



PROGRAMLAB
INNOVATIVE DIGITAL SYSTEMS

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**ВИРТУАЛЬНЫЙ УЧЕБНЫЙ КОМПЛЕКС
«ОРИЕНТАЦИЯ ЛЕТАЮЩЕГО ОБЪЕКТА
НА МЕСТНОСТИ С ИСПОЛЬЗОВАНИЕМ
SLAM МЕТОДОВ»**



ОГЛАВЛЕНИЕ

Общая информация	5
Введение в ROS	6
Назначение ROS.....	6
Базовые понятия ROS.....	7
Обмен сообщениями	9
Определение SLAM.....	10
Принцип работы SLAM с использованием стереокамеры.....	10
Утилиты	11
Поставляемое ПО.....	13
Установка необходимых пакетов для ПК.....	14
Директория ROS:.....	14
Первый запуск.....	15
Топики для просмотра камер:.....	15
Автономный полет (режим Offboard).....	15
Пример использования simple_offboard в данном ПО:	22
Используемый алгоритм SLAM	22
Используемые топики MAVROS	23
Разработка и эксплуатация собственного функционала	24
Использование msg и srv	25
Информация о msg.....	25
Создание нового msg	25
Использование rosmmsg.....	27

Создание srv	28
Использование rossrv	29
Общее для msg и srv	30
Создание Publisher ноды	31
Объяснение кода	32
Написание Subscriber ноды	35
Объяснение кода	36
Создание нод	36
Запуск publisher	36
Запуск subscriber	37
Инструкция по установке и запуску проекта	39
Начало работы с комплексом	42
Подключение к комплексу через Wi-Fi	46
Работа с комплексом в режиме редактора карт	53
Работа с комплексом через веб – интерфейс	56
Работа с полетом	61
Настройка полетного контроллера	63
Выбор рамы	64
Параметры	65
Рекомендованные значения	65
Настройки подсистемы Estimator	66
Настройка PID-регуляторов	67

Общая информация

Лабораторный комплекс (далее - комплекс) «Ориентация летающего объекта (квадрокоптера) на местности с использованием SLAM методов» позволяет проводить практические занятия по решению ряд задач, возникающих при навигации в пространстве и управлении летающим объектом: составление карты местности, нахождение на ней ориентиров, само-позиционирование. Комплекс содержит в своем составе специализированное программное обеспечение, а также физический экземпляр-имитатор объекта управления в виде малогабаритного квадрокоптера с установленной на борту стерео - камерой, для отработки и визуализации процесса управления. Данный учебный комплекс используется в качестве виртуальной песочницы или наглядного пособия-полигона для изучения принципов работы алгоритмов и особенностей их применения и реализации. Взаимодействие с коптером осуществляется при помощи ROS. ROS обеспечивает гибкость и эффективность в управлении квадрокоптером, позволяя оператору легко взаимодействовать с квадрокоптером, а также получить навыки программирования и настройки летательных аппаратов.



Внешний вид летательного аппарата

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 напрямую). 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 определена 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 объекта (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 синхронной 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 этой главе мы подробно рассмотрим способы коммуникации между нодами.

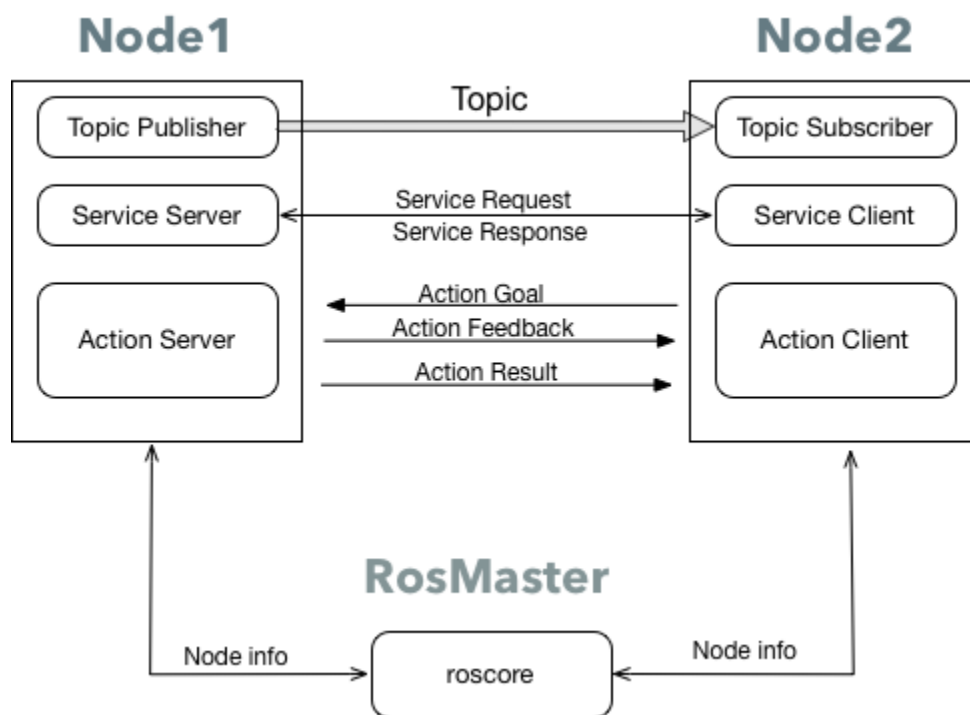
Существуют три основных способа (концепции) коммуникации:

Топик (Topic), 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:



Определение SLAM

SLAM (Simultaneous Localization and Mapping) — это метод, который позволяет мобильному роботу или транспортному средству строить карту неизвестного окружения и одновременно определять своё местоположение в этой карте. Этот процесс необходим для автономной навигации, особенно в сложных и динамических средах, где предварительно созданная карта недоступна или ненадёжна.

Принцип работы SLAM с использованием стереокамеры

SLAM с использованием стереокамеры работает на основе анализа изображений, получаемых с двух камер, расположенных на небольшом расстоянии друг от друга, подобно глазам человека. Этот метод позволяет определять глубину объектов и строить трёхмерное представление окружающего пространства. Основные этапы работы SLAM с использованием стереокамеры включают:

Сбор данных:

- Стереокамера захватывает два изображения одной и той же сцены с разных точек зрения.
- Эти изображения используются для определения параллакса (различий между изображениями), что позволяет вычислить глубину каждого пикселя.

Извлечение особенностей:

- Алгоритмы извлекают ключевые точки (особенности) из пары изображений. Эти особенности могут быть углы, края или другие заметные части изображения.

Соответствие особенностей:

- Извлечённые особенности из двух изображений сопоставляются друг с другом, что позволяет определить, какие части изображений соответствуют одним и тем же объектам в реальном мире.

Определение глубины:

- На основе сдвига между соответствующими особенностями (параллакса) вычисляется расстояние до объектов в сцене, что даёт трёхмерное представление окружающего пространства.

Построение карты и локализация:

- Полученные данные о глубине используются для построения карты окружающей среды.
- Одновременно алгоритм определяет текущее положение и ориентацию камеры в этой карте.

Оптимизация:

- В процессе движения робота собираются новые данные, которые используются для уточнения карты и улучшения точности локализации.

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.

Для запуска визуализация состояния Клевера в реальном времени, необходимо подключиться к нему по Wi-Fi (clover-xxxx) и запустить rviz, указав соответствующий ROS_MASTER_URI:

```
1 ROS_MASTER_URI=http://192.168.11.1:11311 rviz
```

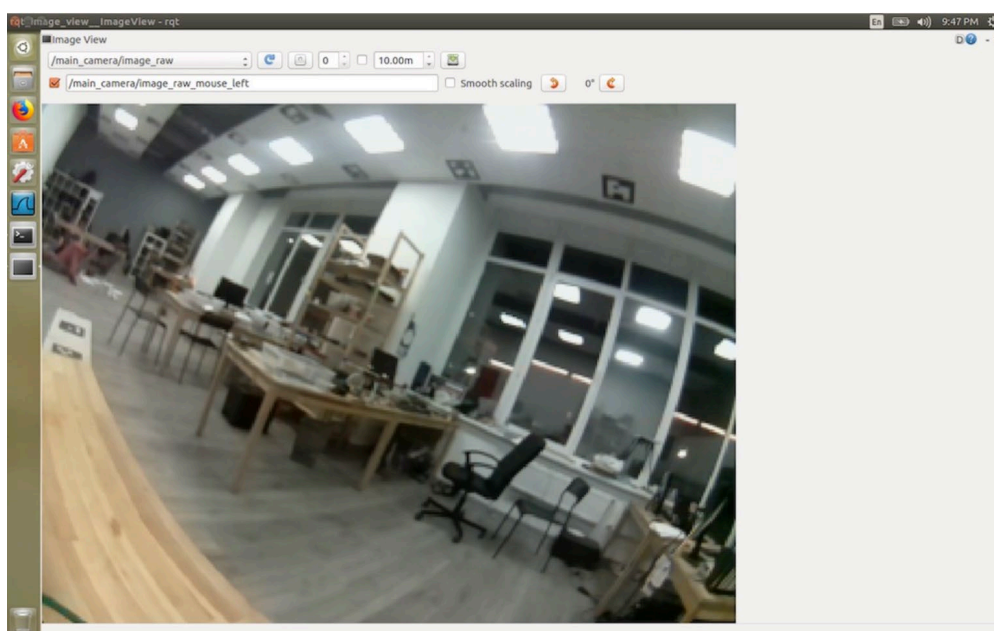
Утилита для просмотра изображения rqt_image_view

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://192.168.11.1:11311 rqt_image_view
```

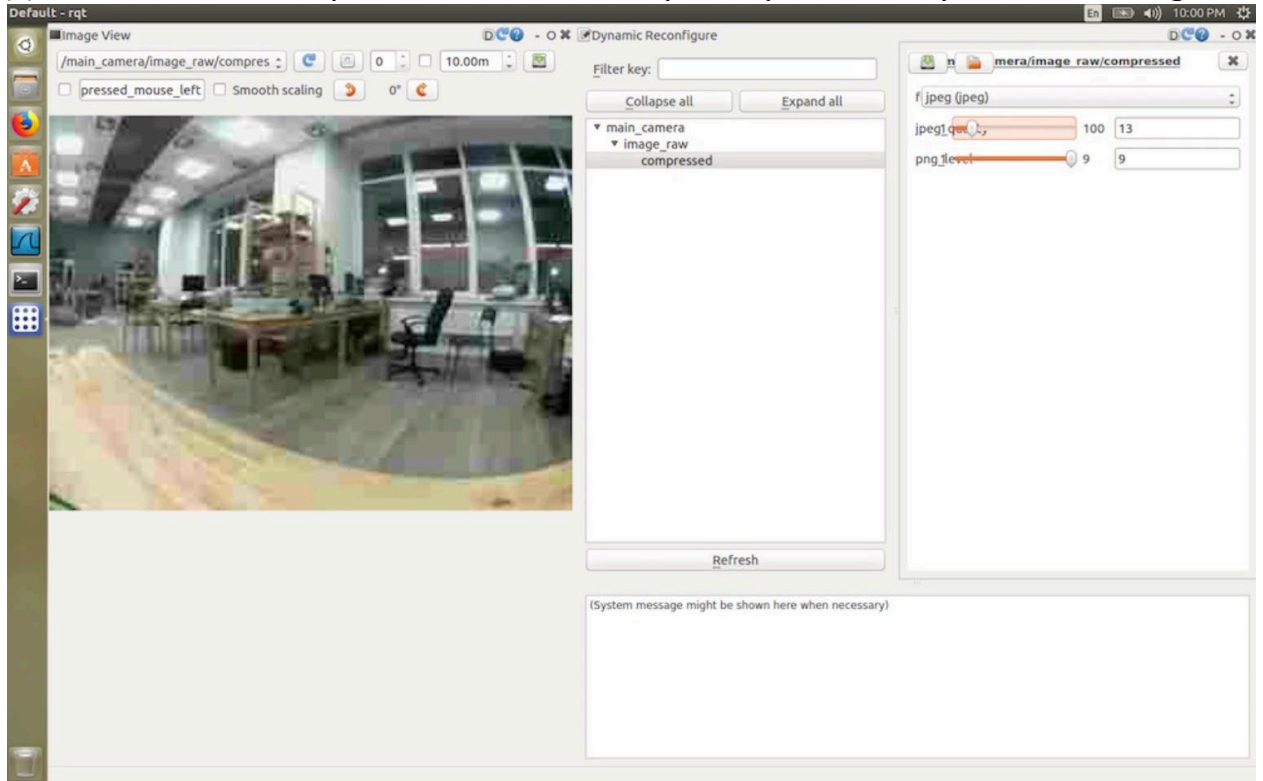
Выберите топик для просмотра, например, /main_camera/image_raw:

Результат визуализации коптера и камеры представлен ниже:



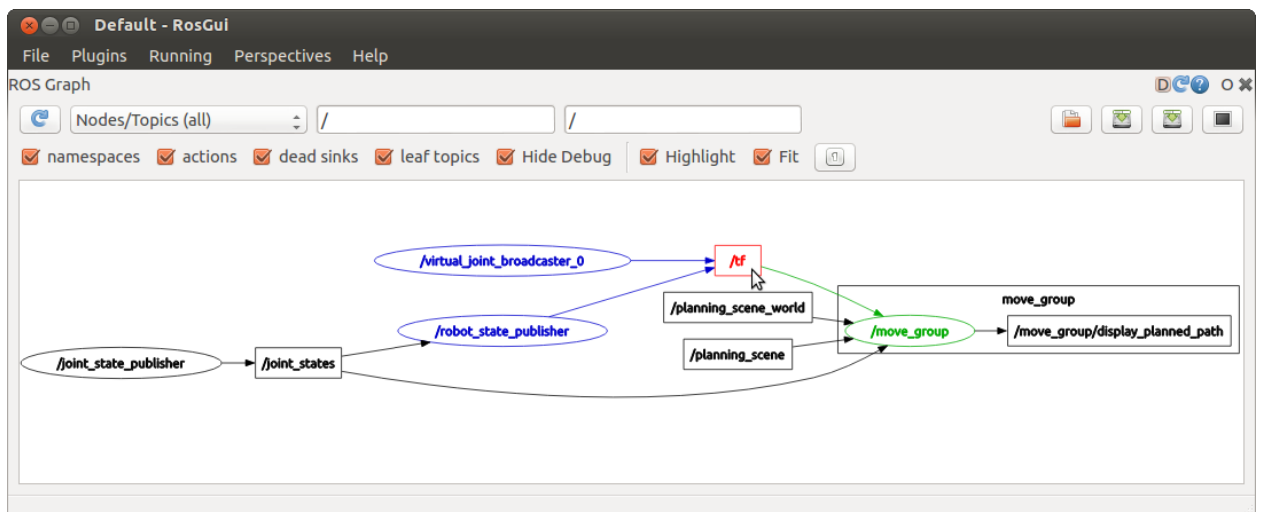
Для снижения нагрузки на сеть и уменьшения задержки используйте сжатый вариант топика – /main_camera/image_raw/compressed.

Для изменения настроек сжатия используйте rqt-плагин Dynamic Reconfigure:



Rqt_graph

Утилита rqt_graph позволяет визуализировать взаимосвязи запущенных нод в системе ROS



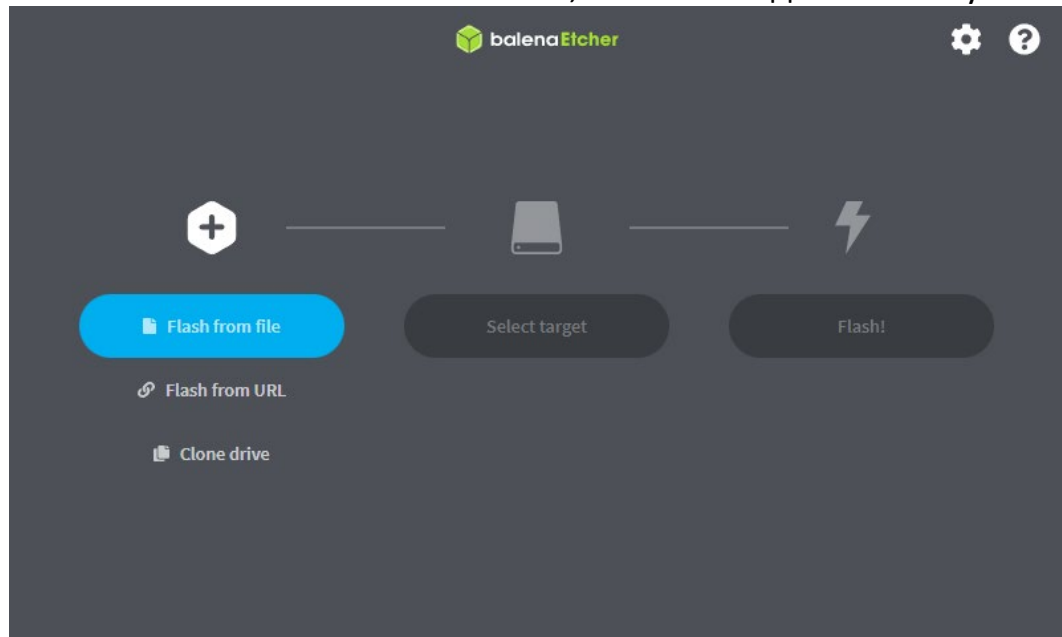
Поставляемое ПО

Для своего продукта мы используем модифицированный образ Clover v.0.24_master.img для Master и образ Clover v.0.24_slave.img для Slave. Данные образы можно найти на флешке, которая входит в комплект поставки

В комплект поставки входят SD - карты, вставленные в Raspberry pi, на которых уже установлены требуемые образы.

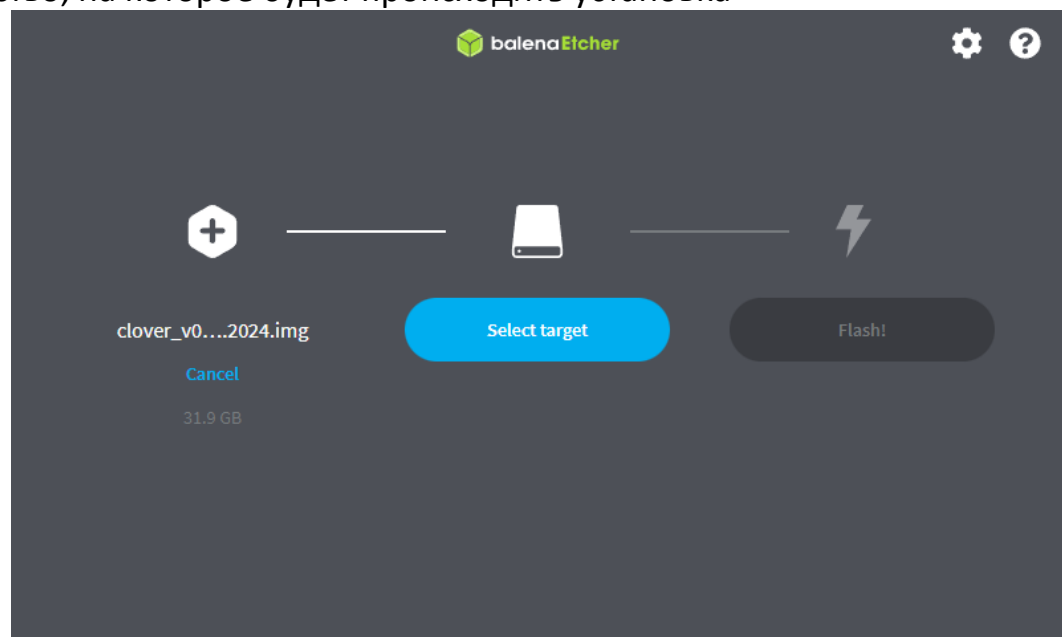
Для того чтобы установить образ на новую флеш-карту необходимо установить программу BalenaEtcher, которая есть на флешке с поставляемым ПО.

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

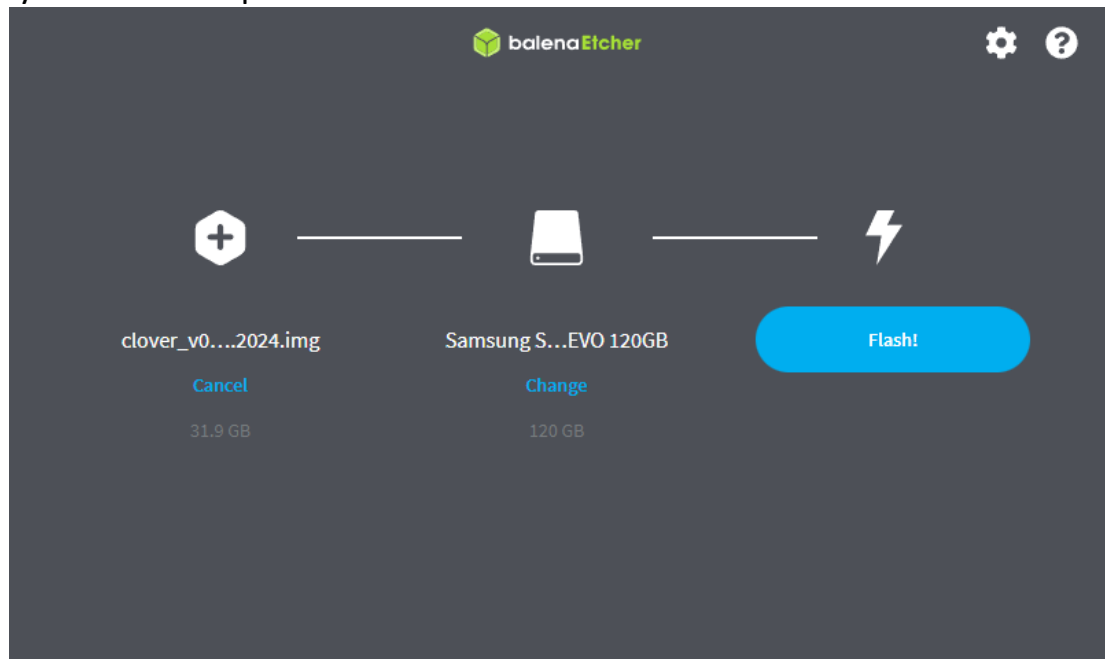


Нажмите кнопку Flash from file, после чего в открывшемся окне выберите нужный вам образ (master или slave)

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/UAVSlamNavigation/Base_M/ROS/

В папке `catkin_ws/src/pl_uav_slam/` 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/UAVSlamNavigation/Base_M/ROS/` открыть терминал и выполнить следующие команды:

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

Это может занять несколько минут.

Топики для просмотра камер:

- Изображение с левой камеры - `/main_camera/image_raw`;
- Сжатое изображение с левой камеры - `/main_camera/image_raw/compressed`;
- Изображение с правой камеры - `/secondary_camera/image_raw`;
- Сжатое изображение с правой камеры - `/secondary_camera/image_raw/compressed`.

Автономный полет (режим Offboard)

Модуль `simple_offboard` пакета `clover` предназначен для упрощенного программирования автономного полета дрона (режим OFFBOARD). Он позволяет устанавливать желаемые полетные задачи и автоматически трансформирует систему координат.

`simple_offboard` является высокоуровневым способом взаимодействия с полетным контроллером. Для более низкоуровневой работы см. `maavros`.

Основные сервисы – `get_telemetry` (получение телеметрии), `navigate` (полет в заданную точку по прямой), `navigate_global` (полет в глобальную точку по прямой), `land` (переход в режим посадки).

Использование из языка Python

Для использования сервисов, необходимо создать объекты-прокси к ним. Используйте этот шаблон для вашей программы:

1	<code>import rospy</code>
2	<code>from clover import srv</code>
3	<code>from std_srvs.srv import Trigger</code>
4	
5	<code>rospy.init_node('flight')</code>
6	

```

7   get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
8   navigate = rospy.ServiceProxy('navigate', srv.Navigate)
9   navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
10  set_altitude = rospy.ServiceProxy('set_altitude', srv.SetAltitude)
11  set_yaw = rospy.ServiceProxy('set_yaw', srv.SetYaw)
12  set_yaw_rate = rospy.ServiceProxy('set_yaw_rate', srv.SetYawRate)
13  set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
14  set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
15  set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
16  set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
17  land = rospy.ServiceProxy('land', Trigger)

```

Неиспользуемые функции-прокси можно удалить из кода.

Описание API

Незаполненные числовые параметры устанавливаются в значение 0.

get_telemetry - Получить полную телеметрию коптера.

Параметры:

frame_id – система координат для значений x, y, z, vx, vy, vz. Пример: map, body, aruco_map. Значение по умолчанию: map.

Формат ответа:

frame_id – система координат;

connected – есть ли подключение к FCU;

armed – состояние armed винтов (винты включены, если true);

mode – текущий полетный режим;

x, y, z – локальная позиция коптера (м);

lat, lon – широта, долгота (градусы), необходимо наличие GPS;

alt – высота в глобальной системе координат (стандарт WGS-84, не AMSL!), необходимо наличие GPS;

vx, vy, vz – скорость коптера (м/с);

roll – угол по крену (радианы);

pitch – угол по тангажу (радианы);

yaw – угол по рысканью (радианы);

roll_rate – угловая скорость по крену (рад/с);

pitch_rate – угловая скорость по тангажу (рад/с);

yaw_rate – угловая скорость по рысканью (рад/с);

voltage – общее напряжение аккумулятора (В);

cell_voltage – напряжение аккумулятора на ячейку (В).

Недоступные по каким-то причинам поля будут содержать в себе значения NaN.

Вывод координат x, y и z коптера в локальной системе координат:

```
1 telemetry = get_telemetry()
2 print(telemetry.x, telemetry.y, telemetry.z)
```

Вывод высоты коптера относительно карты ArUco-меток:

```
1 telemetry = get_telemetry(frame_id='aruco_map')
2 print(telemetry.z)
```

Проверка доступности глобальной позиции:

```
1 import math
2 if not math.isnan(get_telemetry().lat):
3     print('Global position is available')
4 else:
5     print('No global position')
```

Вывод текущей телеметрии (командная строка):

```
rosservice call /get_telemetry "{frame_id: ""}"
```

navigate

Прилететь в обозначенную точку по прямой.

Параметры:

x, y, z – координаты (м);

yaw – угол по рысканью (радианы);

speed – скорость полета (скорость движения setpoint) (м/с);

auto_arm – перевести коптер в OFFBOARD и заармить автоматически (коптер взлетит);

frame_id – система координат, в которой заданы x, y, z и yaw (по умолчанию: map).

Для полета без изменения угла по рысканью достаточно установить yaw в NaN (значение угловой скорости по умолчанию – 0).

Взлет на высоту 1.5 м со скоростью взлета 0.5 м/с:

```
navigate(x=0, y=0, z=1.5, speed=0.5, frame_id='body', auto_arm=True)
```

Полет по прямой в точку 5:0 (высота 2) в локальной системе координат со скоростью 0.8 м/с (рысканье установится в 0):

```
navigate(x=5, y=0, z=3, speed=0.8)
```

Полет в точку 5:0 без изменения угла по рысканью:

```
navigate(x=5, y=0, z=3, speed=0.8, yaw=float('nan'))
```

Полет вправо относительно коптера на 3 м:


```
navigate(x=0, y=-3, z=0, speed=1, frame_id='body')
```

Полет влево на 2 м относительно последней целевой точки полета дрона:

```
navigate(x=0, y=2, z=0, speed=1, frame_id='navigate_target')
```

Повернуться на 90 градусов по часовой:

```
navigate(yaw=math.radians(-90), frame_id='body')
```

Полет в точку 3:2 (высота 2) в системе координат маркерного поля со скоростью 1 м/с:

```
navigate(x=3, y=2, z=2, speed=1, frame_id='aruco_map')
```

Взлет на высоту 2 м (командная строка):

```
rosservice call /navigate "{x: 0.0, y: 0.0, z: 2, yaw: 0.0, speed: 0.5, frame_id: 'body', auto_arm: true}"
```

При программировании миссии дрона в терминах "вперед-назад-влево-вправо" рекомендуется использовать систему координат `navigate_target` вместо `body`, чтобы не учитывать неточность прилета дрона в предыдущую целевую точку при вычислении следующей.

navigate_global

Полет по прямой в точку в глобальной системе координат (широта/долгота).

Параметры:

`lat, lon` – широта и долгота (градусы);

`z` – высота (м);

`yaw` – угол по рысканью (радианы);

`speed` – скорость полета (скорость движения setpoint) (м/с);

`auto_arm` – перевести коптер в OFFBOARD и заармить автоматически (коптер взлетит);

`frame_id` – система координат, в которой заданы `z` и `yaw` (по умолчанию: `map`).

Для полета без изменения угла по рысканью достаточно установить `yaw` в `NaN`.

Полет в глобальную точку со скоростью 5 м/с, оставаясь на текущей высоте (`yaw` установится в 0, коптер ориентируется передом на восток):

```
navigate_global(lat=55.707033, lon=37.725010, z=0, speed=5, frame_id='body')
```

Полет в глобальную точку без изменения угла по рысканью:

```
navigate_global(lat=55.707033, lon=37.725010, z=0, speed=5,  
yaw=float('nan'), frame_id='body')
```

Полет в глобальную точку (командная строка):

```
rosservice call /navigate_global "{lat: 55.707033, lon: 37.725010, z: 0.0, yaw:  
0.0, speed: 5.0, frame_id: 'body', auto_arm: false}"
```

set_altitude

Изменить целевую высоту полета. Сервис используется для независимой установки высоты (и системы координат для расчета высота) в режимах полета `navigate` и `set_position`.

Параметры:

`z` – высота полета (м);

`frame_id` – система координат для расчета высоты полета.

Установить высоту полета в 2 м относительно пола:

```
set_altitude(z=2, frame_id='terrain')
```

Установить высоту полета в 1 м относительно маркерного поля:

```
set_altitude(z=1, frame_id='aruco_map')
```

set_yaw

Изменить целевой угол по рысканью (и систему координат для его расчета), оставив предыдущую команду в силе.

Параметры:

`yaw` – угол по рысканью (радианы);

`frame_id` – система координат для расчета угла по рысканью.

Повернуться на 90 градусов по часовой (продолжая выполнять предыдущую команду):

```
set_yaw(yaw=math.radians(-90), frame_id='body')
```

Установить угол по рысканью в ноль в системе координат маркерного поля:

```
set_yaw(yaw=0, frame_id='aruco_map')
```

Остановить вращение по рысканью (при использовании `set_yaw_rate`):

```
set_yaw(yaw=float('nan'))
```

set_yaw_rate

Изменить целевую угловую скорость по рысканью, оставив предыдущую команду в силе.

Параметры:

yaw_rate – угловая скорость по рысканью (рад/с).

Положительное направление вращения (при виде сверху) – против часовой.

Начать вращение на месте со скоростью 0.5 рад/с против часовой (продолжая выполнять предыдущую команду):

```
set_yaw_rate(yaw_rate=0.5)
```

set_position

Установить цель по позиции и рысканью. Данный сервис следует использовать при необходимости задания продолжающегося потока целевых точек, например, для полета по сложным траекториям (круговой, дугообразной и т. д.).

Для полета на точку по прямой или взлета используйте более высокоуровневый сервис `navigate`.

Параметры:

x, y, z – координаты точки (м);

yaw – угол по рысканью (радианы);

auto_arm – перевести коптер в OFFBOARD и заармить автоматически (коптер взлетит);

frame_id – система координат, в которой заданы x, y, z и yaw (по умолчанию: `map`).

Зависнуть на месте:

```
set_position(frame_id='body')
```

Назначить целевую точку на 3 м выше текущей позиции:

```
set_position(x=0, y=0, z=3, frame_id='body')
```

Назначить целевую точку на 1 м впереди текущей позиции:

```
set_position(x=1, y=0, z=0, frame_id='body')
```

set_velocity

Установить скорости и рысканье.

v_x, v_y, v_z – требуемая скорость полета (м/с);

yaw – угол по рысканью (радианы);

`auto_arm` – перевести коптер в OFFBOARD и заармить автоматически (коптер взлетит);

`frame_id` – система координат, в которой заданы `vx`, `vy`, `vz` и `yaw` (по умолчанию: `map`).

Параметр `frame_id` определяет только ориентацию результирующего вектора скорости, но не его длину.

Полет вперед (относительно коптера) со скоростью 1 м/с:

```
set_velocity(vx=1, vy=0.0, vz=0, frame_id='body')
```

set_attitude

Установить тангаж, крен, рысканье и уровень газа (примерный аналог управления в режиме STABILIZED). Данный сервис может быть использован для более низкоуровневого контроля поведения коптера либо для управления коптером при отсутствии источника достоверных данных о его позиции.

Параметры:

`roll`, `pitch`, `yaw` – необходимый угол по тангажу, крену и рысканью (радианы);

`thrust` – уровень газа от 0 (нет газа, пропеллеры остановлены) до 1 (полный газ);

`auto_arm` – перевести коптер в OFFBOARD и заармить автоматически (коптер взлетит);

`frame_id` – система координат, в которой задан `yaw` (по умолчанию: `map`).

set_rates

Установить угловые скорости по тангажу, крену и рысканью и уровень газа (примерный аналог управления в режиме ACRO). Это самый низкий уровень управления коптером (исключая непосредственный контроль оборотов моторов). Данный сервис может быть использован для автоматического выполнения акробатических трюков (например, флипа).

Параметры:

`roll_rate`, `pitch_rate`, `yaw_rate` – угловая скорость по тангажу, крену и рысканью (рад/с);

`thrust` – уровень газа от 0 (нет газа, пропеллеры остановлены) до 1 (полный газ).

`auto_arm` – перевести коптер в OFFBOARD и заармить автоматически (коптер взлетит);

Положительное направление вращения `yaw_rate` (при виде сверху) – против часовой, `pitch_rate` – вперед, `roll_rate` – влево.

Land

Перевести коптер в режим посадки (AUTO.LAND или аналогичный).

Для автоматического отключения винтов после посадки параметр PX4 COM_DISARM_LAND должен быть установлен в значение > 0.

Посадка коптера:

1	res = land()
2	if res.success:
3	print('Copter is landing')

Посадка коптера (командная строка):

```
rosservice call /land "{}"
```

В более новых версиях PX4 коптер выйдет из режима LAND в ручной режим, если сильно перемещать стики.

release

В случае необходимости приостановки отправки setpoint-сообщений, используйте сервис simple_offboard/release:

1	release = rospy.ServiceProxy('simple_offboard/release', Trigger)
2	release()

Пример использования simple_offboard в данном ПО:

В папке

/var/share/ProgramLab/UAVSlamNavigation/Base_M/ROS/catkin_ws/src/pl_uav_slam_python_examples присутствует Python файл flight.py, который запускает полёт, описывающий квадрат со стороной в 1 метр. Для его запуска необходимо открыть в данной папке терминал и выполнить команду

```
"$python3 flight.py"
```

Используемый алгоритм SLAM

Программный модуль использует модифицированный пакет ROS с открытым исходным кодом OV²SLAM (<https://github.com/ov2slam/ov2slam>).

Основные изменения включают:

- Для уменьшения задержки используются топики с сжатыми изображениями из камер;
- Для удобства переименованы некоторые топики.

Исходный код пакета находится в директории catkin_ws/src/ov2slam.

Основные топики, публикуемые пакетом:

- `/pl_uav_slam/image_track` – отладочное изображение того как работает алгоритм, синие точки обозначают потенциальные точки для отслеживания, красные точки – точки для которых алгоритм успешно посчитал расстояние, зеленые точки – успешно отслеженные из предыдущих кадров точки;
- `/pl_uav_slam/point_cloud` – облако точек, построенное при помощи алгоритма;
- `tf` фрейм `slam_camera` – вычисленное при помощи алгоритма положение камеры.

Обратите внимание, что `point_cloud` и `slam_camera` находятся в оптической системе координат, в которой:

- ось X направлена вправо;
- ось Y направлена вниз;
- ось Z направлена вперед.

Для преобразования данных о позиции дрона из `OV2SLAM` в `MAVROS` используется нода `slam_pose_publisher_node` из пакета `pl_uav_slam`. Она публикует рассчитанную позицию дрона сообщением типа [geometry_msgs/PoseStamped](https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/PoseStamped.html) (https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/PoseStamped.html) в топик `pl_uav_slam/vision_pose/pose`.

Используемые топики MAVROS

Для работы автономного полёта полётному контроллеру необходимо знать своё положение. Встроенный алгоритм SLAM передаёт своё положение контроллеру при помощи топика `/mavros/vision_pose/pose`, принимающего сообщение типа `geometry_msgs/PoseStamped`.

Так как полётному контроллеру необходим непрерывный поток данных о своей позиции с частотой 50 Гц, на стороне дрона запущена нода `pl_uav_slam_vpe_adapter`, пересылающая сообщения из топика `pl_uav_slam/vision_pose/pose` в `/mavros/vision_pose/pose` с постоянной частотой.

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	<i>Header header</i>
2	<i>string child_frame_id</i>
3	<i>geometry_msgs/PoseWithCovariance pose</i>
4	<i>geometry_msgs/TwistWithCovariance twist</i>

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	<i>int64 A</i>
2	<i>int64 B</i>
3	<i>---</i>
4	<i>int64 Sum</i>

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 `rosmmsg`

Убедитесь, что ROS его видит с помощью команды `rosmmsg show`.

Использование:

```
$ rosmmsg show [message type]
```

Пример

```
$ rosmmsg 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:

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

```

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

```

1 $ rosrvc 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
15 if __name__ == '__main__':
16     try:
17         talker()
18     except rospy.ROSInterruptException:
19         pass

```

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	<code>rate = rospy.Rate(10) # 10hz</code>
---	---

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

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

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	<code>\$ cd ~/catkin_ws</code>
2	<code>\$ catkin_make</code>

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

```
$ rosrn beginner_tutorials talker (C++)
$ rosrn 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 rosrn 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_1314931968795I
17	heard hello world 1314931974.28
18	[INFO] [WallTime: 1314931975.283708] /listener_17657_1314931968795I
	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.

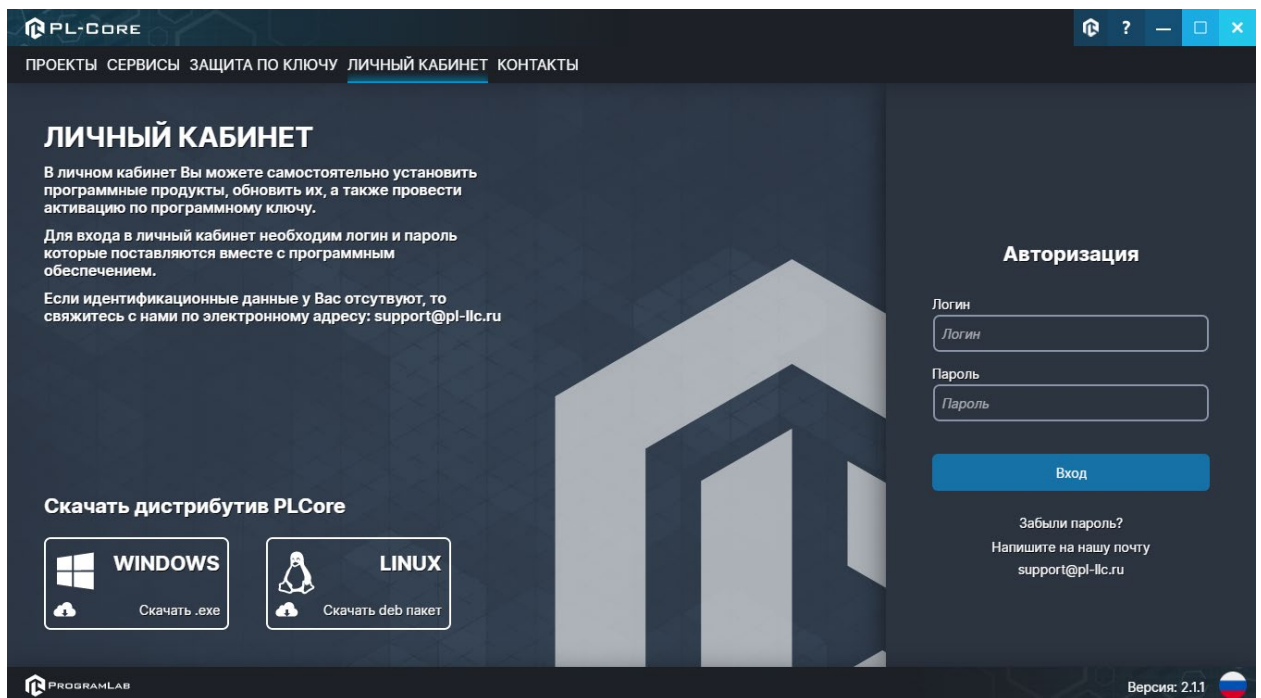
В случае поставки программного комплекса без ПК вам необходимо установить программное обеспечение с USB-носителя.

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».

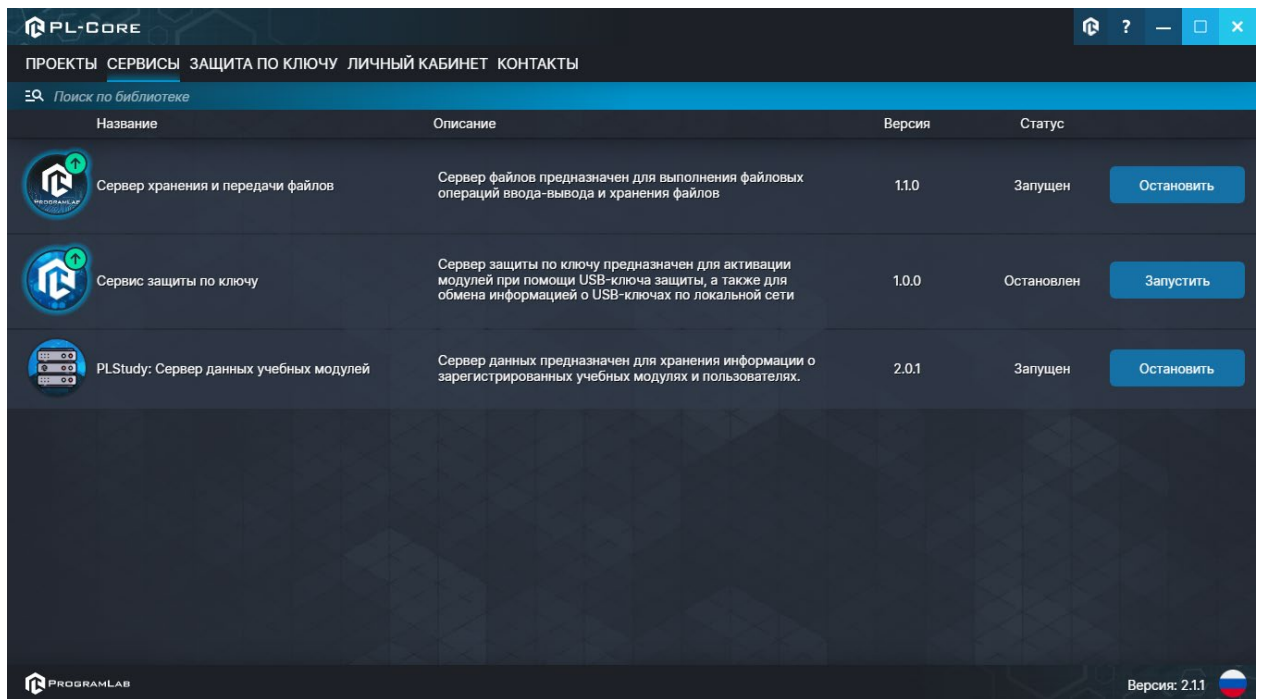
В комплект поставки входит **конверт с идентификационными данными для личного кабинета**. Если конверта нет, то напишите нам на почту support@pl-llc.ru.

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.

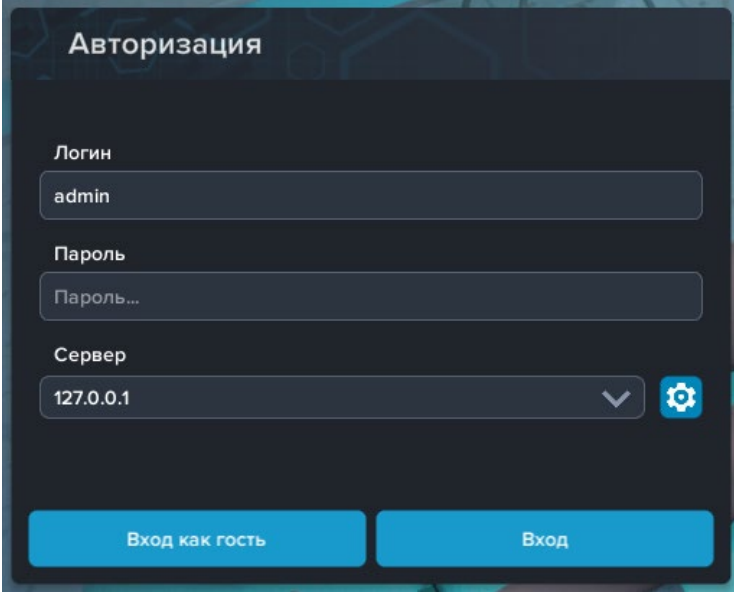
По умолчанию логин администратора: admin; Пароль: admin.

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»**.

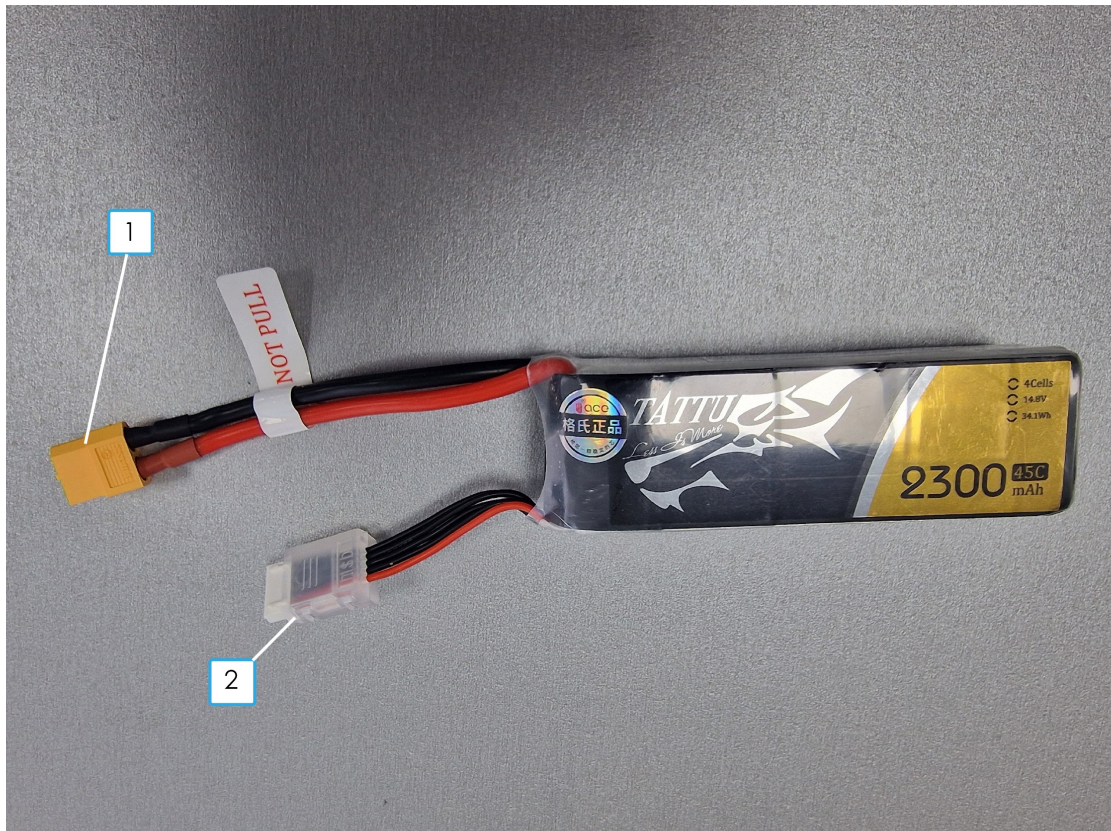


The image shows a dark-themed authorization window titled "Авторизация". It contains three input fields: "Логин" with the value "admin", "Пароль" with the placeholder "Пароль...", and "Сервер" with the value "127.0.0.1". The "Сервер" field has a dropdown arrow and a settings gear icon. At the bottom, there are two blue buttons: "Вход как гость" and "Вход".

Окно авторизации

Начало работы с комплексом

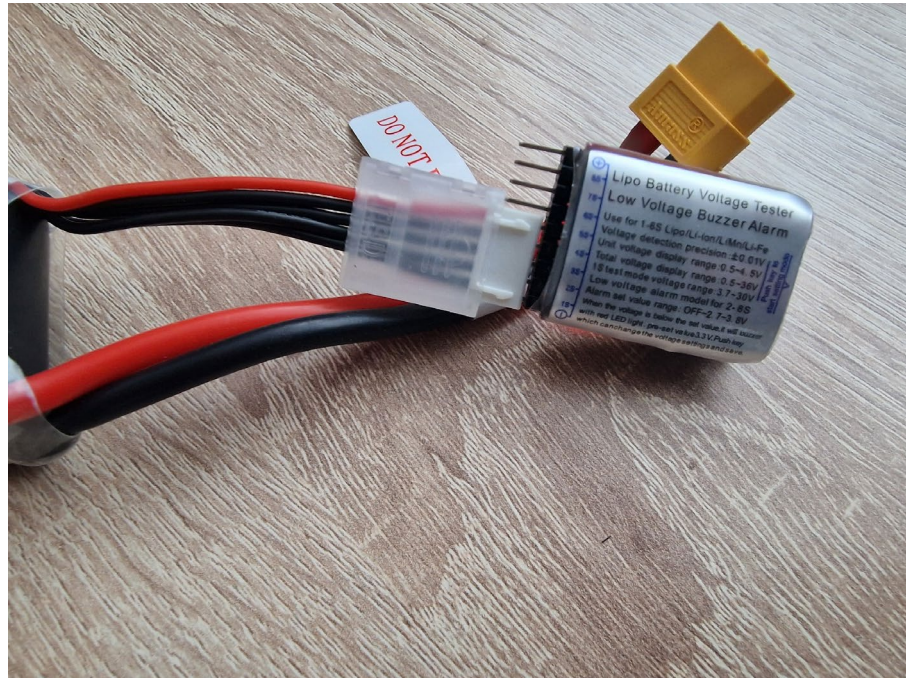
Летательный аппарат питается от аккумуляторной батареи типа Li-Po, состоящей из 4 ячеек (4S), номинальным напряжением 14.8 В.



Аккумуляторная батарея

1. Разъём для подключения аккумуляторной батареи к летательному аппарату.
2. Балансировочный разъем (для зарядки).

Перед началом работы с комплексом убедитесь, что аккумуляторная батарея заряжена. Сделать это можно при помощи тестера, который входит в комплект. Вставьте тестер в балансировочный разъем аккумулятора, как показано на рисунке:



Подключение тестера.

После подключения тестер пропищит и покажет информацию о батарее, а также по всем ее ячейкам. Общее рабочее напряжение (от 14.4 до 16.8)



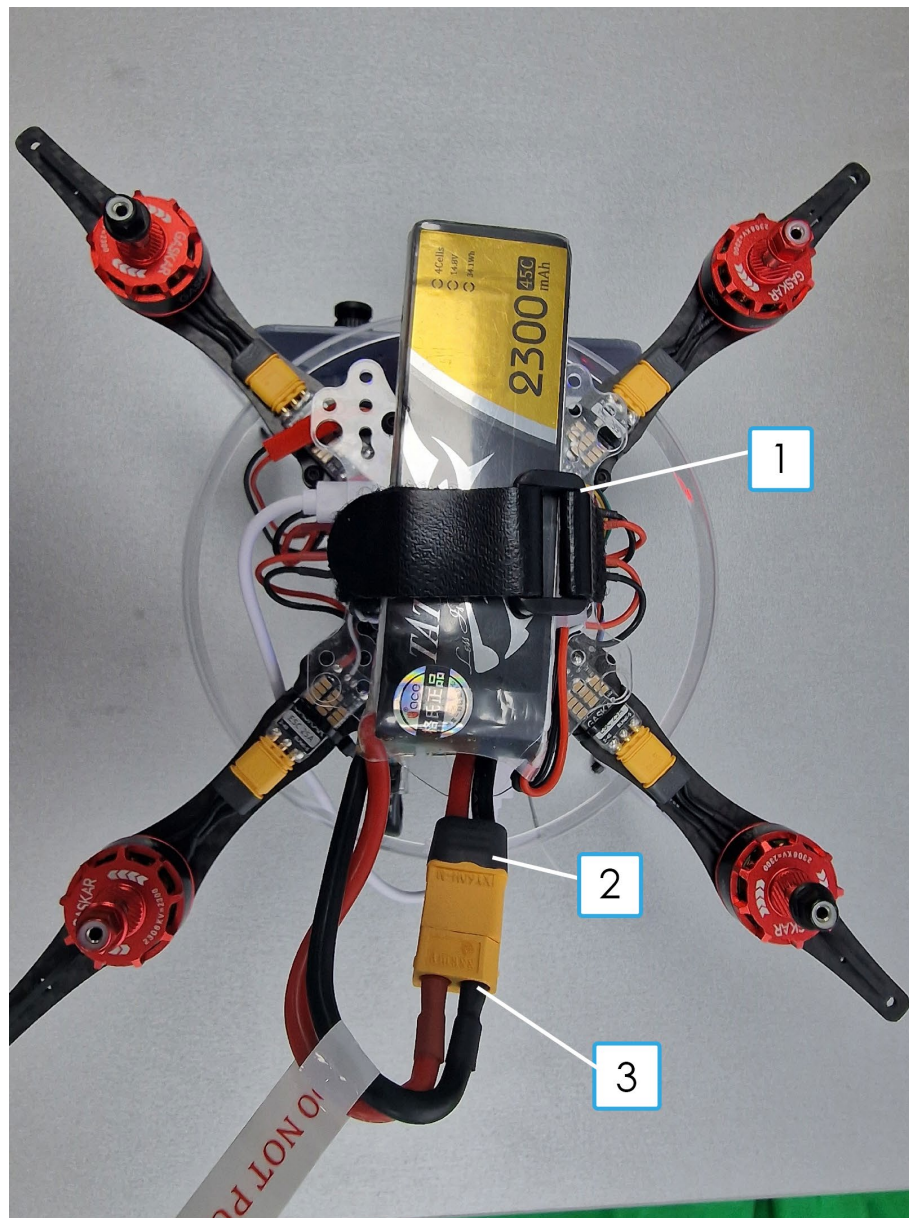
Информация о батарее

Для зарядки АКБ используйте зарядное устройство, которое идет в комплекте, для того чтобы поставить аккумулятор на зарядку, подключите балансирующий разъем аккумулятора к разъему с обозначением 4S на зарядном устройстве



Правильное подключение и процесс зарядки АКБ.

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



Подключение питания к дрону

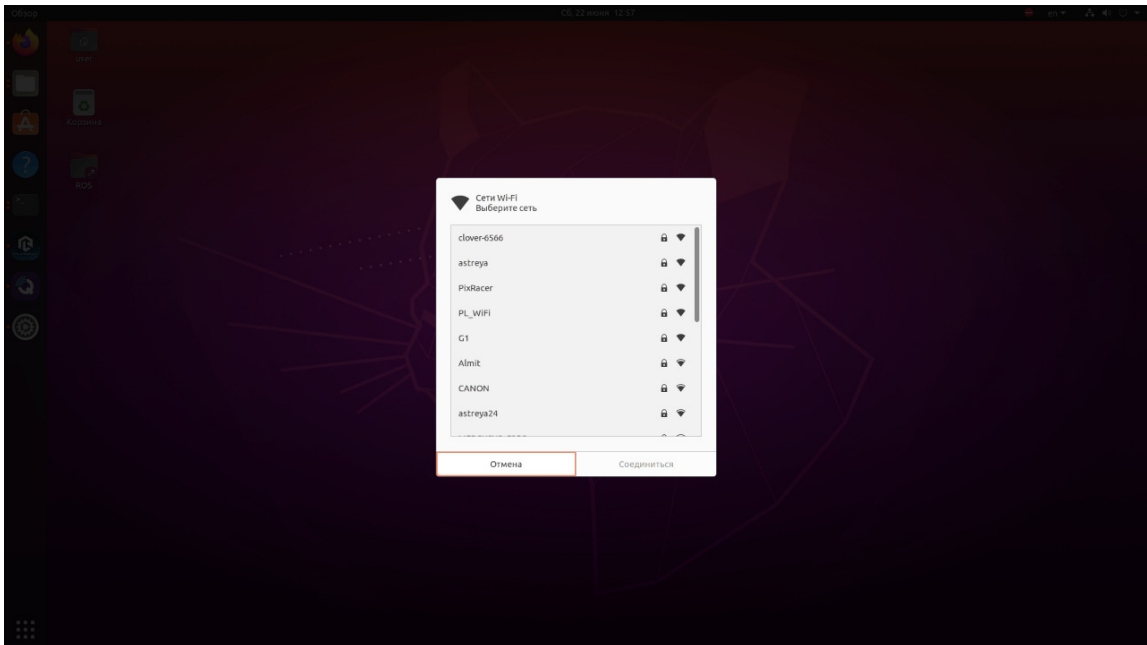
Для того чтобы подключить питание к дрону, необходимо закрепить батарею при помощи фиксатора 1, представленного в виде ленты липучки контактной.

После этого необходимо соединить разъем аккумуляторной батареи 3 с разъемом питания дрона 2.

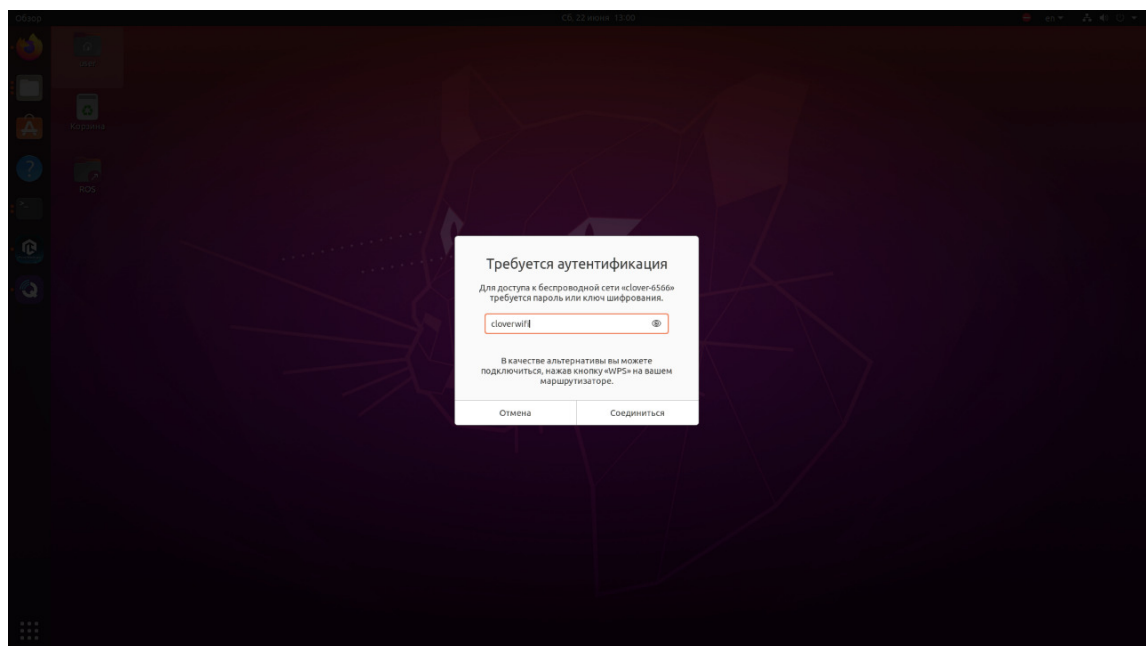
При правильном подключении коптер издаст звуковой сигнал и примерно через 30 секунд раздаст сеть Wi-Fi

Подключение к комплексу через Wi-Fi

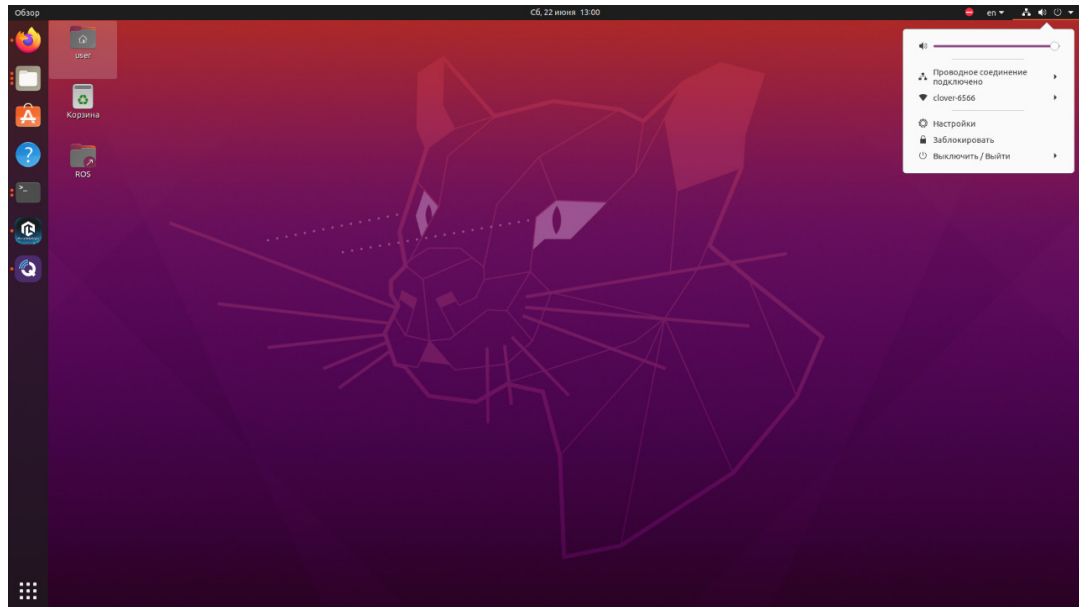
Для того чтобы подключиться к летательному аппарату, необходимо найти его в списке сетей Wi-Fi, он будет раздавать сеть с именем: Clover-xxxx



Выберите эту Wi-Fi сеть и введите пароль «cloverwifi»

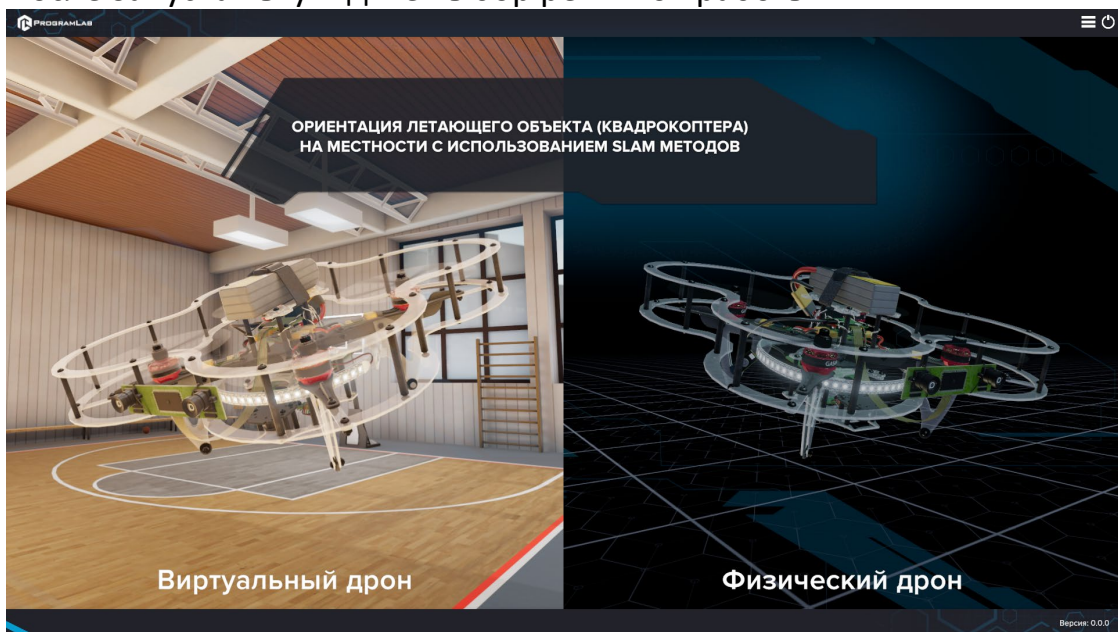


Если вы сделали все правильно то вы увидите что успешно подключены к Wi-Fi



После подключения WI-FI, нужно запустить программное обеспечение комплекса.

После запуска вы увидите выбор режимов работы:



Режимы работы ПО

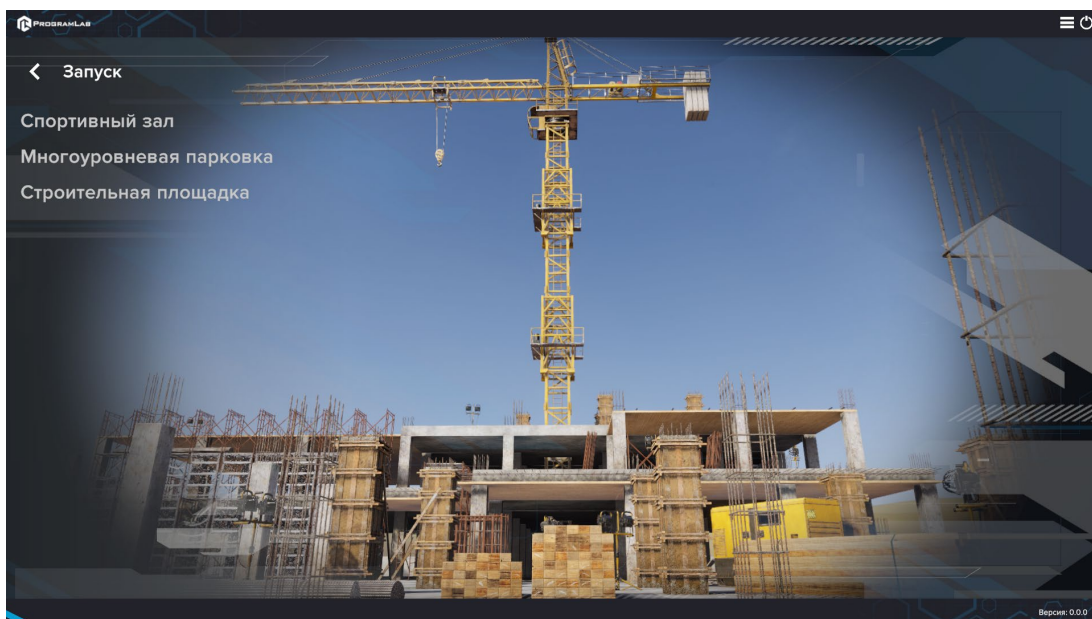
В нашем ПО реализована поддержка работы как с виртуальным летательным аппаратом, так и поддержка работы с физическим дроном.



Меню работы с виртуальным дроном

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

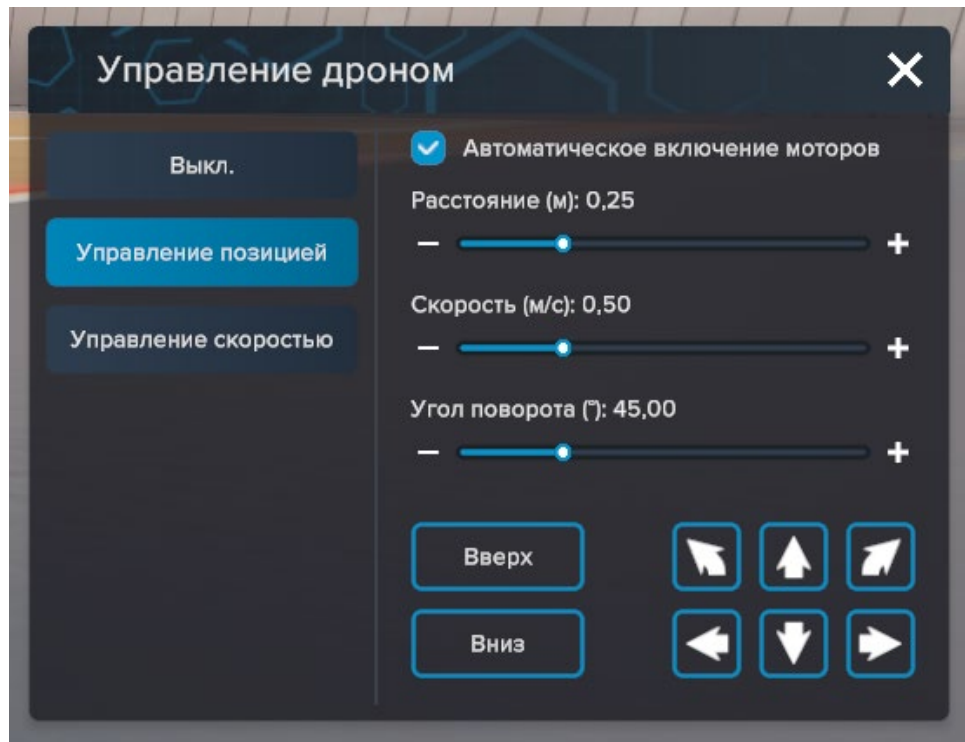
После выбора локации начнется загрузка после которой вы попадете на локацию.

Работа с программным обеспечением происходит при помощи следующих инструментов:



- 1- Окно просмотра изображения со стереокамеры
- 2- Кнопка перезапуска упражнения
- 3- Кнопка вызова настроек SLAM
- 4- Кнопка вызова инструментов управления
- 5- Строка состояния дрона
- 6- Строка компаса
- 7- Блок с текущей информацией о позиции
- 8- Блок с горячими клавишами

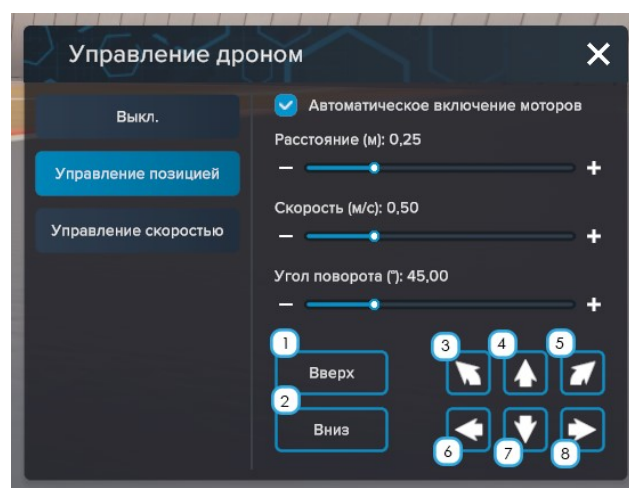
При нажатии на кнопку 4, вызывается меню управления летательным аппаратом:



Меню управления дроном

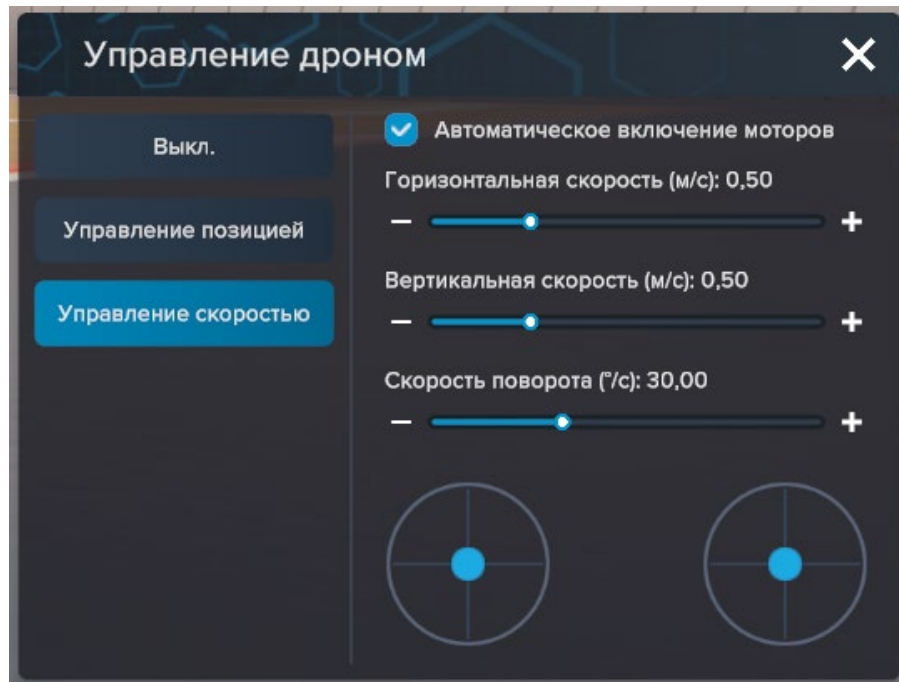
Доступно 2 варианта управления:

- Управление позицией – здесь мы можем настроить расстояние, на которое переместится дрон, скорость, с которой он это сделает, а также угол поворота.



- 1 – поднять дрон на заданную высоту
- 2 – опустить дрон на заданную высоту
- 3 – повернуть дрон влево на заданный угол
- 4- переместить дрон вперед на заданное расстояние
- 5 – повернуть дрон вправо на заданный угол

- 6- переместить дрон влево на заданное расстояние
- 7 – переместить дрон назад на заданное расстояние
- 8 – переместить дрон вправо на заданное расстояние
- Управление скоростью в этом режиме мы сможем ограничить горизонтальную, вертикальную скорость, а также скорость поворота



Управление скоростью

В этом режиме управление происходит при помощи клавиш:

W – Газ +

A - Поворот влево

S – Газ -

D – Поворот вправо

← - Крен влево

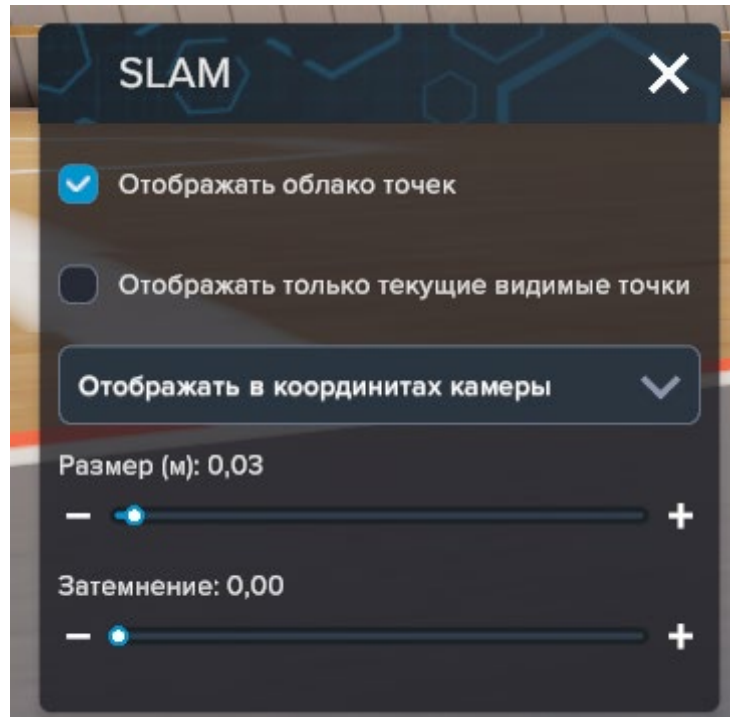
↑ - движение вперед

→ - крен вправо

↓ - движение назад

Настройки SLAM

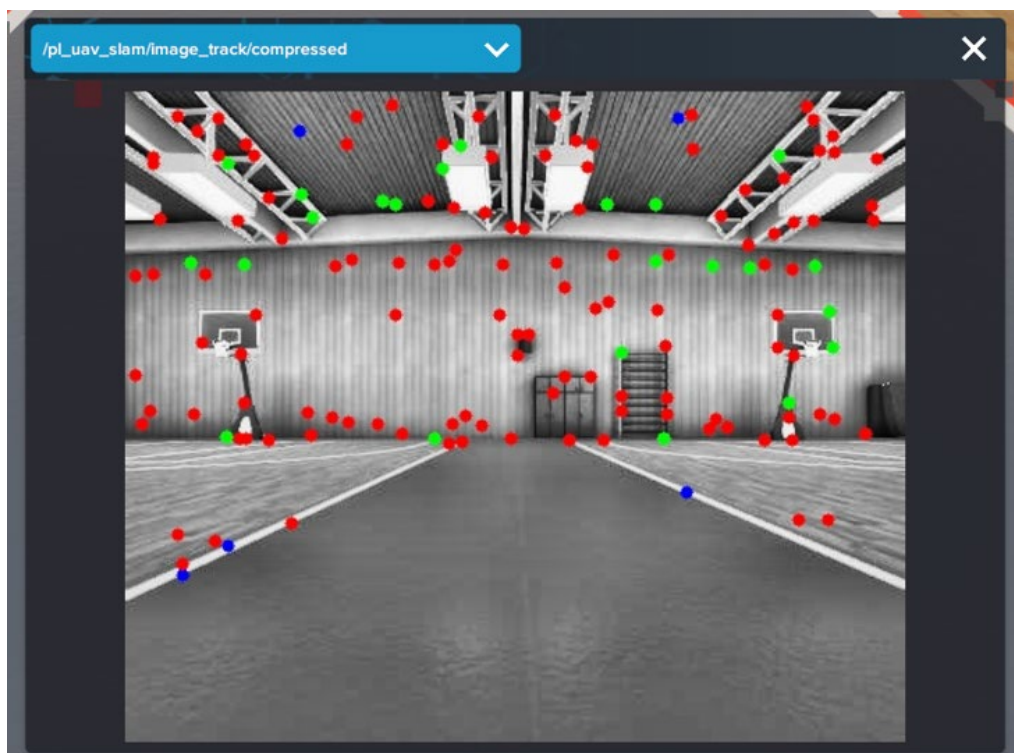
При нажатии на кнопку 3, открывается окно настройки SLAM



Окно настроек SLAM

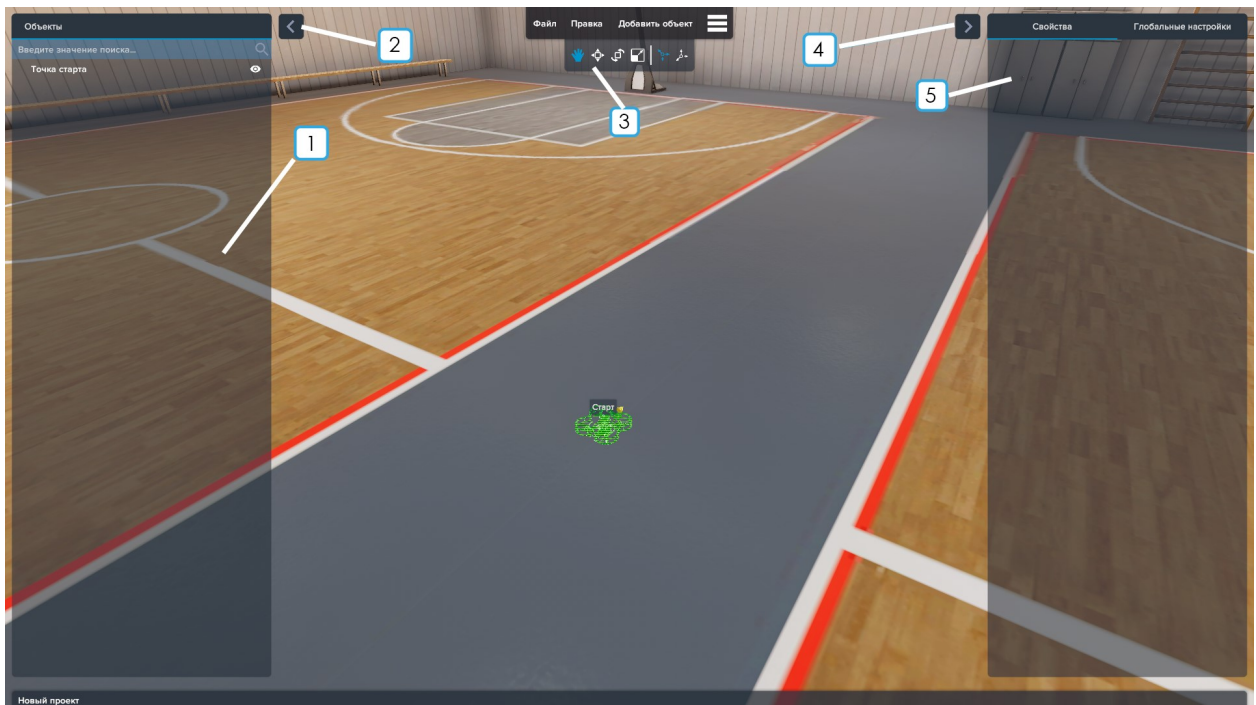
В этом окне мы можем включать/отключать облако точек, отображать только видимые(проинициализированные) точки, выбрать систему координат отображения, задавать размер и затемнение точек SLAM.

При нажатии на окно 1 откроется окно с изображением с камеры, и развернутым SLAM



Работа с комплексом в режиме редактора карт

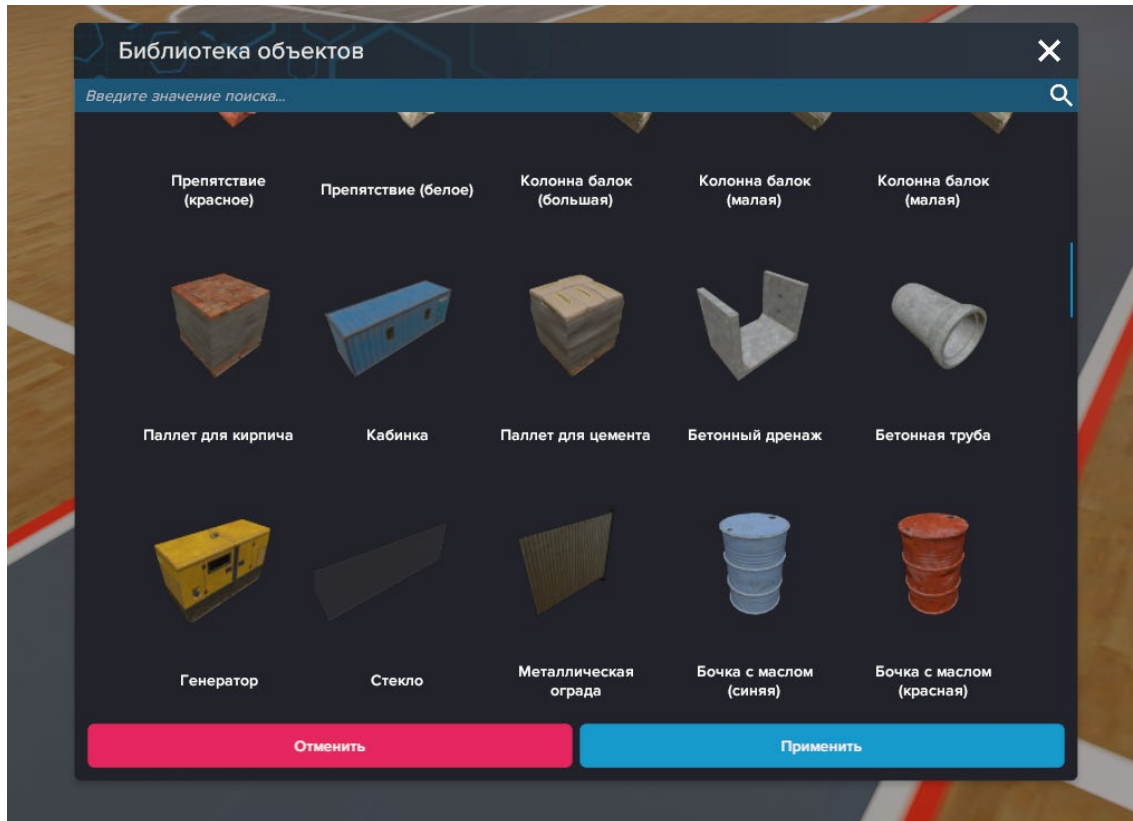
После загрузки в редактор карт вы увидите следующий интерфейс



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

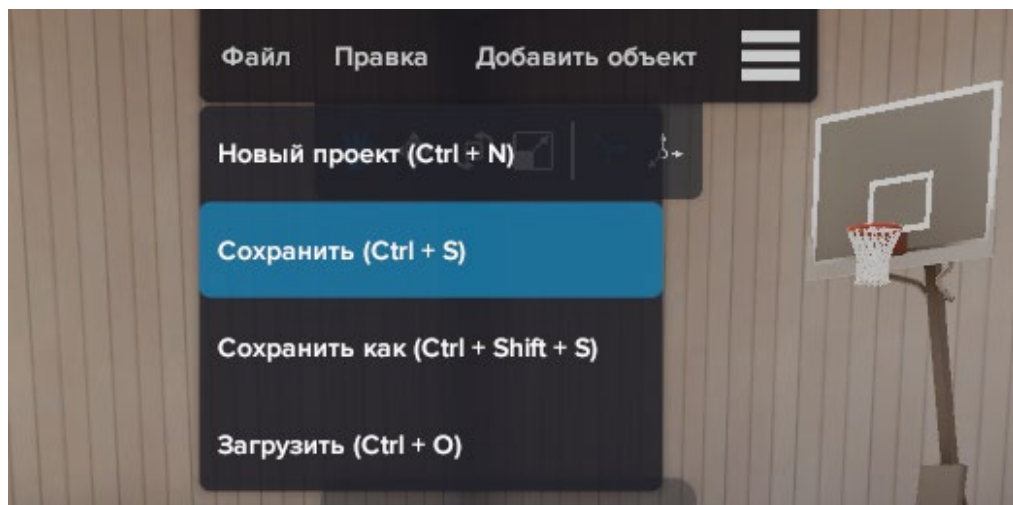
Для того чтобы разместить объект на сцену, необходимо выбрать нужный объект и нажать кнопку применить, после чего объект появится на сцене



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
- 8- Добавленный объект

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



Далее сохраненные файлы сцены можно загружать и изменять.

Работа с комплексом через веб – интерфейс

После подключения к Клеверу по адресу <http://192.168.11.1> будет доступен веб-интерфейс. В нем доступны основные веб-инструменты Клевера: просмотр топиков с изображениями и веб-терминал (Butterfly).

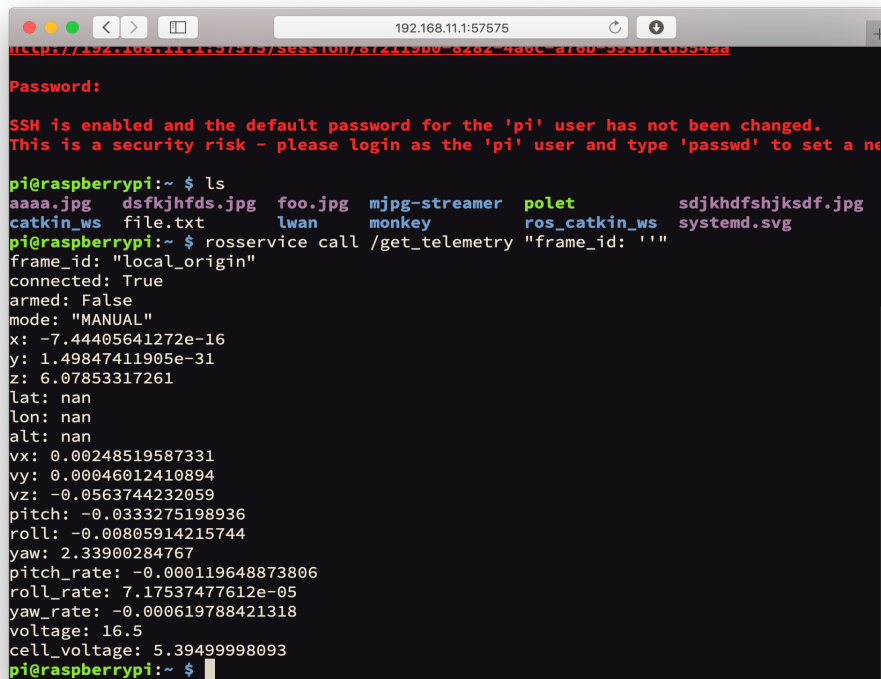
Clover Drone Kit Tools

- [View documentation](#) (snapshot of clover.coex.tech)
- [View topics](#)
- [View image topics](#) (web_video_server)
- [Open web terminal](#) (Butterfly)
- View [View 3D visualization](#), [3D visualization for markers map](#) (ros3djs)
- [Blocks programming](#) (Blockly)
- [Clover console](#) (/var/log/clover.log)

Version: v0.24

Веб - интерфейс

Для того чтобы перейти в терминал нажмите на ссылку Open web terminal, после нажатия на эту ссылку вы попадете в терминал



```

192.168.11.1:57575
http://192.168.11.1:57575/session/57211900-6262-4a6c-876b-593b7cd934aa
Password:
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

pi@raspberrypi:~$ ls
aaaa.jpg  dsfkhfds.jpg  foo.jpg  mjpg-streamer  polet  sdjkhdfshjksdf.jpg
catkin_ws  file.txt      lwan    monkey         ros_catkin_ws  systemd.svg
pi@raspberrypi:~$ rosservice call /get_telemetry "frame_id: ''"
frame_id: "local_origin"
connected: True
armed: False
mode: "MANUAL"
x: -7.44405641272e-16
y: 1.49847411905e-31
z: 6.07853317261
lat: nan
lon: nan
alt: nan
vx: 0.00248519587331
vy: 0.00046012410894
vz: -0.0563744232059
pitch: -0.0333275198936
roll: -0.00805914215744
yaw: 2.33900284767
pitch_rate: -0.000119648873806
roll_rate: 7.17537477612e-05
yaw_rate: -0.000619788421318
voltage: 16.5
cell_voltage: 5.39499998093
pi@raspberrypi:~$

```

Терминал Butterfly

Для того чтобы посмотреть информацию с топиков перейдите по ссылке [View_topics](#)

Topics:

- [/rosout_agg](#)
- [/rosout](#)
- [/mavros/vision_pose/pose](#)
- [/pl_uav_slam/vision_pose/pose](#)
- [/track_markers](#)
- [/vehicle_marker](#)
- [/wp_markers](#)
- [/landing_target](#)
- [/mavros/local_position/pose](#)
- [/mavros/setpoint_position/local](#)
- [/lt_marker](#)
- [/tf_static](#)
- [/tf](#)
- [/rangefinder/range](#)
- [/rangefinder/data](#)
- [/tf2_web_republisher/result](#)
- [/tf2_web_republisher/feedback](#)
- [/tf2_web_republisher/status](#)
- [/tf2_web_republisher/goal](#)
- [/tf2_web_republisher/cancel](#)
- [/led/state](#)
- [/diagnostics](#)
- [/mavlink/from](#)
- [/mavlink/to](#)
- [/mavros/altitude](#)
- [/mavros/distance_sensor/rangefinder](#)
- [/mavros/global_position/raw/fix](#)

Раздел с информацией со всех топиков

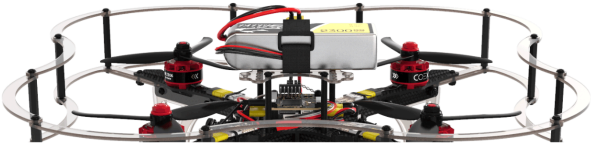
Для того чтобы посмотреть информацию с топиков с изображениями перейдите по ссылке [View_image_topics](#)

Available ROS Image Topics:

- [/secondary_camera/image_raw](#) ([Snapshot](#))

Информация о топиках с изображениями

В веб-интерфейсе приложена документация на летательный аппарат, для ее просмотра перейдите по ссылке [View_documentation](#)

<p>Введите условия поиска</p> <p>Введение</p> <p>Глоссарий</p> <p>Безопасность</p> <p>Сборка</p> <p>Настройка</p> <p>Ручной полет</p> <p>Работа с Raspberry Pi</p> <p>Программирование</p> <p>Дополнительные материалы</p> <p>Мероприятия</p> <p>Проекты на базе Клевера</p> <p>УЧЕБНИК</p> <p>Теория и видеоуроки</p> <p>Учебно-методическое пособие</p> <p>Контрольные материалы</p> <p>Опубликовано с помощью GitBook</p>	<h2 style="text-align: center;">Клевер</h2>  <p>«Клевер» — это учебный конструктор программируемого квадрокоптера, состоящего из популярных открытых компонентов, а также набор необходимой документации и библиотек для работы с ним.</p> <p>Набор включает в себя полетный контроллер COEX Pix с полетным стеком PX4, Raspberry Pi 4 в качестве управляющего бортового компьютера, модуль камеры для реализации полетов с использованием компьютерного зрения, а также набор различных датчиков и другой периферии.</p> <p>Платформа Клевера также включает в себя преднастроенный образ для Raspberry Pi в полном набором необходимого ПО для работы со всей периферией и программирования автономных полетов. Исходный код платформы Клевера и данной документации открыт и доступен на GitHub.</p> <p>Если вы детально изучили документацию, но так и не нашли ответа на свой вопрос, напишите в</p>
---	--

Документация на летательный аппарат

Управление летательным аппаратом через радиопульт FLYSKY FS-i6X.



Внешний вид радиопульта FLYSKY FS-i6X

1. Стик управления газом и рысканием.
Стик вверх – газ больше
Стик вниз – газ меньше
Стик вправо – рыскание (поворот) вправо
Стик влево – рыскание (поворот) влево
2. Тумблер Kill-switch
Тумблер вниз – активация Kill Switch
Тумблер вверх – деактивация Kill Switch
3. Свободный тумблер (на него можно назначить необходимые вам события)

4. Тумблер переключения режимов полета
Тумблер вверх – активация режима:
Тумблер посередине – активация режима
Тумблер вниз – активация режима
5. Тумблер Arm
Тумблер вниз – arm
Тумблер вверх – disarm
6. Стик управления Креном и тангажом
Стик вверх – тангаж(движение) вперед
Стик вниз – тангаж(движение) назад
Стик вправо – крен (поворот) вправо
Стик влево –крен (поворот) влево
7. Слайдер питания
Слайдер вниз – питание выкл.
Слайдер вверх – питание вкл.

Если вы хотите полетать на дроне не используя ПО, то вам необходимо:

- 1) Подключить к дрону АКБ
- 2) Включить пульт при помощи слайдера 7
- 3) Перевести тумблер 5 в нижнее положение что будет соответствовать команде ARM
- 4) Используя стик 1 добавить газ до взлета

Работа с полетом

10.1. Безопасность при подготовке к вылету

- Убедиться, что Li-ion аккумуляторы заряжены;
- Убедиться, что батарейки в аппаратуре управления заряжены;
- Устанавливать пропеллеры только перед вылетом.

Проверить надёжность следующих узлов:

- Затянутость гаек пропеллеров;
- Крепление и целостность защит винтов;
- Надёжность крепления проводов, отсутствие болтающихся проводов.

10.2. Безопасность перед вылетом

- Располагать зрителей за спиной пилота или за линией, проходящей через оба плеча пилота за спиной пилота;
- Не допускать выхода зрителей в полусферу перед лицом пилота;
- Знать и помнить время полёта, на которое рассчитан данный коптер и его аккумулятор;
- ДО подключения Li-ion аккумулятора включить аппаратуру управления (пульт), перевести левый стик (газ) в нулевое положение;
- Подключать Li-ion аккумулятор только перед взлётом, отключать сразу после взлёта;
- Стоять на расстоянии не менее 3 м от коптера;
- Взлетать с земли с ровной площадки, на расстоянии не менее 3 метров от препятствий.

10.3. Безопасность в полете

- Выполнять все указания преподавателя или лётного инструктора;
- Заранее обозначить зону пилотажа. Летать только в обозначенной зоне и не допускать вылета за её пределы. Не залетать за собственную спину;
- При обучении полётам летать на уровне ниже собственного роста;
- Летать рядом с собой на расстоянии, на котором вам видна ориентация коптера в пространстве. Не улетать далеко от себя. В случае сомнений в ориентации коптера немедленно выполнить посадку на месте. Не пытаться взлететь. Подойти ближе к коптеру и выполнить взлёт;
- При управлении все движения стиками выполнять аккуратно и плавно. Не допускать резких движений. При необходимости изменить направление полёта двигать стиками следует энергично, но не резко;

- Летать следует осторожно и выполнять только те элементы, в которых нет сомнений. Запрещается выполнять фигуры пилотажа, в успехе которых возникают сомнения и фигуры, связанные с риском;
- Соблюдать скоростной режим. Скорость полёта коптера держать в пределах скорости идущего человека;
- Вернуть коптер к месту посадки к рассчитанному времени, не допускать полной разрядки аккумулятора в полёте;
- Посадку выполнять только на ровную открытую площадку вдали от препятствий.

10.4. Аварийная посадка

В случае удара об землю или жесткой посадки выполнить следующие действия:

1. Прекратить полёт. Посадить коптер на землю. Левый стик (газ) в минимум;
2. Disarm (левый стик влево-вниз на 3 секунды);
3. Отключить Li-ion аккумулятор на коптере;
4. Выключить пульт;
5. Осмотреть коптер и при необходимости отремонтировать.

10.5. Запланированная посадка

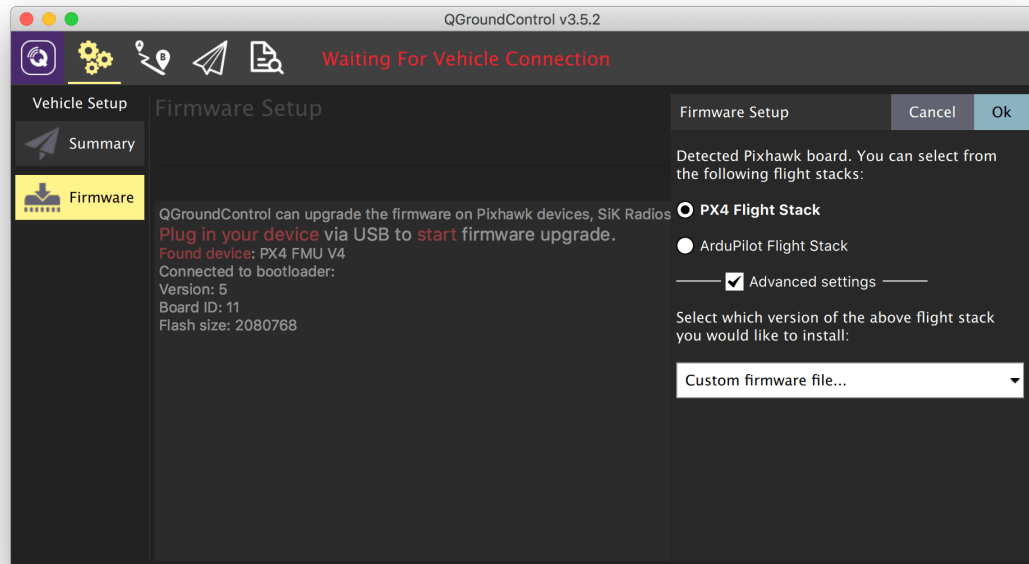
После запланированной посадки выполнить следующие действия:

1. Disarm (Левый стик влево-вниз на 3 секунды);
2. Отключить Li-ion аккумулятор на коптере;
3. Выключить пульт.

10.6. Загрузка прошивки в полетный контроллер

Скачайте актуальную версию прошивки на GitHub — [v1.8.2-clover.13](#).
(Ссылка на прошивку есть на поставляемом USB-Носителе)

1. Отключите полетный контроллер от компьютера (если он подключен).
2. Установите и запустите программу QGroundControl(Установочный файл находится на поставляемом USB носителе).
3. Перейдите в панель *Vehicle Setup* (кликнув на логотип QGroundControl в левом верхнем углу) и выберите меню *Firmware*.
4. Подключите полетный контроллер к компьютеру по USB.
5. Выберите *Advanced settings*.
6. В выпадающем меню выберите *Custom firmware file...*



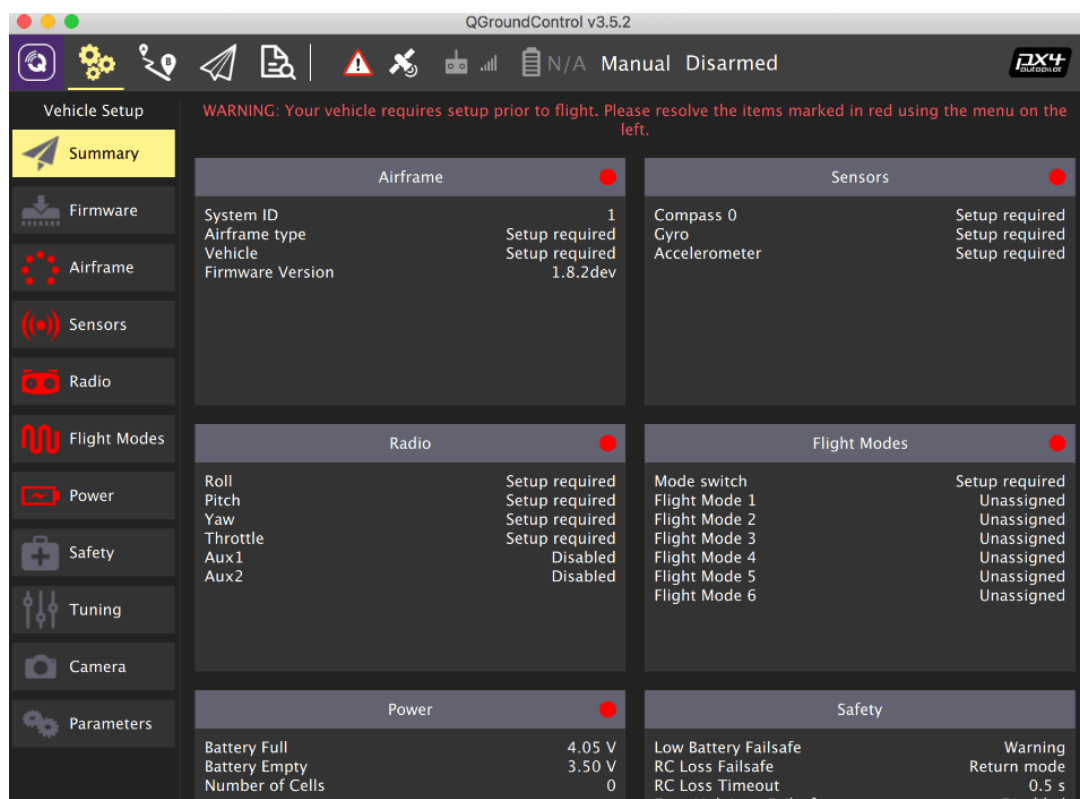
7. Нажмите **OK** и выберите скаченный файл прошивки.

Для загрузки последней версии **стандартной прошивки** сразу нажмите **OK**.

Дождитесь, пока QGroundControl загрузит прошивку и выполнит перезагрузку полетного контроллера.

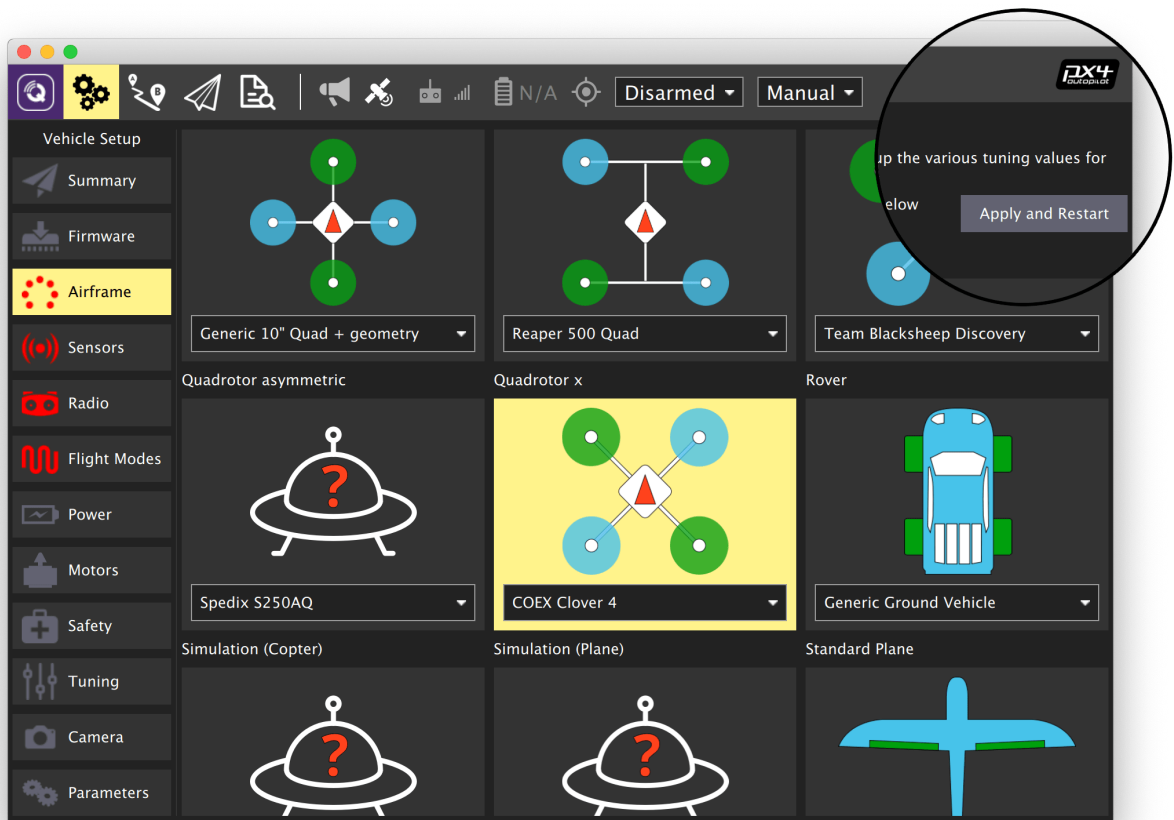
Настройка полетного контроллера

Обзор главного окна настроек QGroundControl:



1. Параметры, нуждающиеся в настройке: *Airframe*, *Radio*, *Sensors*, *Flight Modes*;
2. Текущая прошивка контроллера;
3. Текущий полетный режим;
4. Сообщения об ошибках.

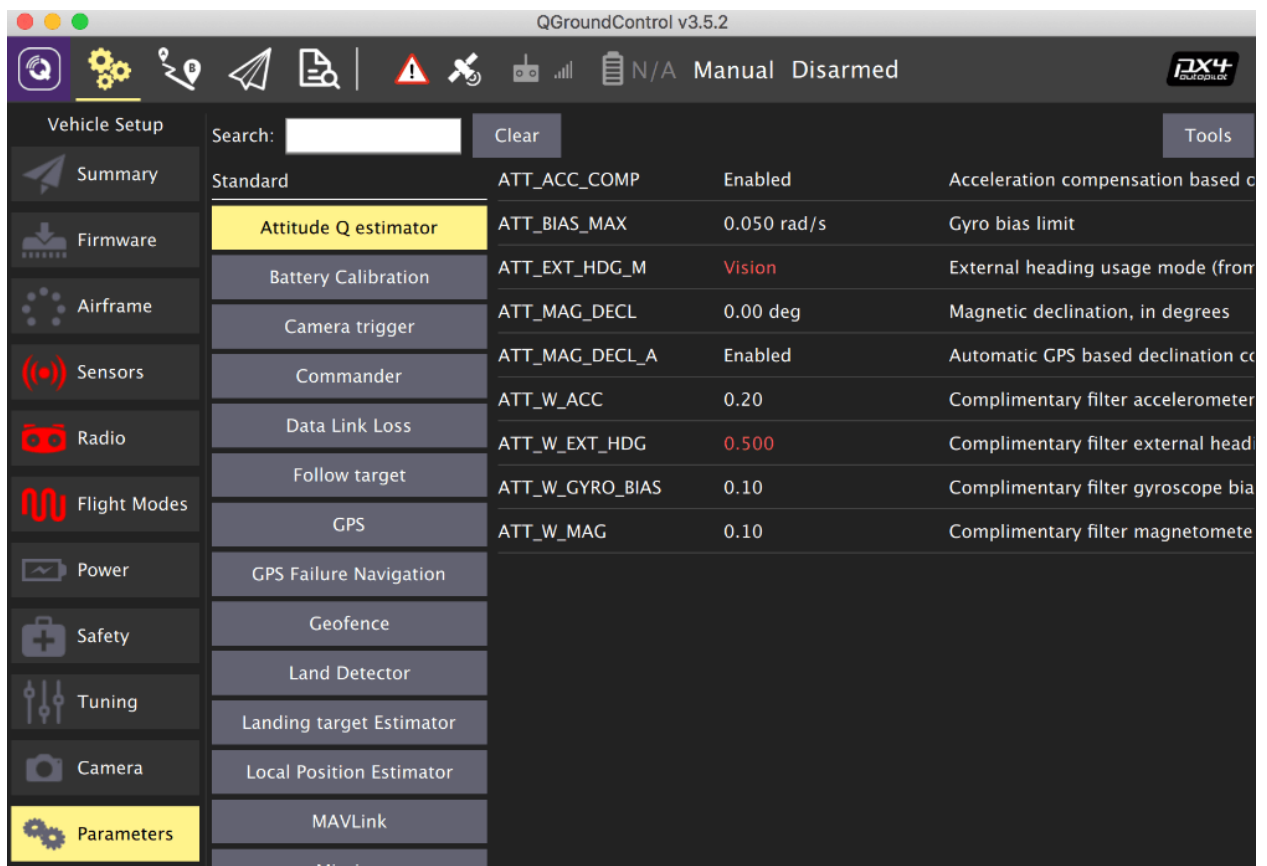
Выбор рамы



1. Зайдите во вкладку *Vehicle Setup*;
2. Выберите меню *Airframe*;
3. Выберите тип рамы *Quadrotor X*;
4. Для Клевера 4 выберите подтипа рамы *COEX Clover 4*. В ином случае – *Generic Quadrotor X*;
5. Переместитесь в начало списка и нажмите кнопку *Apply and Restart*, подтвердите нажатием *Apply*;
6. Дождитесь применения настроек и перезагрузки полетного контроллера.

Параметры

Для настройки параметров полетного контроллера войдите во вкладку *Vehicle Setup* и выберите меню *Parameters*. Вы можете использовать поле *Search* для поиска параметров по имени.



После установки параметра необходимо нажать кнопку *Save*. При необходимости – перезагрузить полетный контроллер, нажав кнопку *Tools*, затем *Reboot vehicle*.

Рекомендованные значения

Общие параметры

Параметр	Значение	Примечание
SENS_FLOW_ROT	0 (No rotation)	В случае использования "железного" PX4Flow , оставьте значение по умолчанию
SENS_FLOW_MINHGT	0.0	Для датчика VL53L1X
SENS_FLOW_MAXHGT	4.0	Для датчика VL53L1X
SENS_FLOW_MAXXR	10.0	
SYS_HAS_MAG	0	При невозможности запуска магнитометра (ошибка <i>No mags found</i>)

Настройки подсистемы Estimator

Параметр	Значение	Примечание
SYS_MC_EST_GROUP	local_position_estimator, attitude_estimator_q (unsupported)	
LPE_FUSION	84	Чекбоксы: fuse vision position, fuse land detector, flow gyro compensation
LPE_VIS_DELAY	0.1	
LPE_VIS_Z	0.1	
LPE_FLW_SCALE	1.0	
LPE_FLW_R	0.2	
LPE_FLW_RR	0.0	
LPE_FLW_QMIN	10	
ATT_W_EXT_HDG	0.5	Включение использования внешнего угла по рысканью (при навигации по карте маркеров)
ATT_EXT_HDG_M	1 (Vision)	
ATT_W_MAG	0	Выключение магнитометра (при навигации внутри помещения)

Настройка PID-регуляторов

Использование типа рамы COEX Clover 4 не требует ввода коэффициентов PID!!!

Усредненные коэффициенты PID для Клевера 4

MC_PITCHRATE_P = 0.087
MC_PITCHRATE_I = 0.037
MC_PITCHRATE_D = 0.0044
MC_PITCH_P = 8.5
MC_ROLLRATE_P = 0.05
MC_ROLLRATE_I = 0.037
MC_ROLLRATE_D = 0.0044
MC_ROLL_P = 8.5
MPC_XY_VEL_P = 0.11
MPC_XY_VEL_D = 0.013
MPC_XY_P = 1.1
MPC_Z_VEL_P = 0.24
MPC_Z_P = 1.2



Sk
Resident

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

