МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний авіаційний університет

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра прикладної математики

Лабораторна робота

З дисципліни «Супровід підсистем цільового навантаження БПС»

Виконала:

Студентка групи ПМ-451

Шаповалова С. В.

Перевірив:

Асистент кафедри

Сорокопуд В.І

2022

Теоретичні відомості

MAVSDK — це набір бібліотек для різних мов програмування для взаємодії з системами MAVLink , такими як дрони, камери або наземні системи.

Бібліотеки надають простий API для керування одним або кількома транспортними засобами, надання програмного доступу до інформації про транспортний засіб і телеметрії, а також керування місіями, пересуванням та іншими операціями.

Бібліотеки можна використовувати на борту дрона на комп'ютері-компаньйоні або на землі для наземної станції чи мобільного пристрою.

MAVSDK є кросплатформним: Linux, macOS, Windows, Android та iOS.

MAVSDK в основному написаний на C++ з обгортками, доступними для кількох мов програмування:

- MAVSDK-C++ (2016): Використовується у виробництві.
- MAVSDK-Swift (2018): Використовується у виробництві.
- MAVSDK-Python (2019): Використовується у виробництві.
- MAVSDK-Java (2019): Використовується у виробництві.
- MAVSDK-Go (2020): функція завершена.
- MAVSDK-JavaScript (2019): Доказ концепції.
- MAVSDK-CSharp (2019). Доказ концепції.
- MAVSDK-Rust (2019): Доказ концепції.

Плагін Offboard MAVSDK надає простий API для керування транспортним засобом за допомогою заданих значень швидкості та повороту. Це корисно для завдань, які вимагають безпосереднього керування з комп'ютера-компаньйона; наприклад, щоб реалізувати уникнення зіткнень.

API використовує позабортний режим польоту PX4 . Цей клас можна використовувати лише з вертольотами та транспортними засобами VTOL (не з фіксованим крилом - обмеження PX4) і наразі підтримує лише команди заданих значень швидкості (PX4 додатково підтримує задані значення положення та тяги).

Код клієнта повинен вказати задане значення перед запуском позабортного режиму . Плагін Offboard автоматично повторно надсилає задані значення на частоті 20 Гц (PX4 вимагає мінімального пересилання заданих значень на частоті 2 Гц). Якщо потрібне більш точне керування, клієнти можуть викликати методи заданих значень з будь-якою необхідною швидкістю або щоразу, коли доступна оновлена уставка.

MAVSDK працює, надсилаючи команди MAVLink на комп'ютер польоту, щоб керувати дроном.

Коли дроном керує MAVSDK, він переводиться в «позабортовий» режим. Під час роботи в «позабортовому» режимі дрон все ще може реагувати на команди «RC_CHANNELS_OVERRIDE», якщо налаштований правильно. Це дозволяє користувачеві вручну керувати карданами, використовуючи RC-прохід від входу RC до виходів ШІМ.

Хоча команди "RC_CHANNELS_OVERRIDE" працюють одночасно з MAVSDK, вони не працюють, якщо підключено RC-передавач.

- Польотний комп'ютер реагуватиме лише на перший тип введення, який він бачить.
- Якщо польотний контролер спочатку отримує повідомлення MAVLink «RC_CHANNELS_OVERRIDE», тоді всі вхідні дані RC-передавача ігноруються, доки комп'ютер польоту не буде перезавантажено.

- Якщо польотний контролер бачить дані RC-передавача до отримання будь-яких повідомлень MAVLink «RC_CHANNELS_OVERRIDE», тоді всі повідомлення «RC_CHANNELS_OVERRIDE» ігноруються.

MAVSDK може працювати незалежно від того, яка форма введення використовується (повідомлення «RC_CHANNELS_OVERRIDE» або вхід RC-передавача), але в позабортовому режимі MAVSDK змусить льотний комп'ютер ігнорувати введення керування польотом для тяги, рискання, крену та тангажу.

Усі інші входи RC-каналу будуть враховані, якщо це доречно (наприклад, RC-канал, налаштований як прохідний RC).

MAVSDK в основному використовується розробниками як інструмент для інтеграції різних компонентів у транспортний засіб – льотної стеки, комп'ютера-супутника та периферійних пристроїв MAVLink (наприклад, камер). Його також можна використовувати для реалізації функціональних можливостей наземної станції, характерних для певного домену (якої зазвичай не було б у загальному GCS, як-от QGroundControl).

У цій статті буде зосереджено на простому налаштуванні за допомогою симулятора, запущеного на вашому комп'ютері (ми називаємо це «Програмне забезпечення в циклі» або «SITL»), до якого MAVSDK підключатиметься.

Обгортка Python заснована на клієнті gRPC, який спілкується з сервером gRPC, написаному на C++. Щоб використовувати обгортку Python, сервер gRPC під назвою «бекенд» має працювати в тій же системі. По суті, обгортка створюється автоматично з визначень повідомлень (протофайлів).

Операційні системи Mac OS, Linux і Windows офіційно підтримуються для розробки PX4. Спробувавши інсталяцію ланцюга інструментів у Windows та Linux (на жаль, не в Mac OS), я виявив, що Windows не рекомендується для початківців. Це тому, що процес встановлення може бути громіздким, якщо ви не маєте попереднього досвіду налаштування середовищ компіляції в Windows. Установка Linux здається простішою та має кращу офіційну підтримку.

Очевидно, Linux широко використовується як переважна платформа для розробки PX4. Спеціальний приклад, представлений у цій статті, був перевірений в Ubuntu 18.04, як нативно, так і на віртуальній машині з використанням VirtualBox на хості Windows. Він також повинен працювати в Ubuntu 16.04. Також доступний контейнер Docker з попередньо встановленим набором інструментів розробки. Однак на момент публікації цієї статті вона підтримувалася лише для Linux.

Тепер давайте розглянемо, як налаштувати ланцюжок інструментів розробки в Ubuntu 18.04. Він включатиме стек польотів PX4 SITL з симуляторами jMAVSim і Gazebo, бібліотеку MAVSDK-Python і програмне забезпечення наземної станції QGroundControl. Я не буду вдаватися в дрібні деталі всіх етапів. Тим не менш, до вихідного коду цієї статті додається текстовий файл INSTALL_RUN_SIMULATION.md , який описує всі кроки, які я зробив для встановлення ланцюжка інструментів. Він також містить посилання на офіційні сторінки PX4 з детальними процедурами. Усі такі файли доступні на веб-сторінці з кодом статті та файлами Circuit Cellar .

Якщо у вас під рукою є лише машина Windows, ви можете використовувати VirtualBox для створення віртуальної машини Linux Ubuntu всередині Windows. Перегляньте розділ РЕСУРСИ в кінці статті для деяких рекомендованих веб-посилань щодо інструментів, процедур та концепцій, викладених у цій статті. Коли у вас буде готова операційна система Ubuntu, виконайте ці кроки, щоб налаштувати ланцюг інструментів розробки. Спочатку встановіть середовище моделювання PX4 SITL jMAVSim/Gazebo. Можуть бути певні передумови, перевірте текстовий файл INSTALL_RUN_SIMULATION.md, щоб отримати докладні кроки.

jMAVSim

jMAVSim — це простий мультироторний/квад-симулятор, який дозволяє вам керувати транспортними засобами типу вертольотів під керуванням PX4 навколо імітованого світу. Його легко налаштувати, і його можна використовувати для перевірки того, чи може ваш автомобіль злітати, літати, приземлятися та належним чином реагувати на різні умови збою (наприклад, збій GPS).

Налаштування jMAVSim включено в наші стандартні інструкції зі збірки (для macOS, Ubuntu Linux, Windows).

Програмне забезпечення в Loop Simulation запускає всю систему на головній машині та імітує автопілот. Він підключається через локальну мережу до симулятора. Налаштування виглядає так:

Щоб розширити або налаштувати інтерфейс моделювання, відредагуйте файли в папці Tools/jMAVSim . Доступ до коду можна отримати через репозиторій jMAVSim (відкриває нове вікно)на Github.

Система збірки примусово перевіряє правильний підмодуль для всіх залежностей, включаючи симулятор. Він не перезаписує зміни у файлах у каталозі, однак, коли ці зміни фіксуються, підмодуль потрібно зареєструвати в сховищі прошивки з новим хешем фіксації. Щоб зробити це, git add Tools/jMAVSimi зафіксуйте зміни. Це оновить хеш GIT симулятор

Gazebo

Gazebo - це потужне середовище 3D-симуляції для автономних роботів, яке особливо підходить для тестування уникнення об’єктів та комп’ютерного зору. На цій сторінці описано його використання з SITL та одним транспортним

засобом. Альтанка також може використовуватися з HITL та для моделювання кількох транспортних засобів .

Після завершення інсталяції процедура залишить вас у підкаталозі компіляції за замовчуванням, який є ~/src/Firmware . З цього підкаталогу ви можете компілювати та запускати стек польотів PX4 SITL за допомогою симуляторів транспортних засобів jMAVSim і Gazebo. Наприклад, щоб зібрати та запустити PX4 SITL з Gazebo, в тому ж командному терміналі запустіть make px4_sitl gazebo. Можливо, компіляція може бути невдалою через відсутність деяких бібліотек у вашій інсталяції Ubuntu. Перевірте файл INSTALL_RUN_SIMULATION.md про те, як виправити деякі помилки компіляції, які ви можете отримати.

Коли компіляція завершиться без помилок, симулятор Gazebo запуститься і покаже змодельований квадрокоптер 3DR Iris+. У тому ж командному терміналі запустіть commander takeoff, і ви побачите, як дрон починає злітати на висоту за замовчуванням. У деяких випадках квадрокоптер приземлиться сам через кілька секунд. Якщо ні, бігайте commander land , щоб він приземлився у вихідне положення. Щоб зупинити моделювання, натисніть <Ctrl+c> у командному терміналі. Ви також можете спробувати PX4 SITL за допомогою симулятора jMAVSim, запустивши make px4_sitl jmavsim. На малюнку 2 показано моделювання jMAVSim після зльоту командира.

Далі встановіть MAVSDK-Python. Завантажте та встановіть QGroundControl та клонуйте репозиторій MAVSDK-Python GitHub, щоб запустити приклади, описані в наступному розділі. Ці останні кроки прості (докладнішу інформацію див. у файлі INSTALL_RUN_SIMULATION.md). Якщо вам вдалося досягти цього моменту, ви готові спробувати деякий код Python, щоб змусити квадрокоптер літати сам!

Асинхронне програмування PYTHON

Ще одна характеристика, на яку варто звернути увагу в прикладах, які ми щойно запустили, — це використання бібліотеки asyncio Python та відповідних їй async/awaitключових слів. Навіть якщо ви раніше програмували на Python, є ймовірність, що ви ніколи не використовували бібліотеку асинхронного програмування asyncio . Під час використання цієї бібліотеки ключове слово async дозволяє нам оголосити асинхронну функцію, яку зазвичай називають «корутиною».

По суті, сопрограма — це функція, виконання якої можна призупинити в певний момент — як правило, для очікування певної події або даних, від яких залежить їх подальше виконання. Іншими словами, асинхронні функції можуть зупинитися, дочекавшись якоїсь події, але без блокування інших завдань у тій же програмі. Отже, бібліотека Python asyncio з її синтаксисом async/await дозволяє паралельне асинхронне програмування.

Підсумовуючи, ви оголошуєте функцію async , щоб зробити її сопрограмою, і в деякій частині її тіла, використовуючи await ключове слово, функція спільно передає контроль назад у цикл подій. Потім він очікує на деяку зовнішню подію — наприклад, надходження даних, необхідних для відновлення виконання завдання, або повернення значення з іншої функції.

Підтримувані транспортні засоби: Quad (Iris і Solo , Hex (Typhoon H480), загальний qua Налаштування альтанки 9 включено в наші стандартні інструкції зі збірки:

- macOS: середовище розробки на Mac
- Linux: середовище розробки на Ubuntu LTS / Debian Linux > Цілі Gazebo, JMAVSim і NuttX (Pixhawk)
- Windows: не підтримується.d Delta VTOL , Tailsitter, Літак, Rover, Submarine/UUV.

Практична частина

MAVSDK на Python

Для початку потрібно встановити Python версії 3.6 або новіші, щоб дізнатись своюю версію Python запустіть *python –version* або *python3 –version* в терміналі, щоб перевірити встановлену версію.

Також потрібен запущений екземпляр SITL (jMAVSim , gazebo , …)

Щоб встановити модуль, просто запустіть:

pip3 install mavsdk

Пакет mavsdk_serveryже містить (раніше називався «бекенд»), який запускається автоматично при підключенні (наприклад,). Погляньте на приклади, щоб побачити, як це використовується на практиці. Це буде щось накшталт: await drone.connect()

Розробка коду программи:

Для початку вирішуємо проблеми кодування Windows із Python та залишаємо примітку у вигляді коментаря у коді:

```
# -*- coding: utf-8 -*-

"""
Застереження при спробі запустити приклади в середовищі без GPS:
`drone.offboard.stop()` поверне `COMMAND_DENIED` результат, тому що це
вимагає перемикання режиму в положення HOLD, що наразі не підтримується в а
середовище без GPS.
"""
```

Далі імпортуємо бібліотеки для асинхронного програмування на Python та саму бібліотеку mavsdk, її класси та функції:

```
import asyncio

from mavsdk import System
from mavsdk.offboard import (OffboardError, PositionNedYaw)
```

Створюємо основну функцію запуску, записуємо коментарі, викликаємо функцію System із класса mavsdk, передаємо необхідні параметри такі як системний адрес.

```
async def run():
    """ Керування допомогою координат NED положення """

    drone = System()
    await drone.connect(system_address="udp://:14540")
```

Примітка: System() приймає два іменовані параметри: mavsdk_server_addressi port. Якщо залишено порожнім, вони за замовчуванням мають Nonei 50051, відповідно, і запускаються . Якщо встановлено (наприклад, «localhost»), то не запускатиме вбудований і намагатиметься підключитися до сервера, що працює за цією адресою. Це корисно для платформ, де він не вбудований, для цілей налагодження та для запуску в місці, відмінному від того, де запускається сценарій MAVSDK-Python.mavsdk_server -p 50051await drone.connect()mavsdk_server_addressawait drone.connect()mavsdk_servermavsdk_servermavsdk_server

Далі за допомогою цикла та параметрів дрона із відповідного класса drone підключаємось до пристрою (дрона), якщо підключення вдалось виводимо цю інформацію на екран та завершаємо цикл підключення

```
    print("Очікування підключення дрона...")
    async for state in drone.core.connection_state():
        if state.is_connected:
            print(f"-- Підключено!")
            break
```

Оцінюємо глобальну позицію дрона:

```
    print("Очікуємо, поки дрон отримає оцінку глобальної позиції...")
    async for health in drone.telemetry.health():
        if health.is_global_position_ok and health.is_home_position_ok:
            print("-- Глобальна оцінка позиції завершена")
            break
```

Після чого якщо все вірно спрацювало ми активуємо дрон, встановлюємо початкову позицію та задаємо що це «нульові координати» для зручності подальшого використання:

```
    print("-- Активація")
    await drone.action.arm()

    print("-- Встановлення початкової уставки")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, 0.0, 0.0))
```

Після цього можемо починати працювати із функцією ‘offboard’ ('поза бортом'), пробуємо активувати цю функцію, у випадку невдачі ігнорюємо помилку, виключаємо дрон та завершуємо функцію безрезультатно, також виводимо значення помилки у консоль, щоб була можливість дізнатись назву помилки та швидко виправити її:

```
    print("-- Початок функції offboard ('поза бортом')")
    try:
        await drone.offboard.start()
    except OffboardError as error:
        print(f"Не вдалося запустити позабортовий режим із кодом помилки: {error._result.result}")
        print("-- Дизактивація")
        await drone.action.disarm()
        return
```

Усе готово, на цьому етапі можна починати переміщення дрона завдаючи необхідні координати та невелику затримку у програмі для польоту:

```
    print("-- Переміщення 0 м на північ, 0 м на схід, -5 м вниз у межах місцевої системи координат")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, -5.0, 0.0))
    await asyncio.sleep(10)

    print("-- Переміщення 5 м на північ, 0 м на схід, -5 м вниз у межах місцевої системи координат, повертаємось обличчям на схід")
    await drone.offboard.set_position_ned(PositionNedYaw(5.0, 0.0, -5.0, 90.0))
    await asyncio.sleep(10)

    print("-- Переміщення 5 м на північ, 10 м на схід, -5 м вниз у межах місцевої системи координат")
    await drone.offboard.set_position_ned(PositionNedYaw(5.0, 10.0, -5.0, 90.0))
    await asyncio.sleep(15)

    print("-- Переміщення 0 м на північ, 10 м на схід, 0 м вниз у межах місцевої системи координат, повертаємось обличчям на південь")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 10.0, 0.0, 180.0))
    await asyncio.sleep(10)
```

Після польоту обов’язково потрібно деактивувати дрон, викликаємо функцію деактивації, при невдачі виводимо код помилки для швидкого виправленння неполадностей:

```
    print("-- Дизактивація")
    try:
        await drone.offboard.stop()
    except OffboardError as error:
        print(f"Не вдалося зупинити позабортовий режим із кодом помилки: {error._result.result}")
```

Це кінець функції, основна функція запуску та управління дроном готова, вона виглядає наступним чином:

```
async def run():
    """ Керування допомогою координат NED положення """

    drone = System()
    await drone.connect(system_address="udp://:14540")

    print("Очікування підключення дрона...")
    async for state in drone.core.connection_state():
        if state.is_connected:
            print(f"-- Підключено!")
            break

    print("Очікуємо, поки дрон отримає оцінку глобальної позиції...")
    async for health in drone.telemetry.health():
        if health.is_global_position_ok and health.is_home_position_ok:
            print("-- Глобальна оцінка позиції завершена")
            break

    print("-- Активація")
    await drone.action.arm()

    print("-- Встановлення початкової уставки")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, 0.0, 0.0))

    print("-- Початок функції offboard ('поза бортом')")
    try:
        await drone.offboard.start()
    except OffboardError as error:
        print(f"Не вдалося запустити позабортовий режим із кодом помилки: {error._result.result}")
        print("-- Дизактивація")
        await drone.action.disarm()
        return

    print("-- Переміщення 0 м на північ, 0 м на схід, -5 м вниз у межах місцевої системи координат")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, -5.0, 0.0))
    await asyncio.sleep(10)

    print("-- Переміщення 5 м на північ, 0 м на схід, -5 м вниз у межах місцевої системи координат, повертаємось обличчям на схід")
    await drone.offboard.set_position_ned(PositionNedYaw(5.0, 0.0, -5.0, 90.0))
    await asyncio.sleep(10)

    print("-- Переміщення 5 м на північ, 10 м на схід, -5 м вниз у межах місцевої системи координат")
    await drone.offboard.set_position_ned(PositionNedYaw(5.0, 10.0, -5.0, 90.0))
    await asyncio.sleep(15)

    print("-- Переміщення 0 м на північ, 10 м на схід, 0 м вниз у межах місцевої системи координат, повертаємось обличчям на південь")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 10.0, 0.0, 180.0))
    await asyncio.sleep(10)

    print("-- Дизактивація")
    try:
        await drone.offboard.stop()
    except OffboardError as error:
        print(f"Не вдалося зупинити позабортовий режим із кодом помилки: {error._result.result}")
```

Далі створюємо спеціальну умову в пайтоні(*if __name__ == "__main__":*), яка дозволяє нам починати виконувати код у її межах та у правильному порядку який ми задаємо. У цій умові викликаємо основну функцію управління дроном як написано у документації бібліотеки mavsdk:

```
72  if __name__ == "__main__":
73
74      loop = asyncio.get_event_loop()
75      loop.run_until_complete(run())
```

Робоча программа для управління дроном готова, повний код можна побачити на наступній сторінці.

```python
# -*- coding: utf-8 -*-

"""
Застереження при спробі запустити приклади в середовищі без GPS:
`drone.offboard.stop()` поверне `COMMAND_DENIED` результат, тому що це
вимагає перемикання режиму в положення HOLD, що наразі не підтримується в а
середовище без GPS.
"""

import asyncio

from mavsdk import System
from mavsdk.offboard import (OffboardError, PositionNedYaw)


async def run():
    """ Керування допомогою координат NED положення """

    drone = System()
    await drone.connect(system_address="udp://:14540")

    print("Очікування підключення дрона...")
    async for state in drone.core.connection_state():
        if state.is_connected:
            print(f"-- Підключено!")
            break

    print("Очікуємо, поки дрон отримає оцінку глобальної позиції...")
    async for health in drone.telemetry.health():
        if health.is_global_position_ok and health.is_home_position_ok:
            print("-- Глобальна оцінка позиції завершена")
            break

    print("-- Активація")
    await drone.action.arm()

    print("-- Встановлення початкової уставки")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, 0.0, 0.0))

    print("-- Початок функції offboard ('поза бортом')")
    try:
        await drone.offboard.start()
    except OffboardError as error:
        print(f"Не вдалося запустити позабортовий режим із кодом помилки: {error._result.result}")
        print("-- Дизактивація")
        await drone.action.disarm()
        return

    print("-- Переміщення 0 м на північ, 0 м на схід, -5 м вниз у межах місцевої системи координат")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 0.0, -5.0, 0.0))
    await asyncio.sleep(10)

    print("-- Переміщення 5 м на північ, 0 м на схід, -5 м вниз у межах місцевої системи координат, повертаємось обличчям на схід")
    await drone.offboard.set_position_ned(PositionNedYaw(5.0, 0.0, -5.0, 90.0))
    await asyncio.sleep(10)

    print("-- Переміщення 5 м на північ, 10 м на схід, -5 м вниз у межах місцевої системи координат")
    await drone.offboard.set_position_ned(PositionNedYaw(5.0, 10.0, -5.0, 90.0))
    await asyncio.sleep(15)

    print("-- Переміщення 0 м на північ, 10 м на схід, 0 м вниз у межах місцевої системи координат, повертаємось обличчям на південь")
    await drone.offboard.set_position_ned(PositionNedYaw(0.0, 10.0, 0.0, 180.0))
    await asyncio.sleep(10)

    print("-- Дизактивація")
    try:
        await drone.offboard.stop()
    except OffboardError as error:
        print(f"Не вдалося зупинити позабортовий режим із кодом помилки: {error._result.result}")


if __name__ == "__main__":

    loop = asyncio.get_event_loop()
    loop.run_until_complete(run())
```

Висновок

На цій лабораторній роботі ми побачили, як відносно легко розробити автономні програми для дронів за допомогою MAVSDK-Python. Звичайно, ми повинні знати Python заздалегідь, а також встановити та правильно працювати з набором інструментів. І оскільки MAVSDK-Python використовує бібліотеку asyncio , нам також потрібно бути знайомим з асинхронним програмуванням — нічого особливого, лише загальне розуміння того, як працює паралельність і, зокрема, як використовувати ключові слова async/await.

Якщо далі поекспериментувати з API MAVSDK-Python, нам слід зробити кілька речей. По-перше, отримайте доступ до робочого простору PX4 «Slack» і форуму «Обговорення», де ми зможете отримати підтримку для всіх речей, пов’язаних з екосистемою PX4, а не тільки MAVSDK. По-друге, спробуйте вивчити всі приклади, включені в репозиторій MAVSDK-Python. Деякі з них інтуїтивно зрозумілі та прості для розуміння, тоді як інші вимагають від вас трохи більше досліджень. Також прочитайте документацію API MAVSDK, щоб отримати детальну інформацію про функції та структури даних API, використані в прикладах.