Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет **СУ и Р**

Образовательная программа Робототехника и искусственный интеллект

О Т Ч Е Т

о производственной практике, производственная, научно-исследовательская

Тема задания:
Получение информации с датчиков устройств с помощью ROS

Обучающийся:
Румянцев Алексей Александрович, гр. R3241

Руководитель практики от университета:
Золотаревич Валерий Павлович, инженер, доцент

Санкт-Петербург
2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время навигация устройств является неотделимой частью жизни каждого человека, предприятия, государственных служб и других объектов. Дроны, роботы-доставщики и другие не могут обойтись без алгоритмов навигации, которые должны работать с особенно высокой точностью. Однако условия не всегда позволяют в достаточной степени протестировать написанные программы на роботах, или это может быть неудобно или невыгодно. Решением проблемы является использование виртуальной симуляции. В ней на модели, основанной на используемом в реальности аппарате, можно проводить различные эксперименты, например на прочность, управляемость, стабильность и конечно корректное и точное выполнение всех алгоритмов по определению положения робота в пространстве. В Linux есть набор пакетов, которые в сумме составляют систему, называемую Robotics Operating System (далее ROS). Благодаря существующим в ней симуляционным возможностям, зачастую именно в ROS проводят различные испытания роботов. Каждое устройство взаимодействует с окружающим миром с помощью датчиков. Любые алгоритмы, в том числе ориентации, нуждаются в информации о реальном мире, которые могут получить только с этих самых датчиков. Для их настройки в симуляционном мире, а впоследствии и применении этих настроек в реальном мире, необходимо, чтобы ROS корректно получал информацию с датчиков. Таким образом можно будет увидеть и исправить любые ошибки в исполняемой датчиками работе, чтобы настроить устройство максимально точно и после использовать робота уже в реальном мире. Подключение устройств к ROS различными способами позволит тестировать код или другие значимые параметры удобным способом, соответствующим используемому аппарату. Например, квадрокоптер с помощью беспроводного подключения, а некоторый маленький неподвижный датчик по проводному. Цель – разобраться в том, как подключить телефон и контроллер полета pixhawk 2.4.8 к ноутбуку с установленным ROS, и написать инструкцию, как воспроизвести подключение.

Установка и настройка ROS. Основные команды

1. Первым делом необходимо установить Linux Ubuntu. Существуют различные версии ROS, например indigo, melodic, noetic, kinetic, а также уже давно есть ROS 2, версия которого называется humble. В ходе выполнения данной практики участниками была согласована установка ROS версии noetic, которая поддерживает только Ubuntu 20.04. Это самая актуальная версия ROS 1, так как далее разработчики решили перейти на ROS 2 на Ubuntu 22.04. Был скачан образ с официального сайта, записан на флешку с помощью rufus и установлен на ноутбук Acer. Было выделено под /home 100 Гб памяти, а под все остальное оставшиеся 400 Гб.
2. Далее производим установку ROS:
    a. Выполним следующую команду, чтобы ноутбук принимал все необходимые пакеты программного обеспечения с packages.ros.org:
    sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
    b. Нам потребуется curl и команда ниже, чтобы скачать открытый ключ для подписывания пакетов ROS и добавить его в систему управления пакетами APT:
    sudo apt install curl
    curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
    c. Установим полную версию ROS на ноутбук командами ниже, предварительно обновив информацию о пакетах. Полная версия включает в себя полезные программы, такие как rviz, gazebo, различные 2D/3D симуляторы и пакеты, а также библиотеки упаковки, сборки и связи ROS:
    sudo apt update
    sudo apt install ros-noetic-desktop-full
    d. Для корректной работы ROS каждый раз при открытии терминала необходимо прописывать специальную команду, с помощью которой будут подтягиваться переменные окружения и другие настройки ROS. Однако вместо прописывания команды вручную можно добавить ее в .bash, чтобы при открытии терминала ROS все делал автоматически:
    echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

source ~/.bashrc

e. Для создания собственных рабочих пространств ROS и управления ими существуют различные инструменты, которые распространяются отдельно. Мы установим rosinstall – часто используемый инструмент командной строки, который позволяет легко загружать множество деревьев исходного кода для пакетов ROS с помощью одной команды: *sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential*

f. Далее установим и инициализируем rosdep. Он позволяет устанавливать системные зависимости для исходного кода, который нужно скомпилировать, и необходим для запуска некоторых основных компонентов в ROS:
sudo apt install python3-rosdep
sudo rosdep init
rosdep update

g. Теперь можно полноценно использовать ROS.

3. Разберем основные понятия в ROS:
   a. Ноды (Nodes) – это процессы, которые выполняют некоторый алгоритм, заданный программистом. В ROS система разбивается на множество небольших программ (нод), каждая из которых выполняет одну задачу. Ноды могут общаться друг с другом через механизмы публикации/подписки, сервисов или действий. Ноды могут быть написаны на разных языках программирования, включая C++ и Python.
   b. Топики (Topics) – это способ асинхронной коммуникации, где одна нода публикует сообщения в топик, а другая нода подписывается на этот топик и получает сообщения. Топики идентифицируются именами, и каждый топик имеет определенный тип сообщения.
   c. Пакеты (Packages) – это основная единица организации кода в ROS. Пакет может содержать ноды, сообщения, сервисы, файлы конфигурации и другие ресурсы. Каждый пакет имеет свой файл конфигурации (package.xml), который описывает его зависимости и метаданные.
   d. Система запуска (Launch system). Файлы запуска используются для автоматического запуска набора нод и установки их параметров. Это

удобно для запуска сложных систем с множеством взаимосвязанных нод, а также для работы с несколькими воркспейсами.

e. Системы сборки (Build systems). ROS использует системы сборки для компиляции и управления зависимостями. Основной системой сборки является catkin. Для ее установки нужна следующая команда:
sudo apt-get install python3-catkin-tools

4. Разберем подробнее структуру рабочей области catkin:
   a. Каткин-пакеты (Catkin Packages) – это основная единица сборки в catkin. Каждый пакет имеет свою структуру директорий и должен содержать файл package.xml, который содержит метаданные пакета, такие как его имя, версия и зависимости. Пакет также включает CMakeLists.txt для управления процессом сборки – в нем указываются необходимые компоненты (например, roscpp, rospy) и другие параметры. Пакет может иметь директории launch и scripts, необходимые для хранения файлов запуска и программ соответственно.
   b. Рабочая область catkin (Catkin Workspace) – это директория “catkin_ws”, содержащая все ваши каткин-пакеты. Она обычно состоит из трех поддиректорий:
      1. src – в этой директории находятся каткин-пакеты. При первом создании необходимо прописать *catkin init* и *catkin build*.
      2. build – эта директория используется для хранения промежуточных файлов сборки.
      3. devel – сокращение от "development space" – здесь находятся все собранные артефакты, которые можно использовать без установки. Организован по пакетам для изолированной сборки.
   c. Для того, чтобы скомпилировать пакеты, будем использовать catkin build вместо catkin make, который предлагается в различных инструкциях. Основное отличие – изолированная среда, которую получает пользователь при сборке catkin. Это делает всю конфигурацию сборки более разделенной и устойчивой к изменениям в конфигурации (добавление/удаление пакета, изменение переменной cmake и т. д.). Кроме того, пользователь получает более структурированный и легко читаемый цветной вывод командной

строки. catkin build работает в любом месте рабочего пространства, не только в верхнеуровневой директории. Вместе с тем можно собирать пакеты по отдельности, не только все сразу. Если пользователь один раз использовал catkin make, то в дальнейшем он не сможет просто воспользоваться catkin build.

d. Также вместе с catkin мы получаем много полезных команд, таких как catkin clean для очистки; build, devel, install, catkin list и т.д.

5. Разберем несколько основных команд в ROS:
   a. *roscore* – это набор узлов и программ, которые необходимы для любой системы на базе ROS. Чтобы узлы ROS могли обмениваться данными, должен быть запущен roscore.
   b. *roslaunch* – команда для старта файла запуска .launch/XML. Синтаксис команды имеет вид:
      roslaunch package_name file.launch
   c. *rostopic list* – отображение списка текущих топиков. Необходим запущенный *roscore*. Выведет все топики, которые на данный момент существуют в сессии и к которым можно обращаться.
   d. *rosnode list* – отображение списка текущих нод. Аналогично *rostopic list*.
   e. *rqt_graph* – отображение окна, в котором автоматически строится диаграмма топиков, нод, их взаимодействия и т.д.

Получение информации с датчиков телефона и pixhawk 2.4.8

1. Для начала подключим к ROS телефон. Для этого понадобится специальное приложение – android_sensors_driver. В открытом доступе существуют различные версии этого приложения, которые можно клонировать на свое устройство и после компиляции получить установочный .apk файл. Однако самое новое из этих приложений для версии ROS 1 последний раз обновлялось 7 лет назад, вследствие чего исполнителям не удалось вручную через Android Studio компилировать код. Система для автоматизации сборки приложений в этих проектах устарела и необходимо найти способ “подогнать” ее и все зависимости под такую версию, в которой эта система позволит создать установочный файл. Так как данная задача требует времени для реализации, участники практики решили поискать в интернете уже

скомпилированный код и нашли .apk файл на одном сайте и установили приложение на смартфон с android версии 12. Скорее всего данное приложение не получится установить на смартфон с android версии ниже 4. Далее проделаем следующие шаги:

a. Необходимо настроить переменные среды для ROS, чтобы указать ROS Master, по какому адресу ему подключаться. Для этого нужно определить IP-адрес сети, к которой подключен ноутбук. В нашем случае он был подключен к Wi-Fi сети, раздаваемой с другого телефона. Мы нашли в настройках сети IPv4-адрес: 192.168.43.226. Зададим любой свободный порт – пусть это будет 11311. Введем в терминал команду ниже

   export ROS_MASTER_URI=http://192.168.43.226:11311/

   Теперь запускаем *roscore*. Если все в порядке, то в консоли будет указано, что ROS Master подключился по указанному в команде адресу IPv4 и порту соответственно.

b. Подключаем телефон к той же сети, к которой подключен ноутбук. Запускаем установленное приложение и в строке для ввода записываем такой же IP-адрес и порт, что задали в переменную ROS_MASTER_URI. Подтверждаем нажатием на кнопку “Ok”. Посмотреть интерфейс программы можно в приложении 1.

c. Теперь телефон и ROS должны быть связаны. Чтобы это проверить, введем в терминал уже знакомую команду *rosnode list*. Она выведет ноды установленных в смартфоне датчиков. Посмотреть вид вывода можно в приложении 2. Также проверку подключения можно осуществить командой

   rosnode ping /android_sensors_driver

d. Для прослушки и взаимодействия с нодами можно написать отдельную программу, однако для проверки достаточно ввести в терминал команду ниже, чтобы убедиться в том, что все данные с датчиков ROS получает корректно:

   rostopic echo /android/fix

   Исполнители решили вывести данные с датчиков */android/imu*. IMU, как правило, состоит из акселерометра и гироскопа, а иногда и магнитометра. Эти сенсоры предоставляют информацию о движении и

ориентации устройства. Посмотреть вывод с датчиков можно в приложении 3.

e. Проверим, что датчики работают верно – рассмотрим параметр linear acceleration, отвечающий за показания акселерометра. В состоянии покоя телефон должен показывать по одной из координат значение, близкое к значению ускорению свободного падения, а по остальным координатам значения, близкие к нулю. Значения координат зависят от положения телефона в пространстве. В приложениях 4, 5, 6 видно, что при различных положениях телефона координаты z, y, x соответственно близки к значению ускорения свободного падения в Санкт-Петербурге, а остальные имеют значения, близкие к нулю. Задача успешно выполнена.

2. Подключим к ноутбуку контроллер pixhawk 2.4.8. Проверим командой *lsusb*, что он определился системой. Далее проведем следующие шаги:

   a. Установим MAVROS – пакет, который служит мостом между беспилотными летательными аппаратами (БПЛА) на базе MAVLink и системой Robot Operating System (ROS). Нам понадобится следующая команда:

      sudo apt-get install ros-noetic-mavros ros-noetic-mavros-extras

   b. Настроим последовательное соединение. Убедимся, что pixhawk включился. Он должен мигать светом. Выполним следующую команду в терминале, чтобы выдать разрешение на доступ к последовательному порту:

      sudo usermod -a -G dialout $USER

   c. Узнаем IP-адрес сети, к которой подключен ноутбук: 192.168.190.128. Воспользуемся знакомой нам командой, где укажем полученный IPv4 адрес, чтобы запустить mavros через протокол UDP:

      roslaunch mavros apm.launch fcu_url:=udp://:14550@192.168.190.128:14555

      apm.launch – это файл запуска, предназначенный для работы с полетными контроллерами на базе прошивки APM (ArduPilot). 14550 – локальный порт, на который MAVROS будет слушать входящие сообщения от полетного контроллера. 192.168.190.128 – IP-адрес полетного контроллера. Используется адрес сети, к которой подключен

ноутбук, так как контроллер подключен к ноутбуку. 14555 – порт полетного контроллера, на который он отправляет данные.

d. С помощью команды *rostopic list* убедимся в том, что pixhawk подключился к ROS. Вывод команды можно посмотреть в приложении 7. Далее с помощью *echo* можно прослушать какие-либо из выведенных каналов. Напишем команду *rqt_graph*, чтобы посмотреть связи между топиками и нодами – результат можно посмотреть в приложении 8. Задача успешно выполнена.

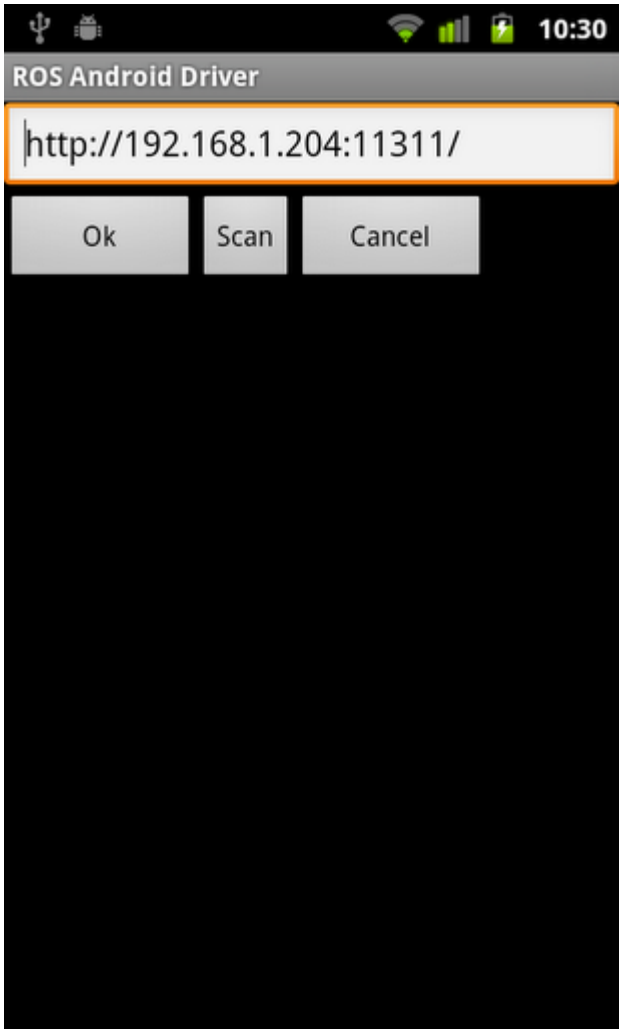
ВЫВОДЫ

В ходе выполнения практики мы повторили основы операционной системы Linux, а также получили некоторые новые знания про работу данной ОС. Мы установили Robotics Operating System и сумели подключить к нему телефон и контроллер pixhawk. Мы достигли поставленной цели и получили результаты в виде считывания данных с датчика телефона и построения графа зависимостей топиков и нод для контроллера. В конце мы подвели итоги, создали план практики и написали отчет, в котором подробно изложили инструкцию для воспроизведения всех наших шагов.

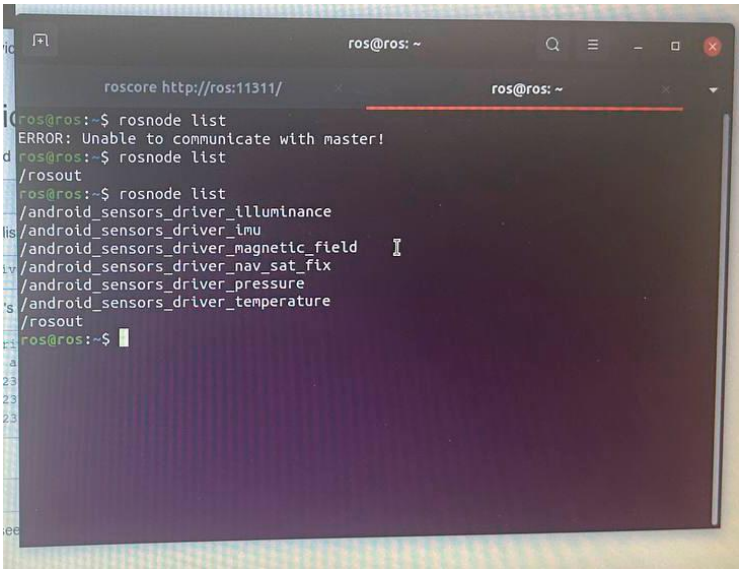
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://wiki.ros.org/noetic/Installation/Ubuntu
2. https://wiki.ros.org/roscore
3. https://wiki.ros.org/roslaunch
4. https://wiki.ros.org/rosnode
5. https://docs.ros.org/en/api/rqt_graph/html/
6. https://wiki.ros.org/android_sensors_driver
7. https://github.com/ros-android/android_sensors_driver
8. https://github.com/rpng/android_sensors_driver
9. https://docs.px4.io/main/en/ros/mavros_installation.html

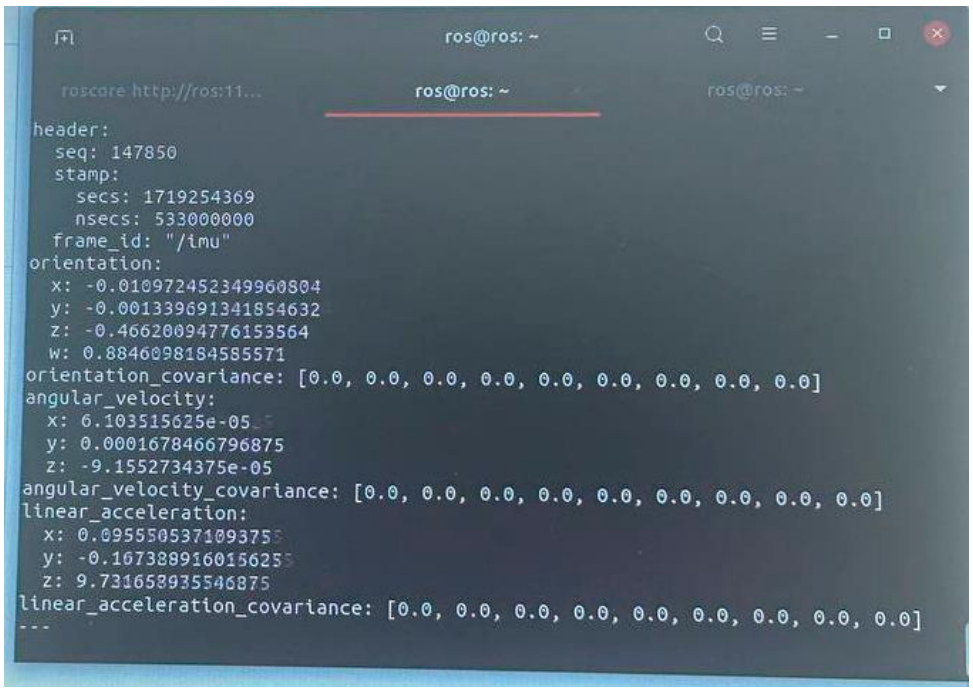
ПРИЛОЖЕНИЯ



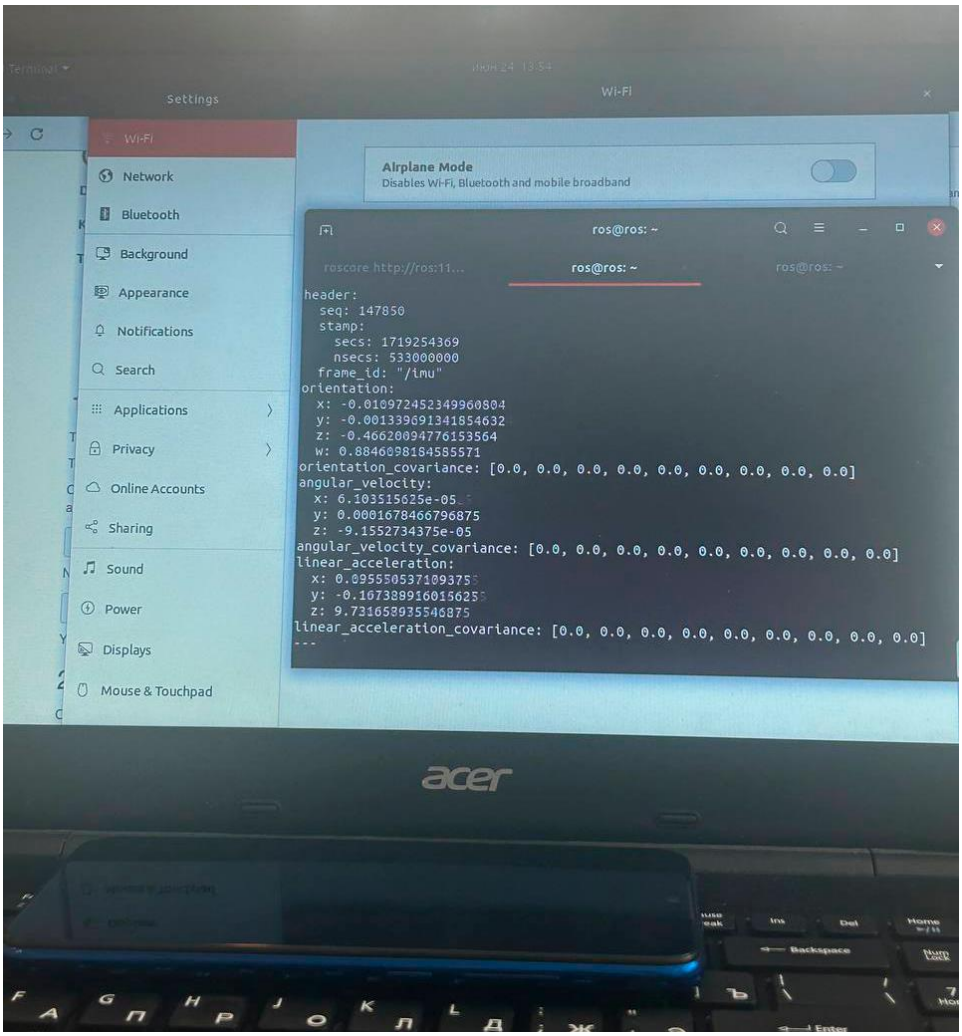
Приложение 1



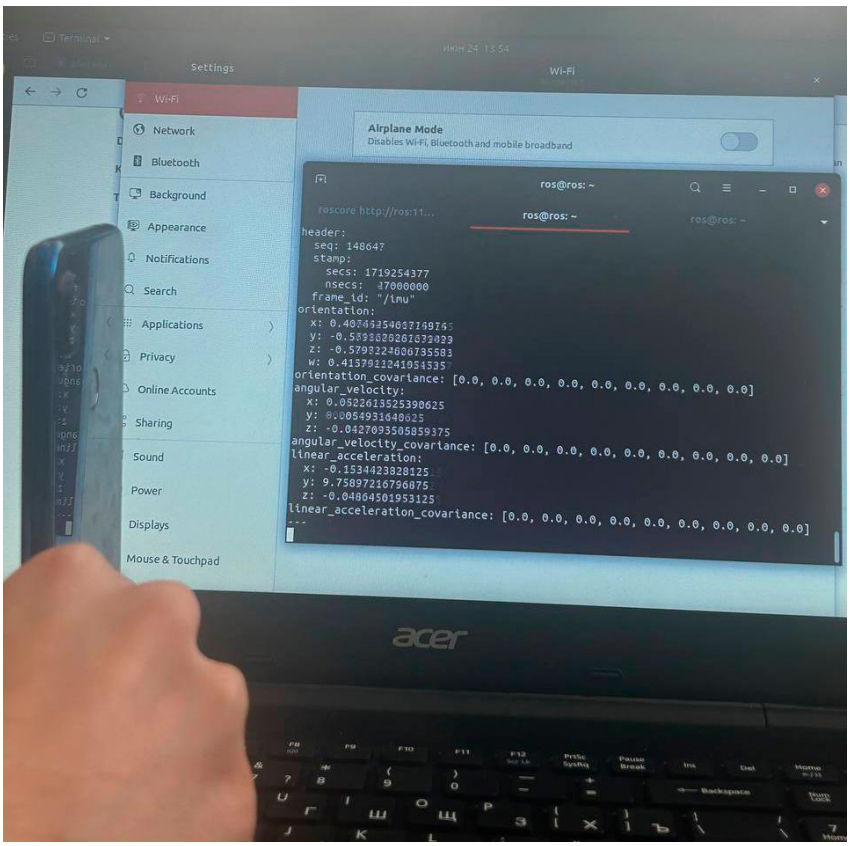
Приложение 2



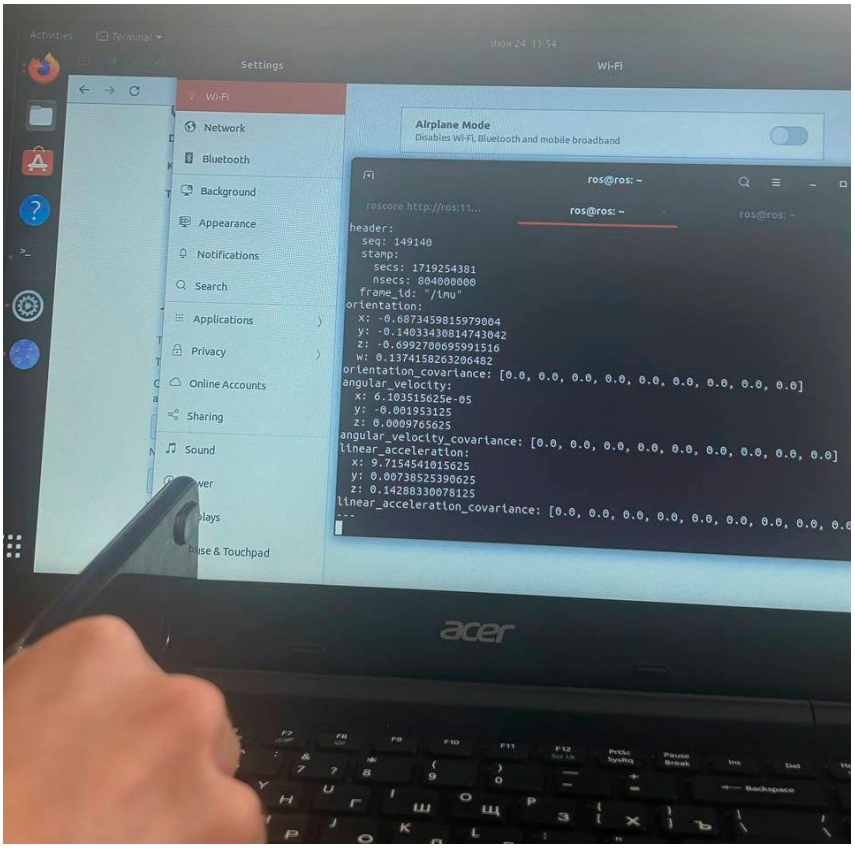
Приложение 3



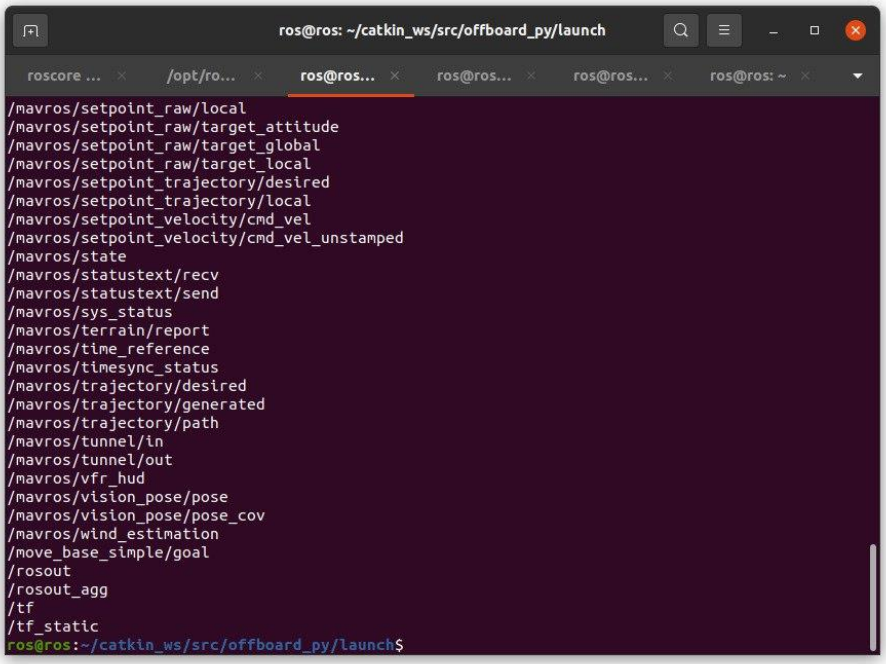
Приложение 4



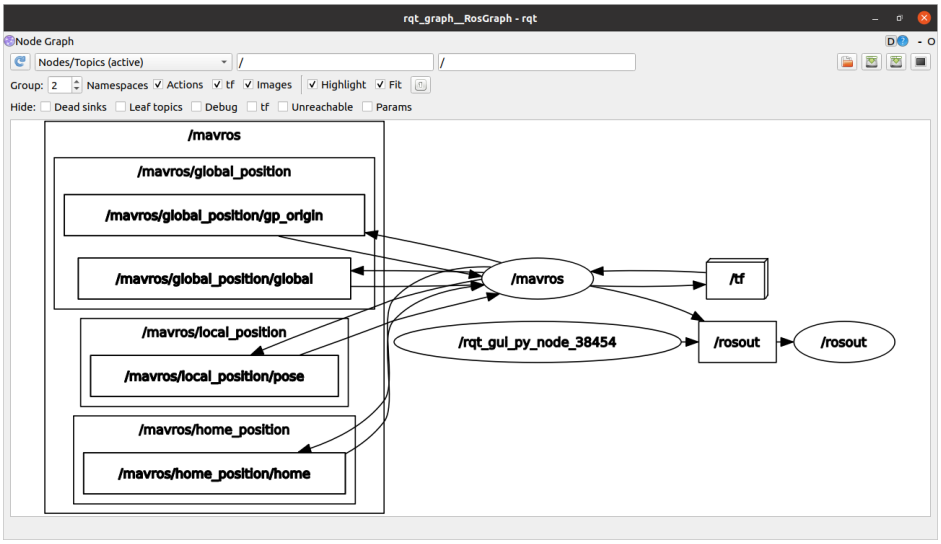
Приложение 5



Приложение 6



Приложение 7



Приложение 8