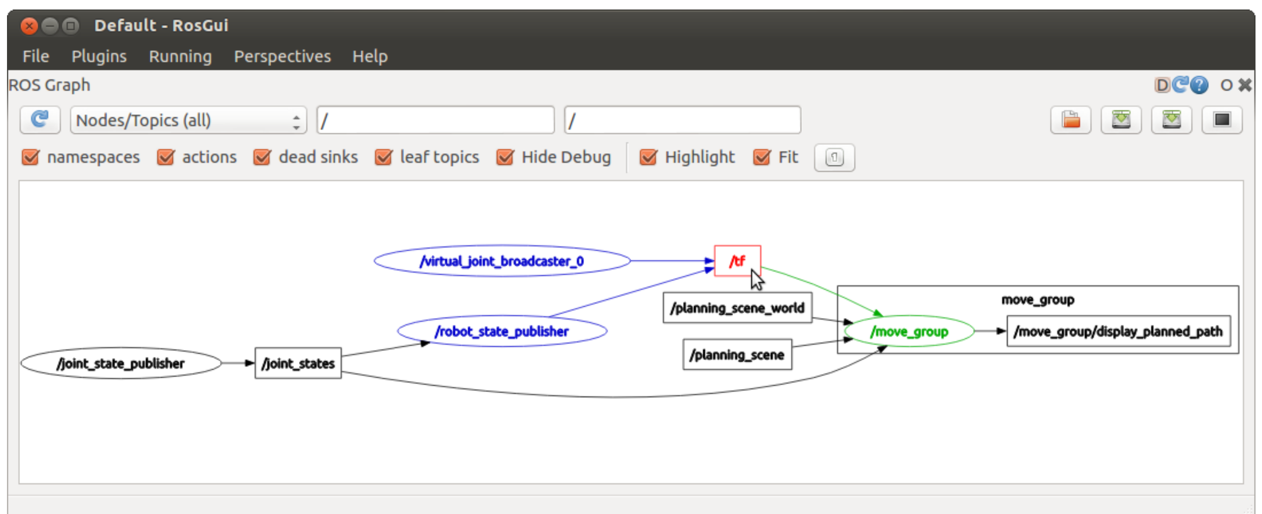
Для снижения нагрузки на сеть и уменьшения задержки используйте сжатый вариант топика – /camera/rgb/image_raw/compressed либо такой /usb_cam/image_raw/compressed.

Для изменения настроек сжатия используйте rqt-плагин Dynamic Reconfigure:



Rqt_graph

Утилита rqt_graph позволяет визуализировать взаимосвязи запущенных нод в системе ROS



Поставляемое ПО

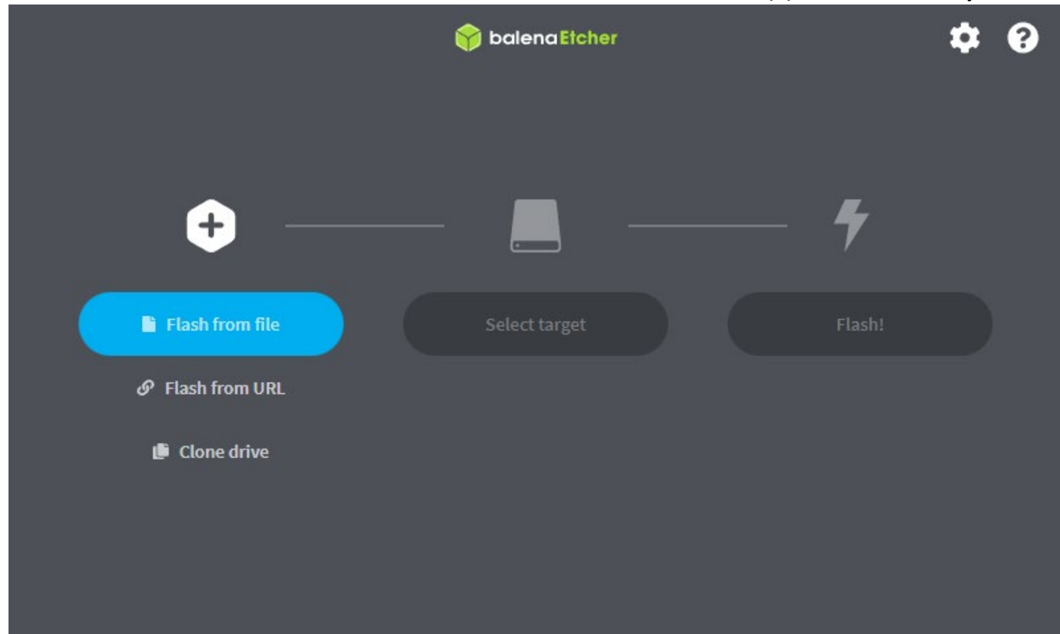
Для своего продукта мы используем модифицированный образ. Данный образ можно найти на флешке, которая входит в комплект поставки.

В комплект поставки входит SD - карта, вставленная в Raspberry pi, на которой уже установлен требуемый образ.

Перед тем как устанавливать образ его необходимо распаковать.

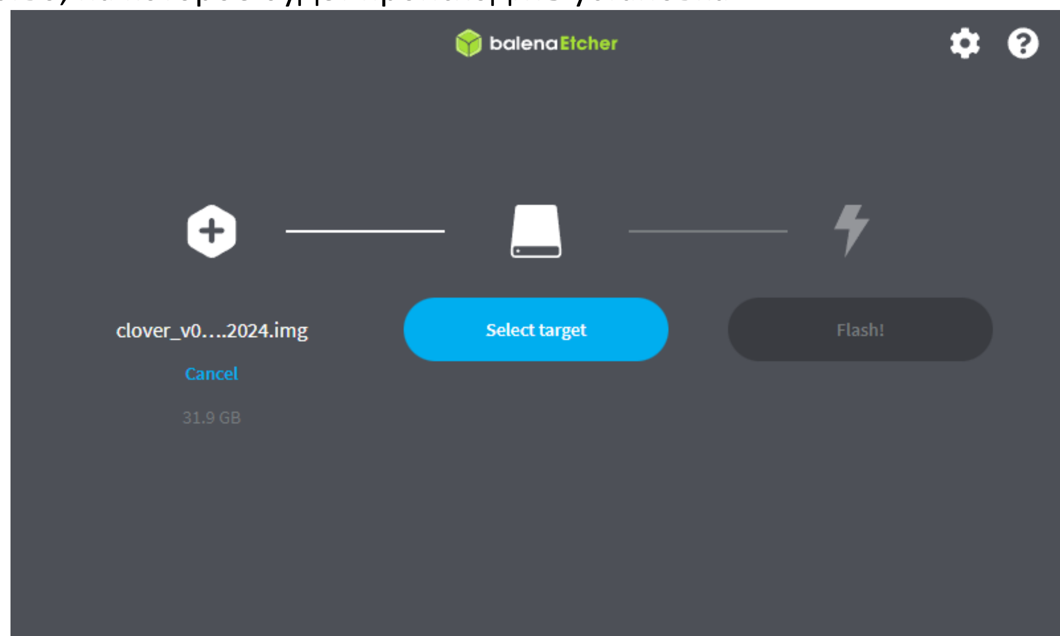
Для того чтобы установить образ на новую флеш-карту необходимо установить программу BalenaEtcher, которая есть на флешке с поставляемым ПО.

После того как вы скачали BalenaEtcher, вам необходимо ее запустить.

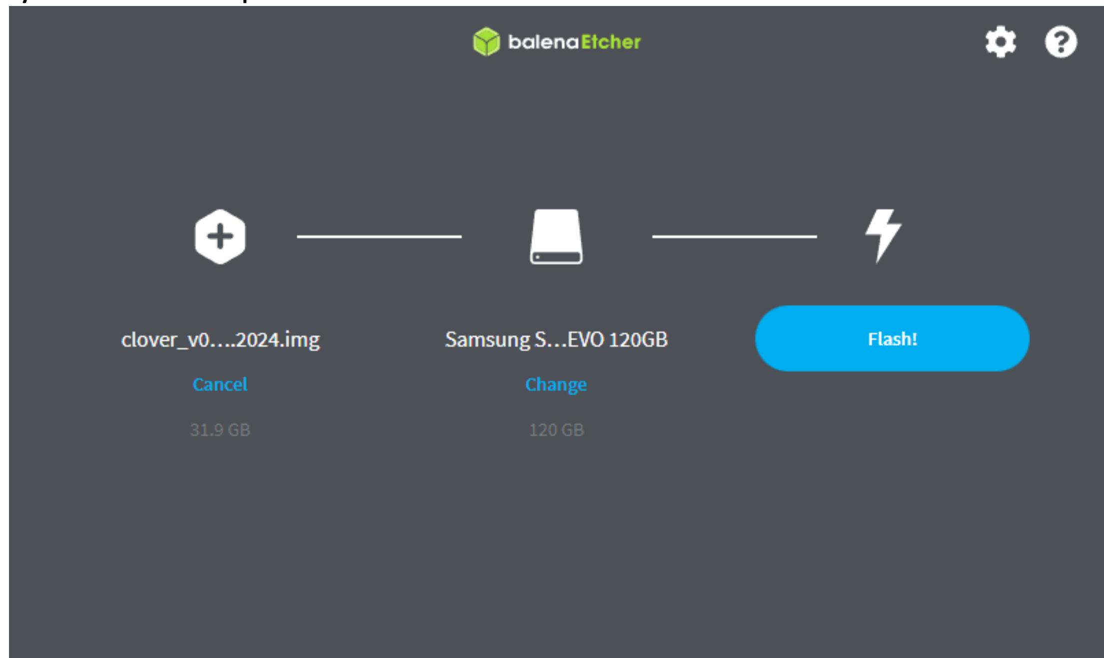


Нажмите кнопку Flash from file, после чего в открывшемся окне выберите нужный образ системы комплекса.

После того как вы выберете нужный образ вам необходимо будет выбрать устройство, на которое будет происходить установка



Нажмите кнопку select target, и выберите то устройство, на которое вы хотите установить образ



После этого нажмите кнопку Flash! И ожидайте конца установки. После завершения записи образа, флеш карту можно вставить в raspberry pi и начать работу.

Установка необходимых пакетов для ПК.

Установка .NET Runtime 6.0

- wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
- sudo dpkg -i packages-microsoft-prod.deb
- rm packages-microsoft-prod.deb
- sudo apt-get update
- sudo apt-get install -y dotnet-runtime-6.0

Директория ROS:

Директория находится по следующему пути:
/var/share/ProgramLab/ CarNavigation/Linux_M /ROS/

В папке catkin_ws/src/Test/ находится исходный код, используемый программным модулем.

В папке executables находятся файлы, используемые программным модулем для запуска ROS. Модифицировать их строго не рекомендуется.

Первый запуск

Перед первым запуском программы необходимо установить ROS и запустить сборку пакетов, используемых программным модулем. Для этого необходимо в директории var/share/ProgramLab/CarNavigation/Linux_M/ROS/ открыть терминал и выполнить следующие команды:

```
1  find ./ \( -type d -name .git -prune \) -o -type f -print0 | xargs -0 sed -i -e 's/\r$//'
2  sudo chmod +x setup.sh
3  ./setup.sh
```

LIDAR

Лида́р — лазерный локатор, использующий технологию испускания лазером волн оптического диапазона с дальнейшей регистрацией лазерных импульсов, которые были рассеяны объектами: лазерную (или оптико-электронную) локацию. Лазерная локация использует методы обнаружения и определения угловых координат объектов аналогичные используемым в радиолокации, однако имеет большую разрешающую способность и точность.

Устоявшийся перевод LIDAR как «лазерный радар» не вполне корректен, так как в системах ближнего радиуса действия (например, предназначенных для работы в помещениях), главные свойства лазера: когерентность, высокие плотность и мгновенная мощность излучения — не востребованы; излучателями света в таких системах могут служить обычные светодиоды. Однако в основных сферах применения технологии (метеорология, геодезия и картография) с радиусами действия от сотен метров до сотен километров используются только лазеры.

Используемые топики

- Напряжение на аккумуляторной батарее - /PowerVoltage
- Изображение с основной камеры /camera/rgb/image_raw
- Сжатое изображение с основной камеры - /camera/rgb/image_raw/compressed
- Управление движением комплекса - /cmd_vel
- /diagnostics
- /imu
- Топик с данными карты, построенной лидаром - /map
- Одометрия комплекса - /robot_pose_ekf/odom_combined
- Базовый топик ROS - /rosout
- Базовый топик ROS - /rosout_agg
- Данные лидара - /scan
- Набор векторов для определения перемещения - /tf
- Набор векторов для определения перемещения - /tf_static
- Изображение с дополнительной камеры - /usb_cam/image_raw
- Сжатое изображение с дополнительной камеры - /usb_cam/image_raw/compressed

Топики для Lidar

- Топик с данными лидара - /scan

Топики для просмотра камер:

- Изображение с основной камеры - /camera/compressed;
- Сжатое изображение с основной камеры - /camera/image_raw/compressed;
- Изображение с дополнительной камеры - /usb_cam/image_raw;
- Сжатое изображение с дополнительной камеры - /usb_cam/image_raw/compressed.

Разработка и эксплуатация собственного функционала

При работе с квадрокоптером вы можете использовать собственные ноды, написанные на языках программирования Python и C++. Создание собственного пакета ROS. Структура простейшего пакета обязательно должна содержать 2 файла:

- CmakeLists.txt
- Package.xml

  В свою очередь пакеты должны находиться в рабочем пространстве Catkin

```
workspace_folder/          -- WORKSPACE
  src/                     -- SOURCE SPACE
    CMakeLists.txt         -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt       -- CMakeLists.txt file for package_1
      package.xml          -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt       -- CMakeLists.txt file for package_n
      package.xml          -- Package manifest for package_n
```

Для того чтобы создать пакет используйте команду

```
1 catkin_create_pkg Имя_пакета std_msgs rospy roscpp actionlib_msgs
2 message_generation message_runtime
```

После создания пакета мы увидим директорию с тем названием, которое указали при создании. В этой директории будут находиться следующие файлы:

- include (Директория для Header файла языка Cpp)
- src (Директория исходных кодов)
- CmakeLists.txt (Файл конфигурации системы сборки)
- Package.xml (Файл конфигурации пакета)

Если вам необходимо добавить зависимости используйте команду:

- catkin_create_pkg <package_name> [depend1] [depend2] [depend3]

После завершения создания пакета, его необходимо собрать для этого используйте команду:

```
1 cd ~/catkin_ws
2 catkin_make
```

Для добавления созданных пакетов в окружение ROS необходимо запустить сгенерированный файл настройки:

```
1 ~/catkin_ws/devel/setup.bash
```

Использование msg и srv

- msg: файлы msg — это простые текстовые файлы, которые описывают поля сообщения ROS. Они используются для создания исходного кода сообщений на разных языках.
- srv: файл srv описывает службу. Состоит из двух частей: запроса и ответа.

Информация о msg

Msg файлы могут быть следующих типов:

```
1  int8, int16, int32, int64
2  float32, float64
3  string
4  time, duration
5  variable-length array
```

В ROS также есть специальный тип: Header. Содержит временную метку и информацию о координатах, которые обычно используются в ROS.

Определить в msg файле его можно по первой строке Header header.

Ниже приведен пример файла msg с использованием Header, примитивного string и двух других msg файлов:

```
1  Header header
2  string child_frame_id
3  geometry_msgs/PoseWithCovariance pose
4  geometry_msgs/TwistWithCovariance twist
```

Файлы srv аналогичны файлам msg, за исключением того, что они содержат две части: запрос и ответ. Две части разделены линией «---». Вот пример файла srv:

```
1      int64 A
2      int64 B
3      ---
4      int64 Sum
```

В приведенном выше примере A и B — это запрос, а Sum — ответ.

Создание нового msg

Давайте определим новое сообщение в пакете, созданном в предыдущей главе.

```
1      $ roscd beginner_tutorials
2      $ mkdir msg
3      $ echo "int64 num" > msg/Num.msg
```

Приведенный выше пример файла .msg содержит только 1 строку. Однако вы можете создать более сложный файл, добавив несколько элементов, по одному в строке, например:

```
1  string first_name
2  string last_name
3  uint8 age
4  uint32 score
```

Есть еще один шаг. Нам нужно убедиться, что файлы msg преобразованы в исходный код для C++, Python и других языков:

Откройте package.xml и убедитесь, что эти две строки находятся в нем и не закомментированы

```
1  <build_depend>message_generation</build_depend>
2  <exec_depend>message_runtime</exec_depend>
```

Обратите внимание, что во время сборки нам нужно «message_generation», а во время выполнения нам нужно только «message_runtime».

Откройте CMakeLists.txt в любом текстовом редакторе. Добавьте зависимость message_generation к find_package, который уже существует в CMakeLists.txt , чтобы вы могли генерировать сообщения. Вы можете сделать это, просто добавив message_generation в список COMPONENTS. Выглядит это следующим образом:

```
1  # Do not just add this to your CMakeLists.txt, modify the existing text to add
2  message_generation before the closing parenthesis
3  find_package(catkin REQUIRED COMPONENTS
4  roscpp
5  rospy
6  std_msgs
7  message_generation
8  )
```

Также убедитесь, что вы экспортировали зависимость времени выполнения сообщения.

```
1  catkin_package (
2    ...
3    CATKIN_DEPENDS message_runtime ...
4    ... )
```

Найдите следующий блок кода

```
1    # add_message_files(
2    #   FILES
3    #   Message1.msg
4    #   Message2.msg
5    # )
```

Раскомментируйте его, удалив символы #, а затем замените подставку в файлах Message*.msg своим файлом .msg , чтобы он выглядел следующим образом:

```
1    add_message_files (
2      FILES
3      Num.msg
4    )
```

Добавляя файлы .msg вручную, мы гарантируем, что CMake будет знать, когда ему придется переконфигурировать проект после добавления других файлов .msg.

Теперь мы должны убедиться, что вызывается функция generate_messages.

Для ROS Hydro и более поздних версий вам необходимо раскомментировать эти строки:

```
1    # generate_messages(
2    #   DEPENDENCIES
3    #   std_msgs
4    # )
5    Выглядеть это будет так:
6    generate_messages(
7    DEPENDENCIES
8    std_msgs
9    )
```

Теперь вы готовы генерировать исходные файлы на основе определения сообщения.

Использование rosmsg

Убедитесь, что ROS его видит с помощью команды rosmsg show.

Использование:

```
$ rosmsg show [message type]
```

Пример

```
$ rosmsg show beginner_tutorials/Num
```

Вывод:

int64 num

В предыдущем примере тип сообщения состоит из двух частей:

- beginner_tutorials – пакет, в котором содержится сообщение
- Num – Имя Msg Num.

Если вы не знаете, в каком пакете находится сообщение, вы можете не указывать имя пакета. Попробуйте:

$ rosmsg show Num

Вывод:

```
1 [beginner_tutorials/Num]:
2 int64 num
```

Создание srv

Давайте воспользуемся только что созданным пакетом для создания srv:

```
1 $ roscd beginner_tutorials
2 $ mkdir srv
```

Вместо того, чтобы вручную создавать новое определение srv, мы скопируем существующее из другого пакета.

Для этого roscp — полезный инструмент командной строки для копирования файлов из одного пакета в другой.

Использование:

```
1 $ roscp [package_name] [file_to_copy_path] [copy_path]
```

Теперь мы можем скопировать сервис из пакета rospy_tutorials :

```
1 $ roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
```

Далее нам нужно убедиться, что srv-файлы преобразованы в исходный код для C++, Python и других языков.

Если вы еще этого не сделали, откройте package.xml и убедитесь, что эти две строки находятся в нем и не закомментированы:

```
1 <build_depend>message_generation</build_depend>
2  <exec_depend>message_runtime</exec_depend>
```

Как и раньше, обратите внимание, что во время сборки нам нужно «message_generation», а во время выполнения нам нужно только «message_runtime».

Если вы еще не сделали это для сообщений на предыдущем шаге, добавьте зависимость message_generate для генерации сообщений в CMakeLists.txt :

```
1 # Do not just add this line to your CMakeLists.txt, modify the existing line
2 find_package(catkin REQUIRED COMPONENTS
3 roscpp
4 rospy
5 std_msgs
6 message_generation
7 )
```

(Несмотря на название, message_генерация работает как для msg , так и для srv .)

Также вам потребуются те же изменения в package.xml для служб, что и для сообщений, поэтому дополнительные необходимые зависимости смотрите выше.

Удалите # , чтобы раскомментировать следующие строки:

```
1 # add_service_files(
2 #   FILES
3 #   Service1.srv
4 #   Service2.srv
5 # )
```

И замените файлы-заполнители Service*.srv на свои служебные файлы:

```
1 add_service_files(
2 FILES
3 AddTwoInts.srv
4 )
```

Использование rossrv

Использование

```
1 $ rossrv show <service type>
```

Пример

```
1 $ rossrv show beginner_tutorials/AddTwoInts
2 Вывод:
3 int64 a
4 int64 b
5 ---
6 int64 sum
```

Как и в случае с rosmsg , вы можете найти такие служебные файлы без указания имени пакета:

```
1 $ rossrv show AddTwoInts
2 [beginner_tutorials/AddTwoInts]:
3 int64 a
4 int64 b
5 ---
6 int64 sum
```

Общее для msg и srv

Если вы еще не сделали это ранее, измените CMakeLists.txt:

```
1 # generate_messages(
2 #   DEPENDENCIES
3 # #  std_msgs  # Or other packages containing msgs
4 # )
```

Раскомментируйте его и добавьте все пакеты, от которых вы зависите и которые содержат файлы .msg , используемые вашими сообщениями (в данном случае std_msgs), чтобы они выглядели следующим образом:

```
1 generate_messages(
2 DEPENDENCIES
3 std_msgs
4 )
```

Теперь, когда мы создали несколько новых сообщений, нам нужно снова создать пакет:

```
1 # In your catkin workspace
2 $ roscd beginner_tutorials
3 $ cd ../..
4 $ catkin_make
5 $ cd -
```

Альтернативой системе catkin_make является использование сборки catkin.

```
1 # In your catkin workspace
2 $ roscd beginner_tutorials
3 $ cd ../..
4 $ catkin build
5  $ cd –
```

Любой файл .msg в каталоге msg будет генерировать код для использования на всех поддерживаемых языках. Файл заголовка сообщения C++ будет создан в ~/catkin_ws/devel/include/beginner_tutorials/ . Сценарий Python будет создан в ~/catkin_ws/devel/lib/python2.7/dist-packages/beginner_tutorials/msg . Файл Lisp находится в папке ~/catkin_ws/devel/share/common-lisp/ros/beginner_tutorials/msg/ .

Аналогично, любые файлы .srv в каталоге srv будут содержать код на поддерживаемых языках. Для C++ это создаст файлы заголовков в том же каталоге, что и файлы заголовков сообщений. Для Python и Lisp рядом с папками «msg» будет папка «srv».

Создание Publisher ноды

"Нода" – это термин ROS для обозначения исполняемого файла, подключенного к сети ROS. Здесь мы создадим ноду publisher, которая будет постоянно транслировать сообщение.
Для создания ноды необходимо перейти в каталог: $ cd ~/catkin_ws/src
Далее создайте пакет, используя команду
$ catkin_create_pkg Имя_пакета std_msgs rospy roscpp

Создайте ноду со следующим кодом:

```
1  #!/usr/bin/env python
2  # license removed for brevity
3  import rospy
4  from std_msgs.msg import String
5  def talker():
6     pub = rospy.Publisher('chatter', String, queue_size=10)
7     rospy.init_node('talker', anonymous=True)
8     rate = rospy.Rate(10) # 10hz
9     while not rospy.is_shutdown():
10        hello_str = "hello world %s" % rospy.get_time()
11        rospy.loginfo(hello_str)
12        pub.publish(hello_str)
13        rate.sleep()
14 if __name__ == '__main__':
15    try:
16       talker()
17    except rospy.ROSInterruptException:
18       pass
19
```

Для правильной установки сценария Python и использования правильного преобразователя, добавьте в файл CMakeLists.txt следующее:

```
1     catkin_install_python(PROGRAMS scripts/talker.py
2       DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
3     )
```

Объяснение кода

Теперь давайте разберем код.

1 #!/usr/bin/env python

В каждой ноде Python ROS будет такой первая строка. Первая строка гарантирует, что ваш скрипт выполняется как скрипт Python.

3 import rospy

4 from std_msgs.msg import String

Вам необходимо импортировать rospy, если вы пишете ноду ROS. Импорт std_msgs.msg позволяет повторно использовать тип сообщения std_msgs/String для публикации.

```
1  pub = rospy.Publisher  ('chatter',  String, queue_size=10)
2  rospy.init_node  ('talker',  anonymous=True)
```

Этот раздел кода определяет интерфейс говорящего с остальной частью ROS.
pub = rospy.Publisher ("chatter", String,queue_size=10) говорит о том, что ваша нода публикует информацию в теме chatter, используя тип сообщения String. String здесь на самом деле является классом std_msgs.msg.String . Queue_size является **Новинкой в ROS Hydro** и ограничивает количество сообщений в очереди, если какой-либо подписчик не получает их достаточно быстро. В старых дистрибутивах ROS его просто опускают.

Следующая строка, rospy.init_node (NAME, ...) очень важна, поскольку она сообщает rospy имя вашего узла — пока rospy не получит эту информацию, он не сможет начать связь с ROS Master . В этом случае ваша нода получит имя talker. ПРИМЕЧАНИЕ: имя должно быть базовым, т. е. оно не может содержать косую черту «/».

anonymous=True сообщает, что ваша нода имеет уникальное имя, добавляя случайные числа в конец NAME .

1	9 rate = rospy.Rate(10) # 10hz

Эта строка создает объект Rate. С помощью метода Sleep ( ) он предлагает удобный способ зацикливания с нужной скоростью. С аргументом 10 мы ожидаем, что цикл будет проходить 10 раз в секунду.

1	while not rospy.is_shutdown():
2	hello_str = "hello world %s" % rospy.get_time()
3	rospy.loginfo(hello_str)
4	pub.publish(hello_str)
5	rate.sleep()

Этот цикл представляет собой довольно стандартную конструкцию rospy: проверка rospy.is_shutdown() и последующее выполнение работы. Вам нужно проверить is_shutdown(), должна ли ваша программа завершить работу (например, если есть Ctrl-C). В данном случае «работой» является вызов pub.publish(hello_str) , который публикует строку в нашей теме беседы . Цикл вызывает функцию rate.sleep() , которая бездействует ровно столько, сколько необходимо для поддержания желаемой скорости в цикле.

(Вы также можете столкнуться с rospy.sleep() , который похож на time.sleep(), за исключением того, что он также работает с моделируемым временем (см. Clock).)

Этот цикл также вызывает rospy.loginfo(str) , который выполняет тройную задачу: сообщения выводятся на экран, записываются в файл журнала узла и записываются в rosout . rosout — удобный инструмент для отладки: вы

можете получать сообщения с помощью rqt_console вместо того, чтобы искать окно консоли с выводом вашего узла.

std_msgs.msg.String — это очень простой тип сообщений, поэтому вам может быть интересно, как выглядит публикация более сложных типов. Общее практическое правило заключается в том, что *аргументы конструктора располагаются в том же порядке, что и в файле .msg* . Вы также можете не передавать аргументы и инициализировать поля напрямую, например

```
1  msg = String()
2  msg.data = str
```

или вы можете присвоить значения в некоторых полях, а остальные оставить со значениями по умолчанию:

```
1  String (data=str)
```

Рассмотрим также конечную часть:

```
1  try:
2  talker()
3  except rospy.ROSInterruptException:
4  pass
```

В дополнение к стандартной проверке __main__ Python при этом перехватывается исключение rospy.ROSInterruptException , которое может быть вызвано методами rospy.sleep() и rospy.Rate.sleep() при нажатии Ctrl-C или при выключении ноды по иным причинам. Причина, по которой возникает это исключение, заключается в том, что вы случайно не продолжаете выполнение кода после Sleep () .

Написание Subscriber ноды

Код

```
1  $ roscd beginner_tutorials/scripts/
2  $ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/
3  rospy_tutorials/001_talker_listener/listener.py
4   $ chmod +x listener.py
```

Содержимое файла выглядит примерно так:

```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String
4  def callback(data):
5  rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
6  def listener():
7  # In ROS, nodes are uniquely named. If two nodes with the same
8  # name are launched, the previous one is kicked off. The
9  # anonymous=True flag means that rospy will choose a unique
10 # name for our 'listener' node so that multiple listeners can
11 # run simultaneously.
12 rospy.init_node('listener', anonymous=True)
13 rospy.Subscriber("chatter", String, callback)
14 # spin() simply keeps python from exiting until this node is stopped
15 rospy.spin()
16 if __name__ == '__main__':
17 listener()
18
```

Затем отредактируйте catkin_install_python() в файле CMakeLists.txt , чтобы он выглядел следующим образом:

```
1  catkin_install_python(PROGRAMS scripts/talker.py scripts/listener.py
2   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
3       )
```

Объяснение кода

Код Listener.py аналогичен коду talker.py , за исключением того, что введен новый механизм подписки на сообщения на основе обратного вызова.

```
1 rospy.init_node('listener', anonymous=True)
2 rospy.Subscriber("chatter", String, callback)
3 # spin() simply keeps python from exiting until this node is stopped
4 rospy.spin()
```

Это означает, что ваша нода подписывается на тему обмена сообщениями типа std_msgs.msgs.String . При получении новых сообщений вызывается обратный вызов с сообщением в качестве первого аргумента.

Мы также несколько изменили вызов rospy.init_node() . Мы добавили аргумент ключевого слова anonymous=True . ROS требует, чтобы каждая нода имела уникальное имя. Если появляется нода с таким же именем, она отталкивает предыдущую. Это сделано для того, чтобы неисправные узлы можно было легко отключить из сети.
Флаг anonymous=True указывает rospy сгенерировать уникальное имя для ноды, чтобы можно было легко запускать несколько нод Listener.py .
Последнее дополнение: rospy.spin() просто не позволяет вашей ноде выйти до тех пор, пока нода не будет выключена. В отличие от roscpp, rospy.spin() не влияет на функции обратного вызова подписчика, поскольку они имеют свои собственные потоки.

Создание нод

Мы используем CMake в качестве системы сборки, и да, вам придется использовать ее даже для узлов Python. Это делается для того, чтобы убедиться, что создан автоматически сгенерированный код Python для сообщений и служб.

Перейдите в рабочую область Catkin и запустите catkin_make :

```
1 $ cd ~/catkin_ws
2 $ catkin_make
```

Запуск publisher

Убедитесь, что roscore запущен и работает:

$ roscore

catkin specific . Если вы используете catkin, убедитесь, что вы создали файл setup.sh вашей рабочей области после вызова catkin_make , но прежде чем пытаться использовать свои приложения:

```
1    # In your catkin workspace
2    $ cd ~/catkin_ws
3    $ source ./devel/setup.bash
```

В прошлом уроке мы создали издатель под названием «Talker». Давайте запустим:

```
1    $ rosrun beginner_tutorials talker      (C++)
2    $ rosrun beginner_tutorials talker.py   (Python)
```

$ rosrun beginner_tutorials talker (C++)
$ rosrun beginner_tutorials talker.py (Python)

Вы увидите что-то похожее на:

```
1  [INFO] [WallTime: 1314931831.774057] hello world 1314931831.77
2  [INFO] [WallTime: 1314931832.775497] hello world 1314931832.77
3  [INFO] [WallTime: 1314931833.778937] hello world 1314931833.78
4  [INFO] [WallTime: 1314931834.782059] hello world 1314931834.78
5  [INFO] [WallTime: 1314931835.784853] hello world 1314931835.78
6  [INFO] [WallTime: 1314931836.788106] hello world 1314931836.79
```

Publisher нода запущена и работает. Теперь нам нужен subscriber для получения сообщений от издателя.

Запуск subscriber

В прошлом уроке мы создали подписчика под названием «слушатель». Давайте запустим:

Когда вы закончите, нажмите Ctrl-C, чтобы завершить работу как слушателя, так и говорящего.

```
1   osrun beginner_tutorials listener     (C++)
2   rosrun beginner_tutorials listener.py  (Python)
3   Вы увидите что-то похожее на:
4   [INFO] [WallTime: 1314931969.258941] /listener_17657_1314931968795I
5   heard hello world 1314931969.26
6   [INFO] [WallTime: 1314931970.262246] /listener_17657_1314931968795I
8   heard hello world 1314931970.26
9   [INFO] [WallTime: 1314931971.266348] /listener_17657_1314931968795I
10  heard hello world 1314931971.26
11  [INFO] [WallTime: 1314931972.270429] /listener_17657_1314931968795I
12  heard hello world 1314931972.27
13  [INFO] [WallTime: 1314931973.274382] /listener_17657_1314931968795I
14  heard hello world 1314931973.27
15
```

```
16 [INFO] [WallTime: 1314931974.277694] /listener_17657_13149319687951
17 heard hello world 1314931974.28
18 [INFO] [WallTime: 1314931975.283708] /listener_17657_13149319687951
   heard hello world 1314931975.28
```

Инструкция по установке и запуску проекта

1. Распакуйте, соберите и подключите к сети компьютер.
2. Установите **«PLCore»**.

Модуль запуска программных комплексов PLCore предназначен для запуска, обновления и активации программных комплексов, поставляемых компанией «Програмлаб».

В случае поставки программного комплекса вместе с персональным компьютером модуль запуска PLCore устанавливается на компьютер перед отправкой заказчику.

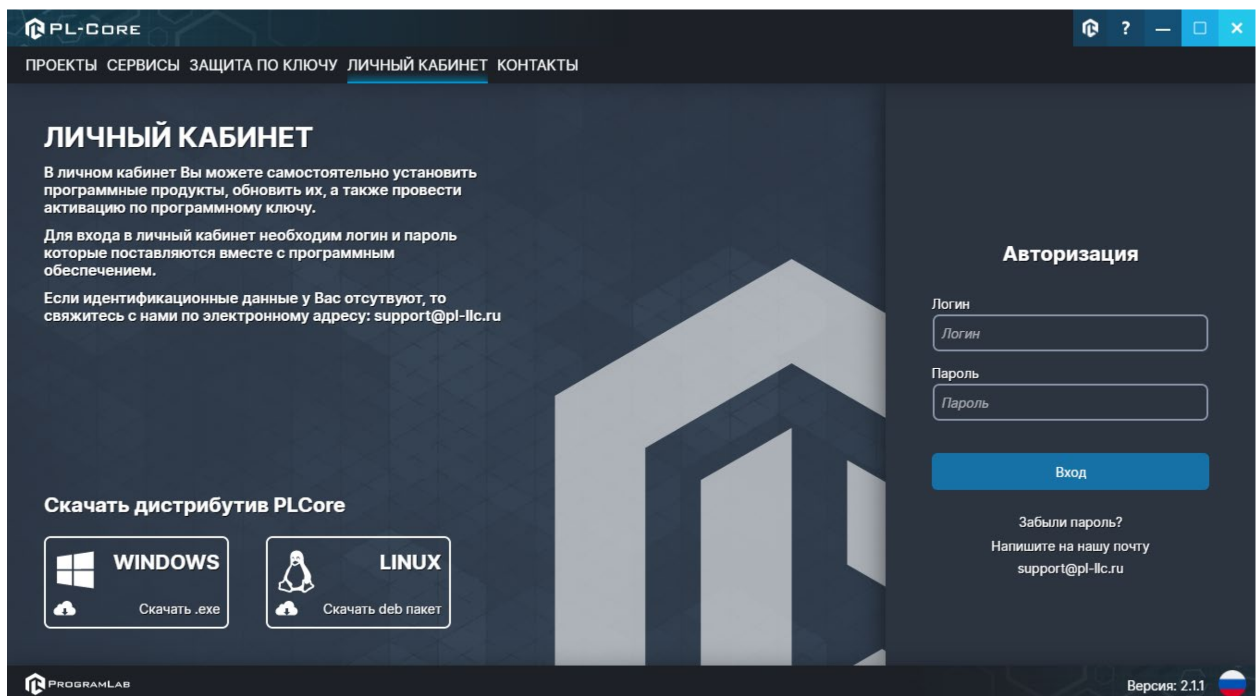
В случае поставки программного комплекса без ПК вам необходимо установить программное обеспечение с USB-носителя.

Перед установкой программного обеспечения установите модуль запуска учебных комплексов PLCORE. Для этого запустите файл с названием вида **PLCoreSetup_vX.X.X** на USB-носителе (Значения после буквы v в названии файла обозначают текущую версию ПО) и следуйте инструкциям.

3. Войдите в личный кабинет «PLCore».

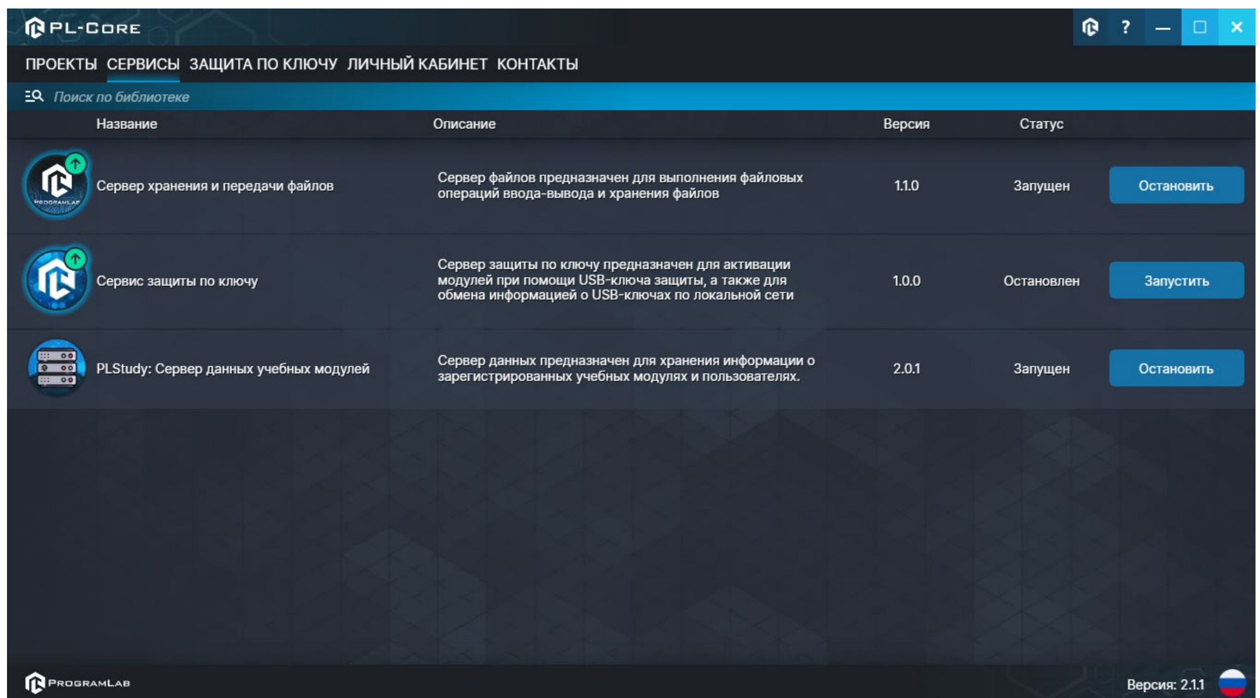
В комплект поставки входит **конверт с идентификационными данными для личного кабинета**. Если конверта нет, то напишите нам на почту *support@pl-llc.ru*.

Во вкладке «Личный кабинет» располагается окно авторизации по уникальному логину и паролю. После прохождения авторизации в личном кабинете представляется информация о доступных программных модулях (описание, состояние лицензии, информация о версиях), с возможностями их удаленной загрузки, обновления и активации по сети интернет.



Вход в личный кабинет «PLCore»

4. Активируйте проект следуя руководству пользователя **«PLCore»**.
5. Если ваш стенд предполагает автоматическую отправку результатов, установите **«PLStudy»** – программный комплекс, состоящий из двух модулей:
   - Сервис **«PLStudy: Сервер данных учебных модулей»**
   - Программный модуль **«PLStudy: Администрирование»**



Вкладка «Сервисы» с установленными и запущенными Сервером хранения и передачи файлов и PLStudy: Сервер данных учебных модулей

Установите сервер данных учебных модулей, если он ещё не установлен, на компьютер, который будет являться сервером. Для этого воспользуйтесь руководством пользователя **«PLStudy: Сервер данных учебных модулей»**. Для управления базой данных студентов и их результатов для всех комплексов нашей компании сразу можно воспользоваться модулем **«PLStudy: Администрирование»**.

По умолчанию в системе создается пользователь с именем Администратор и ролью Администратор. Этот пользователь не может быть удален, но его параметры могут быть изменены.

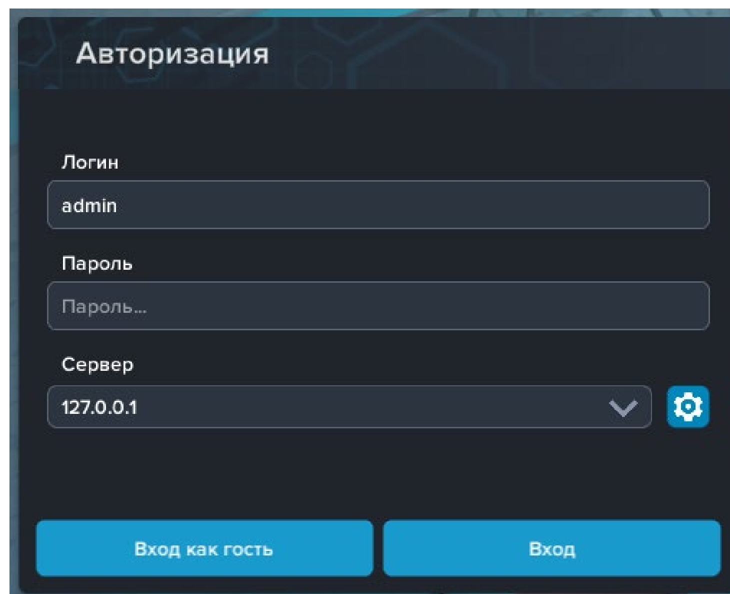
По умолчанию логин администратора: admin; Пароль: admin.

6. Для некоторых проектов необходим сервис «**Сервер хранения и передачи файлов»**. Сервер необходим для сохранения и загрузки с него файлов большого объема. Например, отчетов о прохождении тестирования в формате PDF.

7. Запустите проект.

Перед входом программа запросит логин, пароль. Здесь необходимо ввести параметры администратора или созданного на сервере пользователя. При авторизации в поле «Сервер» должен быть указан IP-адрес компьютера, на котором установлен сервис **«PLStudy: Сервер данных учебных модулей»**.

Чтобы изменить IP-адрес см. пункт «Запуск и управление в модуле» в руководстве пользователя **«PLStudy: Сервер данных учебных модулей»**.



Окно авторизации

Начало работы с комплексом

Аппарат питается от аккумуляторной батареи типа Li-Po, состоящей из 3 ячеек (3S), номинальным напряжением 12.6 В.

Аккумуляторная батарея подключенная к стандартному и балансировачному разъему

1. Разъём для подключения аккумуляторной батареи к аппаратному комплексу.
2. Балансировочный разъем (для зарядки).

Перед началом работы с комплексом убедитесь, что аккумуляторная батарея заряжена. Сделать это можно при помощи зарядного устройства, который входит в комплект.

Присоедините АКБ к зарядному устройству согласно следующим инструкциям, предварительно подсоединив зарядное устройство к питанию.

Подключение АКБ к балансировочнму разъему

Далее с помощью кнопок **1** и **2** необходимо выбрать программу. На кнопку 2 листаем программы вперёд до программы: **Lipo BATT**, затем нажимаем кнопку **4** Start.

Начало зарядки

После выбора режима зарядки литий-полимерной батареи, на экране прибора отобразиться новая информация.

Зарядка литий-полимерной батареи

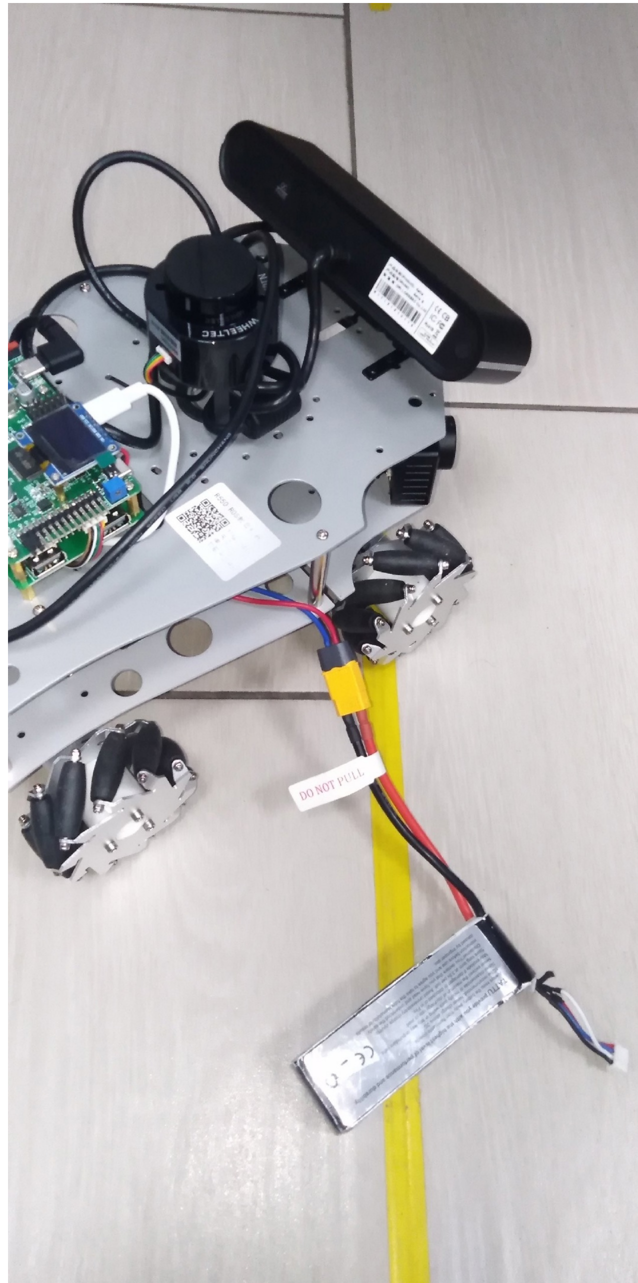
1. – Тип выбранной батареи
2. – Действие производимое с батареей
3. – Подаваемый ток в амперах
4. – Напряжение в батарейке в вольтах
5. – Количество ячеек

Далее с помощью кнопок **2** и **3**, отрегулируйте значение на 11.1 вольта и 3 ячейки для зарядки АКБ идущего в комплекте.

После регулировки ампер и напряжения, зажимаем кнопку **4** start, дожидаемся проверки и повторно нажимаем кнопку **4** start, после этого необходимо нажать кнопку **3** для равномерной зарядки батарей в АКБ.

Готово! АКБ будет заряжена, когда на экране прибора будет написано FULL.

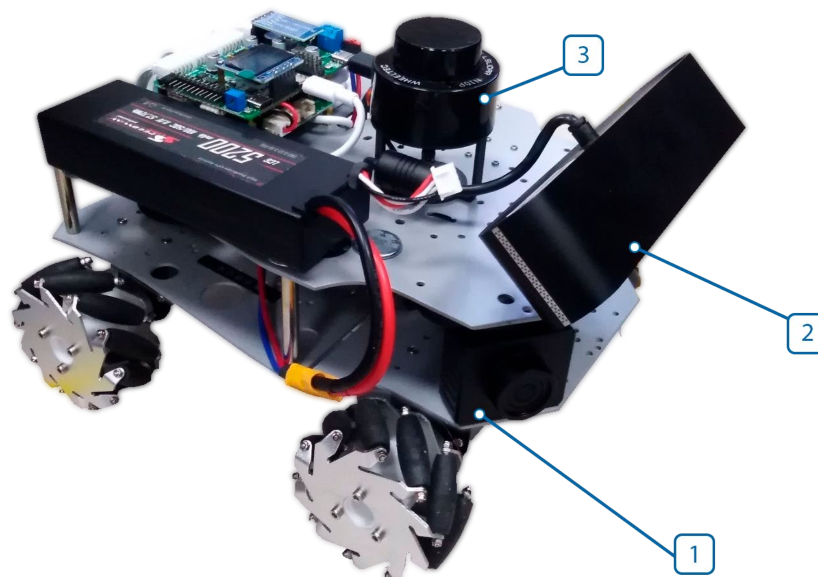
После того как вы убедились, что аккумулятор заряжен и находится в рамках рабочих напряжений, его можно подключить к аппарату.



Подключенный АКБ

После подключения аккумулятора включаем комплекс, для этого необходимо включить.

1. Переключатель, вкл./выкл. двигатели
2. Переключатель, вкл./выкл. машину



1) Вспомогательная камера, для распознавания знаков дорожного движения
2) Основная камера комплекса
3) Lidar

В комплекте идет 5 комплектов дорожных знаков, включающих в себя 4 знака (Stop, ограничение 20, ограничение 40, пешеходный переход) каждый. Комплекс содержит трассы разных размеров, 2 на 2 метра, 3 на 3 метра и 4 на 4 метра.

Дорожные знаки

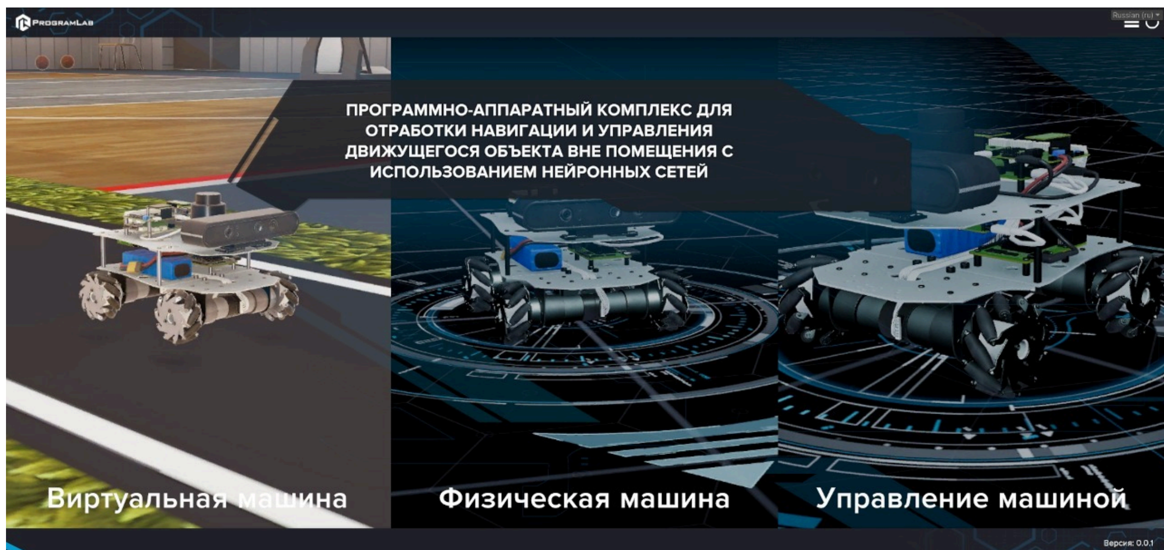
Трасса 2 на 2 метра

Трасса 3 на 3 метра

Трасса 4 на 4

Режимы работы ПО

В нашем ПО реализована поддержка работы как с виртуальным аппаратным комплексом, так и поддержка работы с физическим комплексом.



Меню работы с виртуальным аппаратным комплексом

Для начала работы с виртуальной машиной, выберете в главном меню **Виртуальная машина**.

Для начала работы с физической машиной выберете **Физическая машина** соответственно.

Режим управление машиной необходим для самостоятельного управления аппаратным комплексом.

В меню работы с ПО вы увидите Запуск на заранее подготовленной локации, редактор подготовленных локаций и кнопку настройки.

Нажмите кнопку запуск для того, чтобы перейти к выбору локации

Экран выбора локаций

После выбора локации начнется загрузка, после которой вы попадете на локацию. В ПО аппаратный комплекс движется с помощью геймпада либо автоматически.

Работа с комплексом в режиме редактора карт

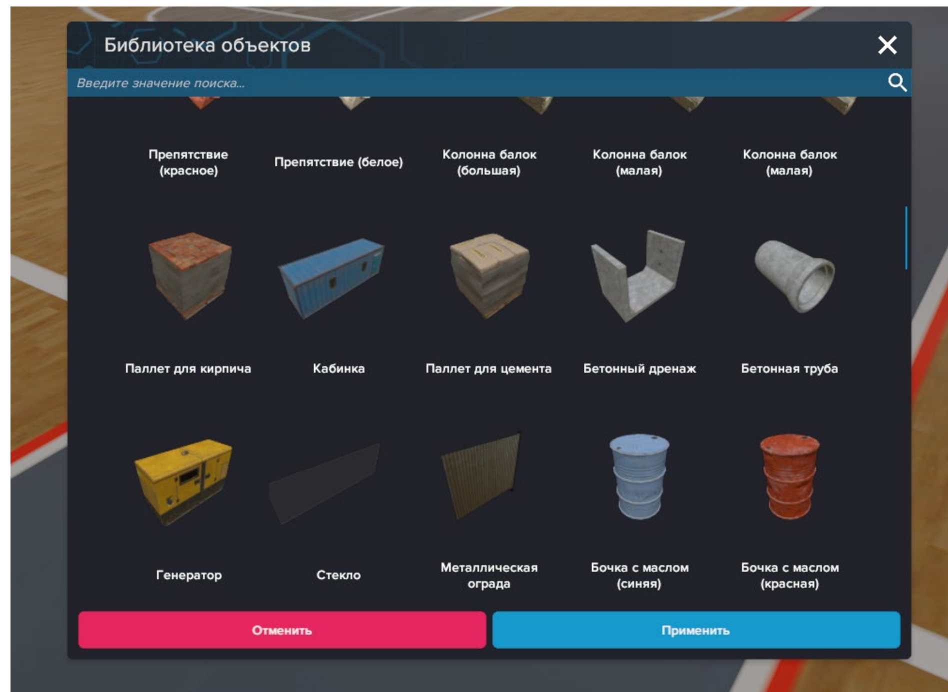
После загрузки в редактор карт вы уидите следующий интерфейс

Интерфейс редактора карт

1- Окно работы с объектами
2- Кнопка «Свернуть окно работы с объектами»
3- Панель с инструментами для работы с объектами

4- Кнопка свернуть меню с информацией о (создаваемом) проекте
5- Окно с информацией о (создаваемом) проекте

Для того чтобы начать работу необходимо нажать кнопку добавить объект, которая находится в панели 3, после ее нажатия вы попадете в библиотеку объектов.



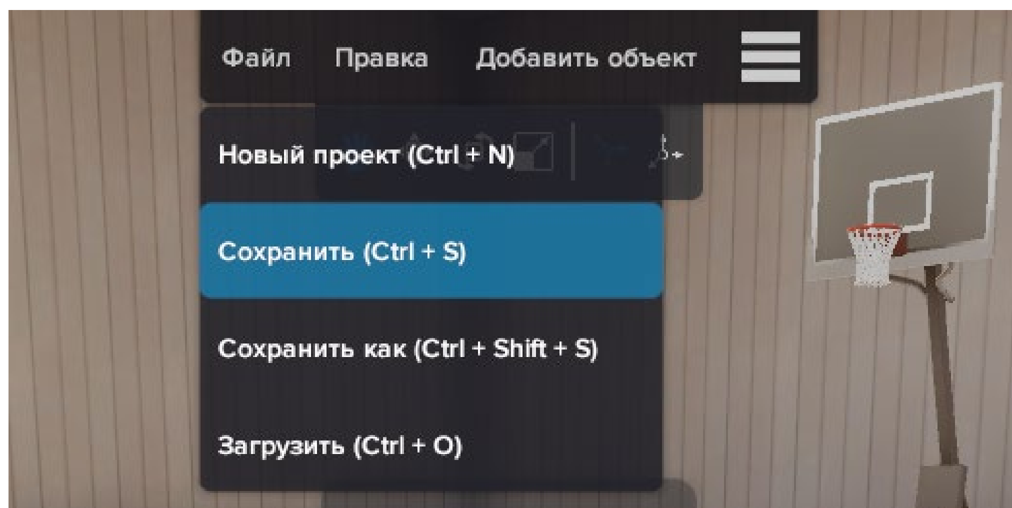
Библиотека объектов

Для того чтобы разместить объект на сцену, необходимо выбрать нужный объект и нажать кнопку применить, после чего объект появится на сцене

Работа с объектом в редакторе

1- Окно со списком объектов
2- Режим перемещения по сцене
3- Режим перемещения объекта на сцене
4- Режим поворота объекта
5- Режим масштабирования объекта
6- Режимы камеры
7- Окно свойств объекта

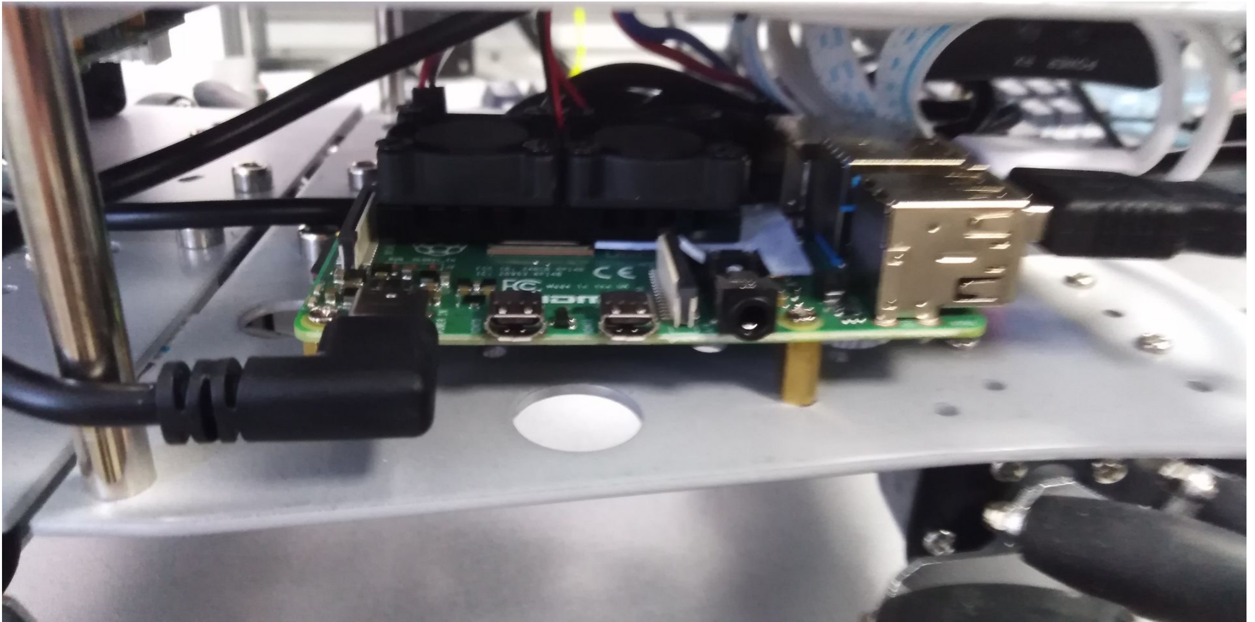
Для того чтобы сохранить измененную сцену необходимо нажать кнопку файл в панели инструментов и кнопку сохранить.



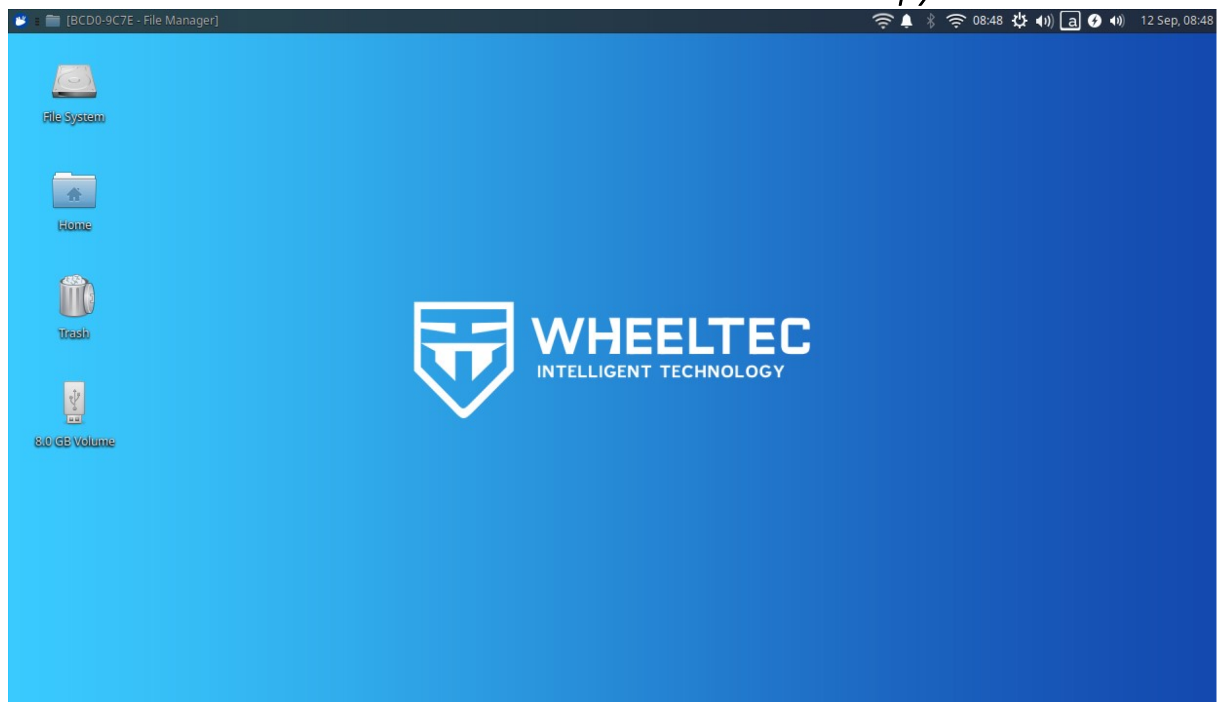
Далее сохраненные файлы сцены можно загружать и изменять.

Подключение к аппаратному комплексу через кабель HDMI.

Аппаратный комплекс имеет свою операционную систему. Для ее визуализации подключите кабель монитора к разъему аппаратного комплекса.

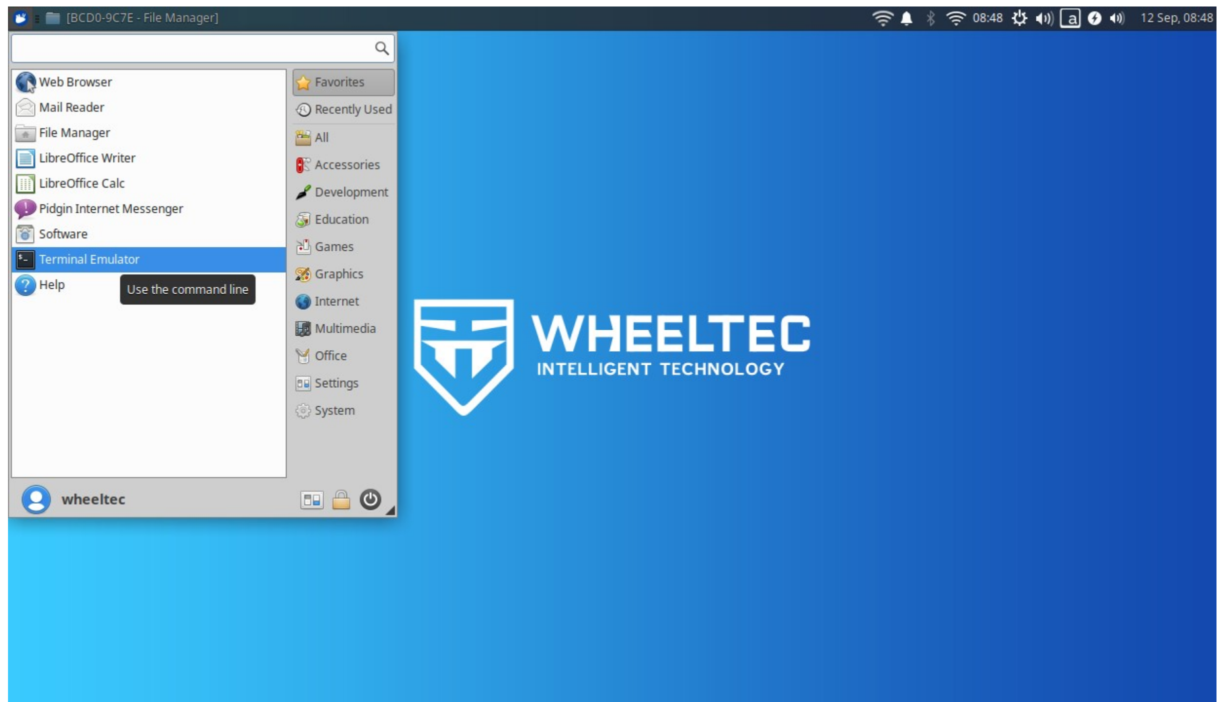


micro USB для подключения к монитору



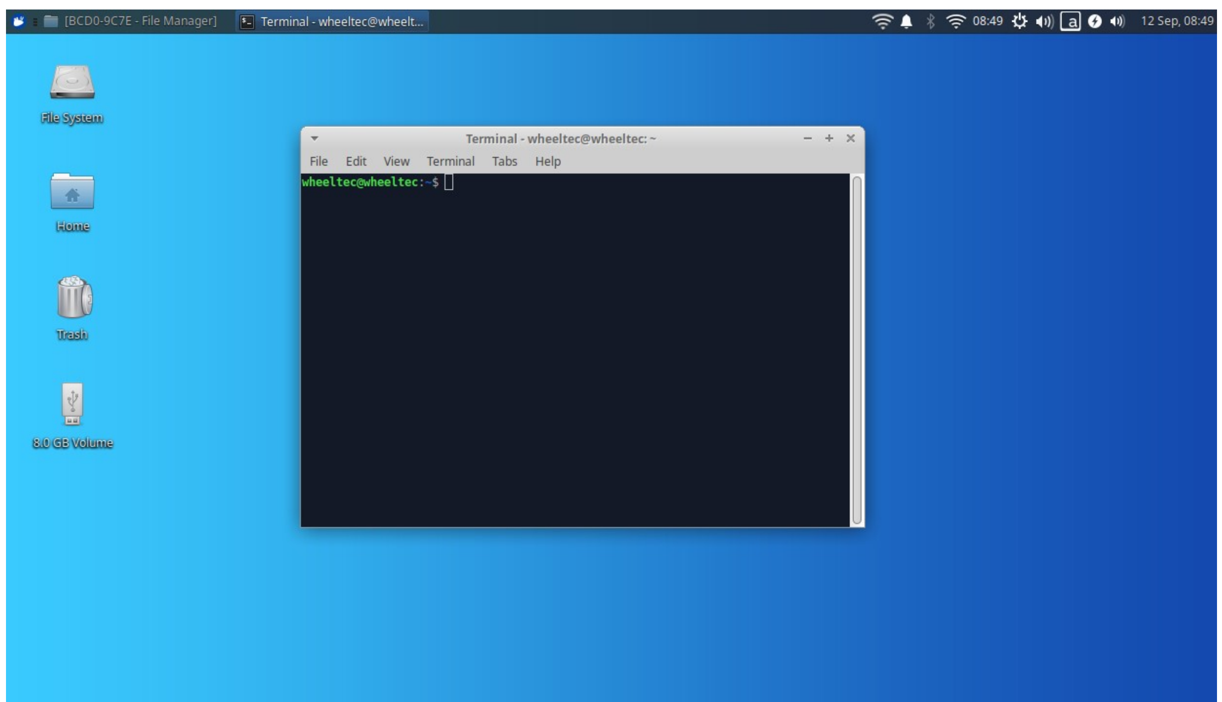
Экран операционной системы аппаратного комплекса

Нажмите на кнопку пуска в левой верхней части экрана. Выберите программу **Terminal Emulator**.



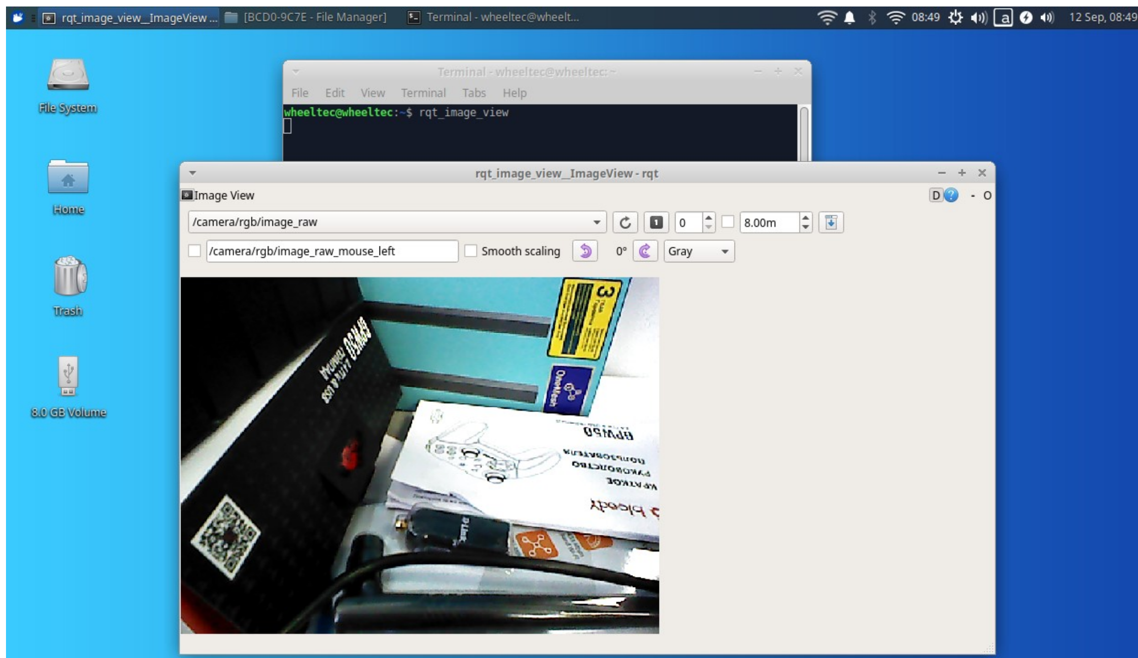
На экране откроется консольная панель. С помощью нее вы можете вносить изменения и управлять аппаратным комплексом в зависимости от требований.

Для некоторых консольных команд потребуется ввести пароль указанный в отдельном прикреплённом к поставке документу.



Консольная панель ОС

В консоли с помощью различных команд вы можете взаимодействовать с комплексом, например отобразить камеру.



Вызов камеры через консоль

Подключение к аппаратному комплексу через SSH и Putty.

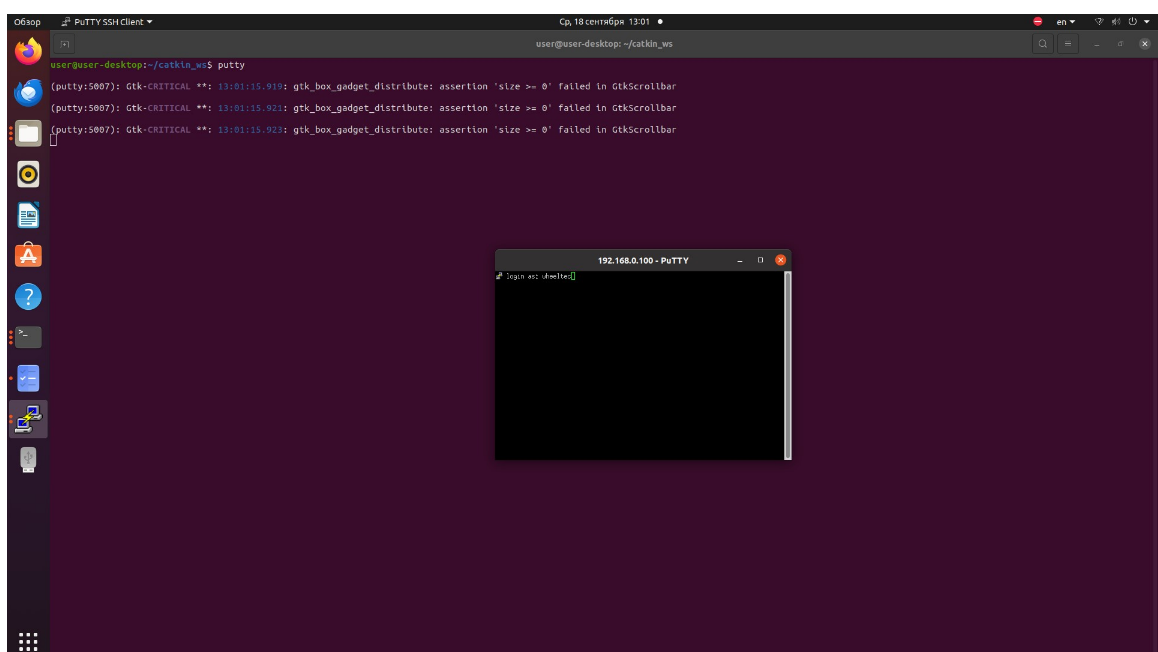
На компьютере, входящим в комплект поставки, установлена операционная система Linux.

Если к компьютеру подключен Wifi-роутер, идущий в комплекте поставки, он автоматически будет подключен к аппаратному комплексу.

Для воспроизведения предыдущих действий на операционной системе Linux откройте консольный терминал и введите в нем команду ***putty*** для подключения к аппаратному комплексу.

Терминал можно найти в главном меню операционной системе.

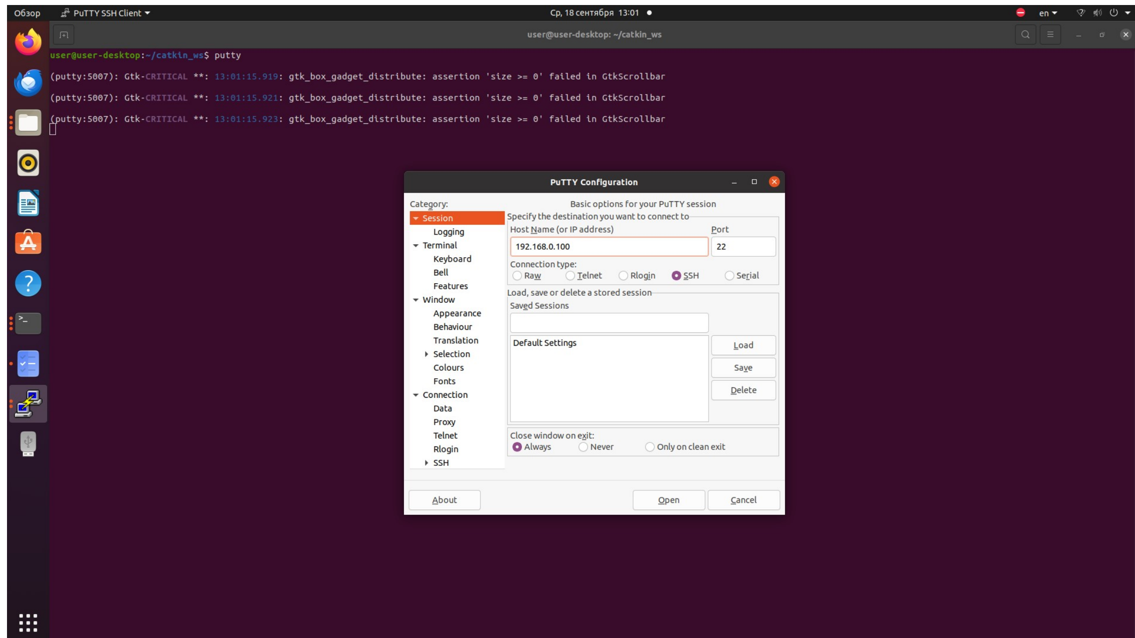
Откроется окно подключения:



Окно подключения putty

В этом окне необходимо ввести логин и пароль аппаратного комплекса.

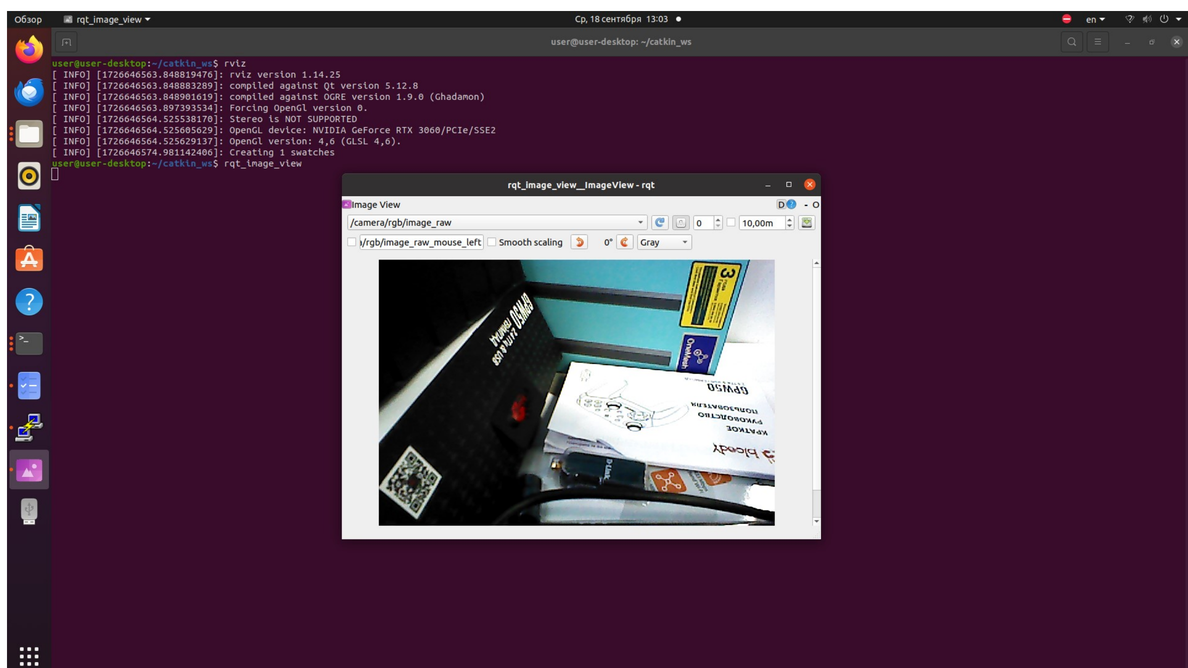
Далее откроется окно конфигурации. В этом окне конфигурации вы можете эмулировать операционную систему аппаратного комплекса с помощью SSH. Для этого введите в host name IP комплекса - **192.168.0.100,** выберите тип подключения **SSH** и нажмите **Open**. После этого откроется терминал, в котором вы можете выполнять те же самые действия, что и с прямым подключением к аппаратному комплексу.



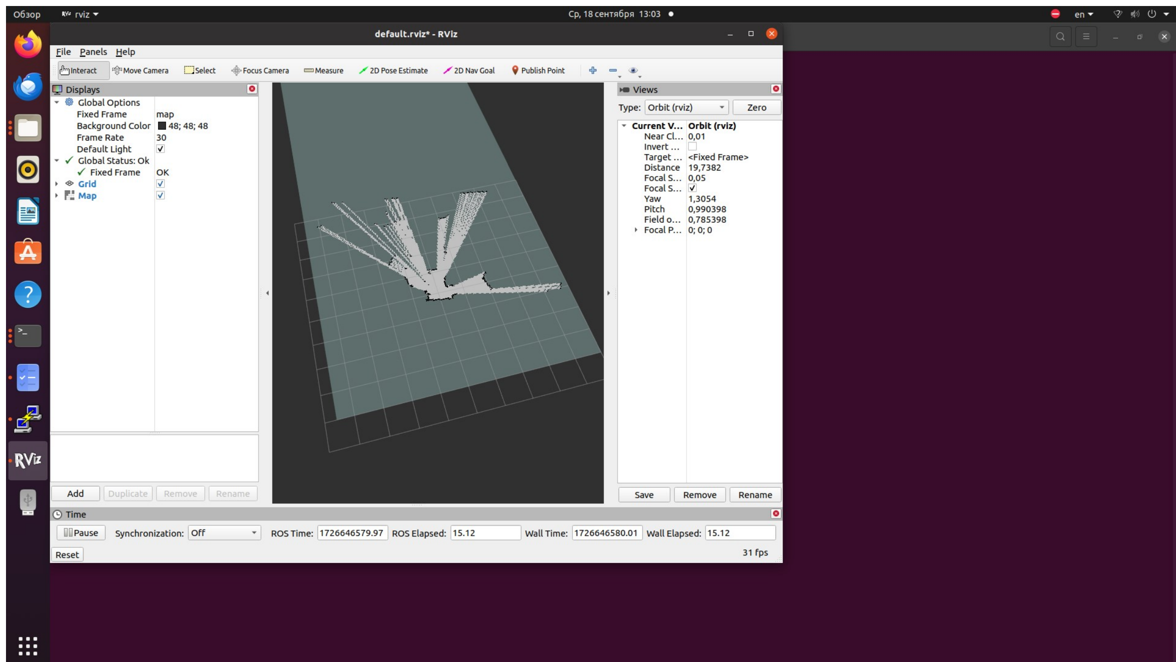
Окно конфигурации putty

После успешного подключения вы точно также сможете взаимодействовать с комплексом через консольные команды, как и в предыдущем варианте.

Например, вызвать камеру.



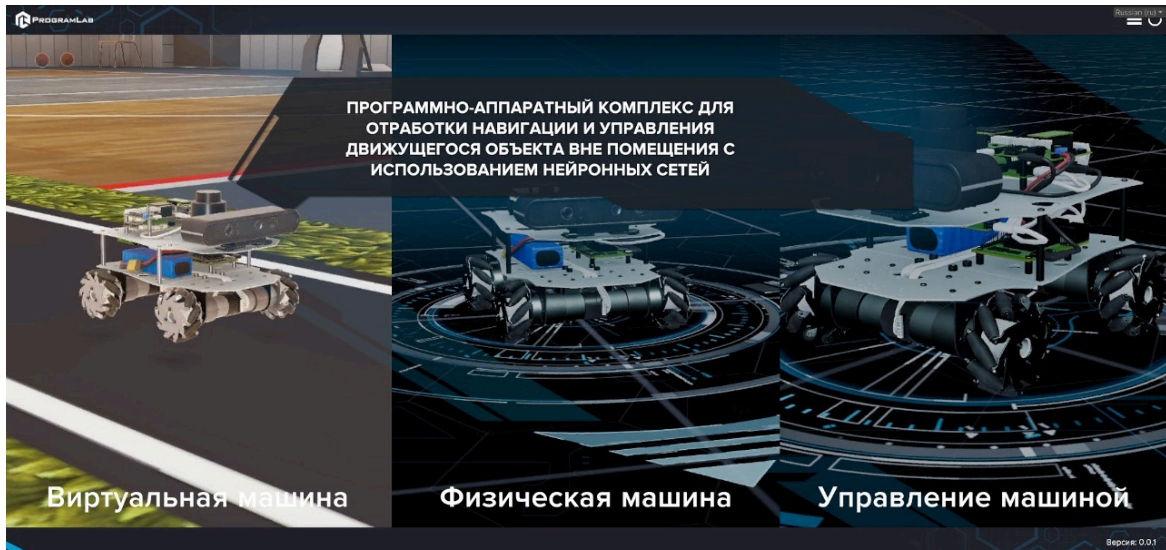
Или же с помощью команды **rviz** открыть построенную lidar карту.



Для того чтобы подключиться к комплексу через SSH необходимо ввести в терминал команду: ssh wheltec@192.168.0.100 dongguan

Управление при помощи геймпада

Для начала работы с геймпадом, в главном меню ПО, необходимо выбрать **Управление машиной**, геймпад автоматически подключится к аппаратному комплексу и появиться возможность управлять им самостоятельно.



Управление аппаратным комплексом через геймпад осуществляется следующим образом:

1 – управление поворотами машинки
2 – управление линейным движением машинки.

Sk Resident

ВИРТУАЛЬНЫЕ ЛАБОРАТОРИИ
ТРЕНАЖЕРЫ - СИМУЛЯТОРЫ
ИНТЕРАКТИВНЫЕ МАКЕТЫ
ЛАБОРАТОРНЫЕ СТЕНДЫ
ЦИФРОВЫЕ ДВОЙНИКИ
VR И AR КОМПЛЕКСЫ

ООО “ПРОГРАМЛАБ” INFO@PL-LLC.RU PLLLC.RU 8 (800) 550 89 72 +7 (499) 685 80 00