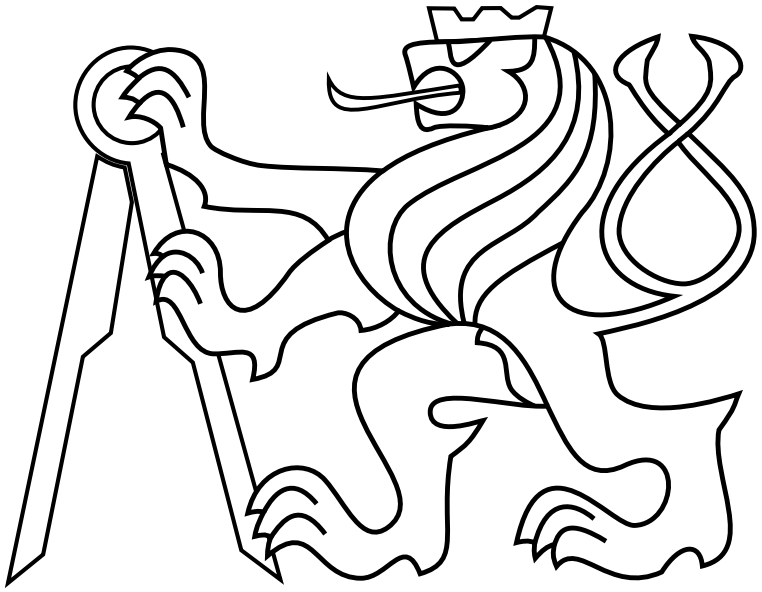
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Filip Bulander

Interface iOS for control of an unmanned helicopter in ROS

Department of Control Engineering

Thesis supervisor: **Dr. Martin Saska**

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Bulander Filip** Personal ID number: **456876**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Systems and Control**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Interface iOS for control of an unmanned helicopter in ROS

Bachelor's thesis title in Czech:

Rozhraní iOS pro řízení bezpilotní helikoptéry v ROSu

Guidelines:

The goal of the thesis is to design, implement, and experimentally verify in Gazebo simulator and real experiments an application in iOS for control an Unmanned Aerial Vehicle (UAV) equipped by onboard Linux PC with Robot Operating System (ROS).
1. Implement an interface in iOS to operate by iPhones a UAV equipped by Linux onboard computer with ROS [1,2].
2. Design and implement an iOS application for basic UAV control by iPhones (joystick, setting GPS points, displaying a UAV telemetry - position estimation, battery status, data from selected onboard sensors).
3. Verify the application in Gazebo and with a real platform in outdoor conditions.
4. Design and implement an iOS application to setup and control an inspection/monitoring task. A user submits a sequence of points of snapshots and camera orientations in these points and the application returns a collision-free path in a known map. The user can edit the obtained path and confirm its execution.
5. To verify the inspection/monitoring application in Gazebo in scenarios of warehouse monitoring and inspection of historical buildings [3,4]. To verify the application with a real platform in outdoor conditions.

Bibliography / sources:

[1] T. Baca, P. Stepan and M. Saska. Autonomous Landing On A Moving Car With Unmanned Aerial Vehicle. In The European Conference on Mobile Robotics (ECMR), 2017.
[2] G. Loianno, V. Spurny, J. Thomas, T. Baca, D. Thakur, D. Hert, R. Penicka, T. Krajnik, A. Zhou, A. Cho, M. Saska, and V. Kumar. Localization, Grasping, and Transportation of Magnetic Objects by a team of MAVs in Challenging Desert like Environments. IEEE ICRA and RAL, 2018.
[3] S. Winkvist, E. Rushforth, K. Young. Towards an autonomous indoor aerial inspection vehicle. Industrial Robot, 40(3):134-156. 2013.
[4] M. Saska, V. Kratky, V. Spurny, and T. Baca, "Documentation of dark areas of large historical buildings by a formation of unmanned aerial vehicles using model predictive control," in IEEE ETFA, 2017.

Name and workplace of bachelor's thesis supervisor:

Ing. Martin Saska, Dr. rer. nat., Multi-robot Systems, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **31.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Martin Saska, Dr. rer. nat.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Date	Signature

Acknowledgments

I would like to thank my adviser and supervisor Dr. Martin Saska for his advice. And also thank all members from team multi-robotic systems, who helped me to finish this thesis.

Abstract

This work aims to control unmanned helicopter with a mobile device with operating system iOS. The goal is to implement a solution, that can command helicopter in three ways. The first is Joystick Control, the second set trajectory in Google Maps and last one is indoor navigation, for which a precise map is needed. The work is based on results from Department of Cybernetics at Czech Technical University in Prague group Multi-Robotic systems.

Keywords

UAV, iOS, MacOS, ROS, mobile application, indoor navigation,

Abstrakt

Práce je zaměřená na ovládání bezpilotni helikoptery za pomoci mobilního zařízení s operačním systémem iOS. Cílem je implementovat řešení, které bude schopno ovládat helikoptéru ve více módech. První je ovládání joystickem, druhá nastavení trajektorie pomocí Google Map, poslední ja navigace v interiéru budov, pro kterou je potřeba získat přesnou mapu. Práce vychází z výzkumu pracovníku na Katedře kybernetiky skupiny multi-robotických systémů.

Klíčová slova

UAV, iOS, MacOS, ROS, mobilní aplikace, indoor navigace,

Contents

List of Figures

1 Introduction

The twenty-first century is the century of big technological progress. Smartphones are the most available and the most powerful ever. Almost everybody on the Earth has a mobile phone. On the other side of progress are autonomous vehicles, which have the biggest progress in the last ten years. This progress is in water, ground and aerial vehicles. It led to the decision to implement an application for smartphone, that would be able to control unmanned aerial vehicle. This project has been separated into few sub-projects, where every work has different requirements. The first major requirement is an operating system. The most works in this way are focused on Android devices. This thesis is for iOS devices. Next requirements are very variable because every project can be focused on a different trend. Trends can be agriculture, where UAV will map field and decide where should be engraved. Next trend is processing in storage halls, which is operated by humans now, but autonomous vehicles can improve processes in this way.

2 Problem definition

The first requirement is UAV, for this purpose has been used the final product developed by team multi-robot systems[4] at FEE CTU. All helicopters contain a computer, running on the operating system Ubuntu. The main feature on that computer is ROS. ROS is middle-ware, which control everything on the board, deeper description of the helicopter is in chapter 3. With this board can be communicated in many ways. In this work will be used for communication between UAV and a mobile device a Wi-Fi. Communication is wider described in chapter 6.2. A ROS for real helicopter computer is modified to work with Gazebo. The gazebo is a simulator of the real world. The gazebo s described in chapter 4.3. Every needed experiment can be before real flying tested in Gazebo simulator. That is very recommended because an error on a real helicopter can be fatal.

Next requirement is a mobile application for the iOS operating system. Implementation of mobile applications for iOS can be done in many programming languages, but the most recommended is Swift. Next need for developing an iOS application is to develop on Apple device, which is the first problem of this work because development has to use Ubuntu for running ROS and MacOS for running IDE. That conflict leads to virtualization of the second operating system, or have dual boot on one device, attempts to this simplification is described in chapter 5.

This work inhibits exploration of an indoor map, that is part of another work.

3 Unmanned helicopter

Figure 1: Micro Aerial Vehicle

As a device for controlling will be used an unmanned aerial vehicle, invented on Department of Cybernetics by the group of multi-robotic systems. A requirement of this work is only one UAV. Set of this vehicle are used for students research and development. For example Control and Navigation in Manoeuvres of Formations of Unmanned Mobile Vehicles[5].

The main feature of the helicopter is a self-stabilizing system. It can hold its latitude, longitude, and height according to built-in very precise GPS locator. Due to this system drone has better commanding options, for example, to move one meter left, or to rotate by 50 degrees. Helicopters without self-stabilizing system have directly controlled motors. That type of drones is often used for FPV racing.

The most used types of UAV are quad-copters with four propellers, but the used helicopter is hexacopter, which means that it has six propellers. The reason, why to have six propellers, is because each helicopter contains expensive parts and if some motor or propeller fails, the inner system is able to safely land with five propellers, which is not possible with quad-copter.

3.1 UAV

Definition of UAV tells that Unmanned aerial vehicle is a vehicle without any passenger or driver on board. It can be controlled autonomously or remotely. It can carry lethal package, which can be very dangerous.

3.2 Multi-robotic experiments

Even if this project is aimed only at one vehicle, multi-robotic experiments are undivided and main experiments of multi-robotic system group. As was mentioned in the introduction, every experiment has lots of tests in a simulator before real visualization, but on real hardware can every program fail. For this reason, has been developed Collision avoidance system[6]. This system can avoid failed planned trajectory, in case of a possible collision. System using multi-master system, where every vehicle knows about each other. Thanks to the multi-master system can Collison avoidance system predict the future position of every aerial vehicle at the same time and set distance can cancel trajectory and plan trajectory so they will miss.

3.3 Requirements for UAV in this project

In this project, UAV will use, necessary systems prepared in default settings. The drone has the all sensors for self-stabilizing. Next connected component is GPS device, which cares about every localization, that device is also present in every helicopter. For localization is not used in standard GPS system, but UTM system, UTM coordinates is better described in chapter 6.2. The last necessary component is Wi-fi because drone must be connected to the same Wi-fi as a mobile device because every communication is managed over Wi-fi. Before using the real helicopter, implementation has to be tested in a simulation which is provided in Gazebo, graphic simulator connected with ROS, described in next chapter.

4 ROS[7]

ROS means robot operating system. It is middle-ware used by many developers for testing and implementing algorithms for run robots, starting with small simpler robot projects going over our drones, ending with smart robots with almost infinity features. All ROS is designed to be as distributed and modular as possible, which is also a reason why there are over 3000 available packages in ROS ecosystem. Around ROS is a big community of developers. Its community has about 1500 participants, and there are thousands of discussed themes on their Wiki.

4.1 Uav_core

Uav_core is package implemented by students of FEE CTU in Prague from Department of Cybernetics. It package implements control methods of UAVs and UGVs (Unmanned aerial vehicles and Unmanned ground vehicles). This thesis aims to UAV. Therefore UGVs will not be described. Around uav_core has been implemented lots of packages controlling simulated devices, which are using on real UAVs. Devices are, for example, cameras, RFID locators used in this bachelor thesis[8] and many similar components. For control of helicopter in this project is used Uav_core package.

4.1.1 Uav_core commands

Commands for drone can run via terminal on PC. There are few basic commands implemented[9]. Their functions are mainly moving to absolute or relative positions complexly or divided by axes. We can also set the trajectory for a drone. A setting of relative positioning is simple. The unit of each parameter is one meter. For setting absolute position is work little harder. Because every command use UTM coordinates and they are usually offset, for adjusting zero position to the center of the testing area. This will be the last step in communication with UAV. This leads to finding a communication tool for an iOS device, described in chapter 6.2.

4.2 UTM coordinates

Universal Transverse Mercator coordinate system (UTM) is in contrast to Global Positioning System (GPS) separated into zones from -60 to 60, zero is excluded. Zones with a negative number are in the southern hemisphere, and the zones with a positive number are in northern hemisphere[10]. With UTM was at the first time defined one meter, because the distance from the equator to North pole was 10 000 km. That leads to

a result, where Easting and Northing coordinates are always in meters.

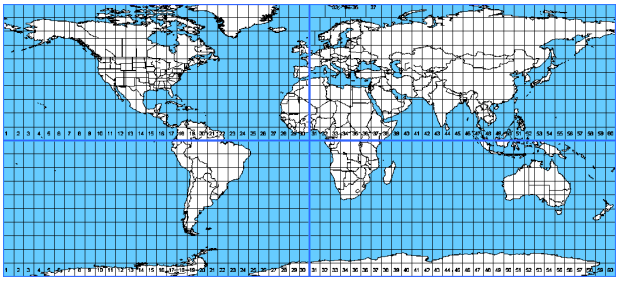


Figure 2: UTM Zones [1]

4.3 Gazebo

As was many times mentioned application for UAV has to be tested on a simulator, and after successful tests, it can be built and run on real UAV. ROS is the middle-ware that proceeds every controlling of drone, but for simulation is there Gazebo, robot simulation tool. The gazebo can simulate dynamic systems, robots, sensors like a camera and cloud simulation also. For our purpose has been prepared models of UAVs, with a camera and typically same as real UAV contains.

Figure 3: Screenshot from gazebo [2]

5 ROS on MacOS

Best way to develop and implement iOS application is to use Xcode, which is recommended IDE for iOS development. Xcode is only able to run on MacOS and maybe on some illegal copies, for example Hackintosh. When we want to go via the legal way, we can only use the MacOS. So here comes the first complication. There is a ROS running on Ubuntu and Xcode running on MacOS. To make development more comfortable and straight by using one computer which would run both operating systems. There are few ways how to do that.

5.1 Dual boot

The first option which can remove the problem of two machines is to use dual boot. With dual boot can be reserved half of drive to Ubuntu and half for MacOS. But of course, there is a problem. Apple devices have a guarantee, and by installing other OS on it, the guarantee would have expired. Otherwise installing Ubuntu on Apple device is not so easy, because these devices are very protected. Even if all problems are resolved, time to restart the computer after updating the mobile application to start Ubuntu and ROS will take much time. And in long-term work, it will be very uncomfortable.

5.2 Boot camp

Lots of Mac users use Bootcamp to run Windows, parallel with MacOS. That is the cleanest way to start different operating system on Apple device and Windows are very stable, because every Apple configurations are known, and there are not any differences with a different setup. But the problem is, Boot camp is no compatible with Ubuntu or other Linux derives.

5.3 Virtual machines – Parallels, VMWare, Docker, VirtualBox

5.3.1 Parallels Desktop[11]

Parallels Inc. is presented as the global leader of cross-platform solutions, and their product Parallels Desktop is a program for Mac, that can virtualize Ubuntu and other operating systems. With this type of virtualization, I was not able to run ROS.

5.3.2 Docker[12]

Docker does not work as similar to the other ones. Docker comes with the method, that divides all apps into containers. Docker is very often used as a development product for testing new apps on "clear systems." It guarantees high security, portability, and agility. But also, here I was not able to run ROS, major problem was with graphic interface, in this case Gazebo.

5.3.3 VirtualBox[13]

VirtualBox is one of most used virtualization product, developed by Oracle. I tested many Linux distributions on VirtualBox. I started with running classic Ubuntu, which was recommended for uav_core. There was ROS successful, but we were on about 5 FPS in Gazebo simulation, but it is not so stable and easy to develop. After all these tests I found out, VirtualBox with Ubuntu is not able to setup 3D acceleration, which can improve the power of a virtual machine, increase frames in Gazebo simulation and of course, improve the fluency of simulation. Here I moved to testing other derives from Linux, for example, Lubuntu, Xubuntu. These types have the best rating for this type of applications. On both operating systems I observed better fluency, higher FPS, but the problem comes when I tried to repeat installation it always failed.

5.3.4 Conclusion of virtual machines

The idea of virtual machines seemed to be very positive at the beginning of this research, but the result is that virtual box is not suitable for running such types of operations, which ROS offer.

5.4 Conclusion of ROS on MacOS

The Result of this research should be for example package with the installation script, that will compile and install all necessary components of ROS with uav_core on MacOS. But after all tests and tries, we decide to stop with this way and move back to two devices and implement mobile app on MacOS and simulate drone on PC with Ubuntu. Result is not very good, but hardware requirement for both development will split into two computers.

6 Mobile app

6.1 Operating system

At the department of cybernetics were this year developed four mobile applications, three of them were developed for operating system Android. For this project has been chosen iOS as an operating system.

6.1.1 Android applications

Applications for system Android could be developed in few programming languages, mostly used is Java. These days lots of developers have started using Kotlin, which is now recommended programming language for writing Android applications because it helps programmers to increase stability and efficiency of their application. All my colleagues, which has developed Android mobile application used to development Java, because they had much experience with Java and it would be very hard to learn a new language and simultaneously implement advanced features for mobile applications. I would like to refer to the website of Multi-robot systems group[4], where can be found all these projects, in bachelor theses folder.

6.1.2 IOS development

Developing of iOS applications started with releasing the first iPhone in the year 2007. At the beginning of iOS development was mostly used in programming language Objective-C usually shortened as ObjC. ObjC is programming language base on C, which is one of most known programming languages worldwide. For development is highly recommended Xcode as IDE developed by Apple Inc. Nowadays is mostly used language for development Swift, which is little similar with Kotlin, mentioned in Android section, but Swift is older. The latest version of Swift is Swift 4.0, which as first version is backward compatible. Backward compatibility was one of the major issues of Swift because for example Android application developed in Kotlin can also contain Java classes without any difficulties.

6.1.3 Operating system comparison

There is no relevant comparison for these operating systems. There will be described major pros and cons.

1. Android pros
    - Open source operating system
    - Customizable system
    - More affordable devices
2. iOS pros
    - Stable system
    - Apple support
3. Android cons
    - Less stable system
    - Easier to hack
4. iOS cons
    - Closed system
    - Expensive devices

Comparison of developing applications, will not be described, because in these days quality of IDEs, languages, and community are at the very high level, and in my opinion in both both cases are very similar.

Important to be mentioned is description of distributing of applications. Both operating systems have applications stores, places where to download applications. For Android it is the Google Play and for iOS is there AppStore. Android developers have to pay 30 USD to be able to distribute applications on Google Play. After that payment developer has an unlimited license without expirations for distributing Android applications. Next option for distribution to share a file with *apk* format this file is automatically generated from Android Studio. Every Android user can easily install that file.

With iOS development, it is harder. For distributing on AppStore is necessary to license, this license costs 100 USD and has one-year validity. The reason, why this license is so expensive, is firstly Apple policy, secondly Apple review team. After every application upload to AppStore, developers must wait for a review. The review is done by one of Apple review teams. These guys check the completeness of the application, unauthorized system flows and forbidden themes. A review should decrease the count of applications of poor quality with security problems.

6.2 Communication

As mentioned in ROS section of this thesis, there is set of useful scripts to control the drone. With this option comes possibility to connect the app with drone or simulator via SSH. For this communication, was used NMSSH library[14]. Library NMSSH can be used in three ways. First is manual download from GitHub and insert classes into the project, the second one is via Carthage. Carthage[15] is dependency manager, which helps developers to use libraries developed by other developers to increase the quality of their applications. Last way is to use dependency manager called CocoaPods. CocoaPods is the most used dependency manager, and it was used for this application. With NMSSH library can application start connection via SSH, log in as an arbitrary user, send bash commands and receive responses for this commands.

6.3 Architecture

In the begging of developing mobile applications was not architecture or structure of code resolved. In bigger has begun to be a problem with readability and reusability. For developing a readable, reusable app, is needed to hold self-descriptive and clean code. For this purpose, app architectures were used. After my experiences, I use B-VIPER, which is little difficult, but very clean architecture. This architecture represents all screens as separated blocks. Over these block is a component called router. This component routes all blocks through the app. Every block contains three main components/classes. First one is a view, this class holds instances of every view, which should be visible on that screen. And handle all user interactions, that is presented into the second part presenter. Its main purpose is to connect view with interactor and router. Interactor should do all logic processes such as communication with API, parsing models from network responses, communicate with databases. The penultimate class is the router. Router handle only starts and finishes of the block. Name of architecture has meaning, Builder, View, Interactor, Presenter, Entity, Router. Description of the builder is to construct that block and entity only contain variables.

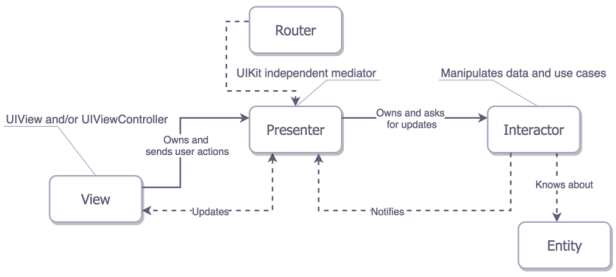


Figure 4: VIPER architecture [3]

6.4 Design

On all screens are used native basic native graphics components. I user two specific views, first one is joystick from CDJoystick library[16]. This library was used only to joystick commanding. Second one is Google Maps library for iOS[17].

6.4.1 Navigation controller

Every screen in the mobile application whether iOS or Android and for controlling transitions between screens, hiding and showing screens is necessary some controller, in iOS is navigation controller. Working with the controller can be in two ways. The first one is to use storyboards, where every they are completely taking care of navigation controller. The second one is to take care of navigation controller as a programmer.

6.4.2 Storyboard

Storyboard is a type of a file, which can be used for implementing mobile app user interface in one section. A benefit is that every change and new screens can be updated from one place. A disadvantage can be taken orientation in one big storyboard in some big application. An important feature of storyboards as described in this paper [18], is that it contains an opportunity to add a transition between screens. Thanks to this transition can be implemented routing. Storyboard is a file which can be edited in Xcode with tool Interface Builder. Anyways developers nowadays use programmatically implemented layouts, and also switch screen by accessing the navigation controller.

6.4.3 Programatically implemented layouts and transitions

As was mentioned in the description of Storyboards, for development can be used graphics implemented the same language as rest of application. Swift offers same features as storyboards do. Setting view is not always absolutely, but almost every view is positioned relatively. A benefit of this settings is that created layouts fit every resolution and sizes of iOS devices. Every screen can be designed separately, but it is unnecessary except layouting screens for tablets and mobile phones, this separation makes sense.

For there are functions from *UINavigationController* class, which are for example *pushViewController(_ viewController: UIViewController, animated: Bool)* this function present new UIViewController, optionaly animated. UIViewController is class representing generally one screen. There are also function for hiding view controlller *popViewController(_ viewController: UIViewController, animated: Bool)*. Next varible of navigation controller is root view controller, which is view controller showed in first, when starting application.

For setting root controller is there function *set(rootViewController: UIViewController)*. Important function is *popToRootViewController(animated: Bool)*. With this function can developers easily start application flow from beggining.

6.5 External libraries

For straighter development has been developed open sourced external libraries. In this application has been used dependency manager called CocoaPods[19]. This is mostly used dependency manager. From used libraries, I would like to mention Stevia Layout[20]. Stevia is a library that helps developers to more comfortable implement graphics as part of an application directly in Swift instead of using storyboards.

7 Main page

First page5 contains three fields to insert IP address of controlled drone, the name of user account and password, next possible field should be UAV number, which would specify drone for actual session.

The mobile device has to be connected to same Wi-Fi as the drone for successful communication. After inserting correct data into fields, the application asks to choose a type of control. If a user inserts wrong data or device is not connected to the same network as UAV, the application makes pop-up dialog with the error message.

There are four types. The first type is Joystick commanding, and the second one is Google Maps trajectory, the third is Indoor map trajectory and the last one Terminal communication. Every communication except terminal is described in next sections. A terminal was implemented only for development purpose. It works as a classic terminal on a desktop computer. One input cares of commands and output are responses from commands.

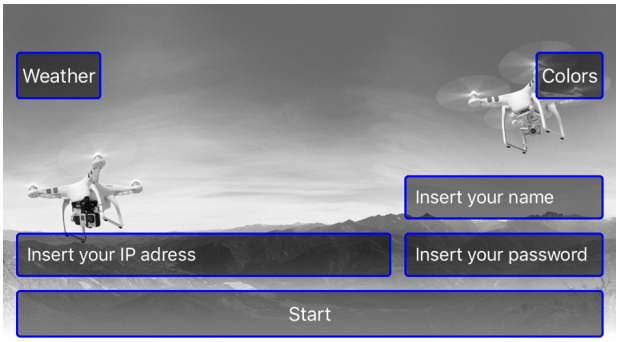


Figure 5: Login page to the application

8 Joystick commanding

The first type of control is joystick commanding, which is basis control.
On this page is two joysticks. Left one controls rotation and height of drone. Right one is for moving forward, backward, left or right.

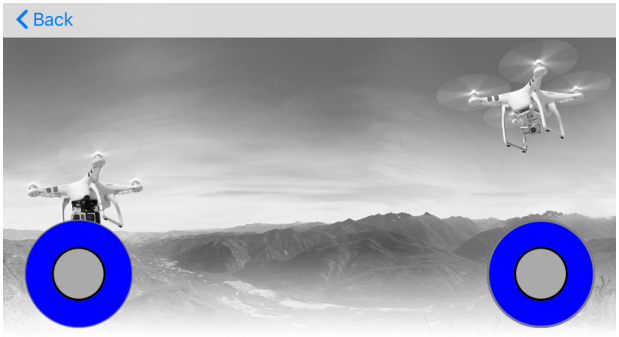


Figure 6: Joystick controlling screen

In the begging of development, I was facing few issues about setting SSH communication, because NMSSH library uses non-interactive access. After a short research, I found the solution. After that, I move to testing. Testing on simulator was successful, but on the real drone, SSH showed that it is not useful.
The problem was in speed of SSH connection, the standard command takes about one second and one second was also paused interval in repeating loop of sending commands. That leads to very noncontinuous movement. The result of these tests is, that SSH commanding is too slow for real-time drone control.
The better way to control drone in real time is via sockets. In this way can be frequency increased from 1 Hz to about 100 Hz.

9 Google Maps trajectory

Next type of drone control, which was implemented is Google Maps trajectory. This way can be the groundwork for more sophisticated planning algorithms.
On the screen7 is Google map after click on some place marker will be placed there. This marker represents on of point where drone has to fly. These points can be easily added and moved, after prepared points, the user clicks into the left side of the screen where is a square button of drone, which approximate trajectory to the smaller distance between each point and send this trajectory to the drone. In right top corner is button **goto** which represent a command, which will send the drone to his start position. If a drone is one start position, button **start** can be tapped, and the drone will start following set trajectory.

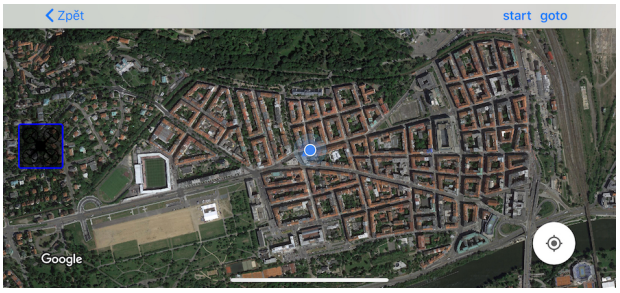


Figure 7: Map controlling screen

With this type, I did few experiments, at the UAV camp, which is one week of testing all experiments for bachelor, magistrates and doctoral students. These experiments will be described in next section.

9.1 Installation Google Maps library

Google Maps is free to use for all developers around the world. To implement maps into application developer had to have Google account. With that account, the developer has to register the application with its Bundle ID to Google Maps database. According to this registration, Google provides guide and tutorial for integrating Google Maps into iOS application. Google also provides App ID for identifying.

9.1.1 Bundle ID

Bundle ID is the unique identifier for every application and its environments. This ID contains region identification, the name of company and name of the application all separated by dots, e.g., *cz.fel.cvut.droneapp*. As I told this ID can also consist of an environment. Application environment is a specification of an application, which includes

different application icons, different base URL to communicate with a server and many others. Applications are generally implemented in three environments. Development, staging and release version. Development is for developers and testers only, staging version is for client or crowd testers, and release version goes only into AppStore in case of the Android operating system to Google Play.

10 Indoor navigation

10.1 Overview

Last part of this project is an implementation of indoor navigation because drone will be controlled as same as in the Google Maps. For implementing this feature can be used some codes from the last part.

10.2 Preparation

For indoor navigation app needs some map. Scanning of an area to navigation is a goal of another project, so it will not be described here. An output of that project will be data in Point Cloud Data (PCD) format. This format is nothing more than a long list of points in three dimensions. PCD is very often used in many projects, e.g., area mapping, object modeling.

10.3 Ways of representation

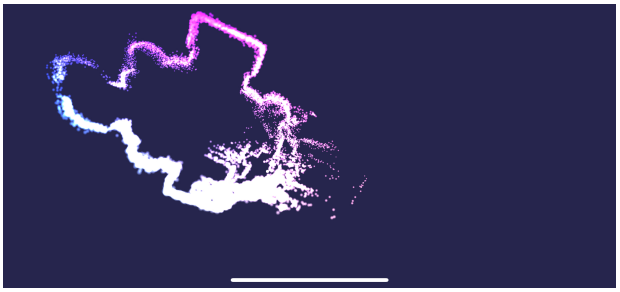


Figure 8: Screen with displayed Metal Kit

10.3.1 MetalKit[21] 8

The first way to display point cloud data is to use MetalKit as a native iOS component. This component is available from iOS version 8.0, which is not complication because the minimal version of the app is 10.0, that corresponds to iOS development standards. These standards recommend at least supporting last two version of operating system. In the year 2018 is the latest version 11.3. MetalKit can display lots of points in three dimensions. The problem is that many points mean hundreds, but an average PCD file contains about 50 000 points, and there should be even more. This problem can be solved with approximation, that will be described in next section. The last and the biggest problem of MetalKit is user interaction. Three dimension is very comfortable to observe and watch models and data, but very uncomfortable to insert some points which will be necessary to plan some trajectory.

10.3.2 UIBezierPath[22]

UIBezierPath is also a native component of Apple company. This feature is initialized by the list of points, that should be shown. PCD is from its name based on points, but Bezier does not place point but draws lines. This problem can be easily resolved in with some type of sorting by the nearest point. This procedure will be described in next section as well. But again this way is not very straight for PCD representation.

10.3.3 View in view

As a last try of representation of PCD is to use classic UIView[23]. UIView is basic UI component to develop an iOS application using Swift language and Xcode. It is available from lowest version of operating system and work is pretty simple. This way will be only in two dimensions, that won't be as nice as MetalKit, but user experience to add points for trajectory would be better. The first process to start implementing this issue is to sort all point by z-axes, sorting will be described in next section as well. After sorting all data will be separated into ten layers in same heights. Every layer will be chosen by the slider.

10.4 Processing

10.4.1 Approximation

Approximation for MetalKit has two steps first step is to sort all point by random axis. For sorting is used QuickSort 1 algorithm and axis, which I had to choose is x. The second step is to decrease density!2 of points by the removing points, which are to close to others.

Algorithm 1 QuickSort - Pseudocode

```
function quickSort(arrayToSort, lowestObject, highestObject)
    if lo <hi then
        p = partition(arrayToSort, lowestObject, highestObject)
        quickSort(arrayToSort, lowestObject, p)
        quickSort(arrayToSort, p + 1, highestObject)
    end if
end function

function partition(arrayToSort, lowestObject, highestObject)
    pivot = arrayToSort[lowestObject];
    i = lowestObject - 1;
    j = highestObject + 1;
    while true do
        repeat
            i = i + 1
        until arrayToSort[i].x <pivot.x
        repeat
            j = j - 1
        until arrayToSort[j].x >pivot.x
        if i.x ≥j.x then return j
        end if
        swap(A[i], A[j])
    end while
end function
```

Algorithm 2 Decreasing density of points - Pseudocode

```
function decreaseDensity(arrayToDecrease, threshold)
    comparedPoints = 0
    checkedPoints = 0
    while  docomparedPoints <arrayToDecrease.count
        actualPoint = arrayToDecrease[comparedPoints]
        while (comparedPoints + checkedPoints) <arrayToDecrease.count do
            checkingPoint = arrayToDecrease[comparedPoints + checkedPoints]
            if abs(checkingPoint.x − actualPoint.x) >treshold then
                break
            else if actualPoint.distanceTo(point : checkingPoint) <treshold then
                arrayToDecrease.remove(at: comparedPoints + checkedPoints)
            else
                checkedPoints = checkedPoints + 1
            end if
        end while
        checkedPoints = 0
        comparedPoints = comparedPoints + 1
    end while
    return arrayToDecrease
end function

function distanceTo(firstPoint, secondPoint)
    xDiff = firstPoint.x - secondPoint.x
    yDiff = firstPoint.y - secondPoint.y
    zDiff = firstPoint.z - secondPoint.z
    xDiff = xDiff * xDiff
    yDiff = yDiff * yDiff
    yDiff = zDiff * zDiff
    return sqrtxDiff + yDiff + zDiff
end function
```

10.4.2 Sorting by the nearest point

This sorting was not implemented, but the process would be much similar to the preparation for MetalKit. The first step would be to sort points in the z-axis, for this can be used QuickSort mentioned above. The second step is to separate points into layers. After separation points have to be sorted according to the lowest distance. In begging there is array sorted by z-axis and an empty array. Take the first point from sorted array insert it to an empty array and find next nearest point, z-axis can take as a threshold of minimal distance. Repeat the last step until the sorted array is not empty. New array can be used as an initializer of the UIBezierPath. This leads to connecting all possible point by lines.

10.4.3 Sorting for views in view

Last processing method belongs to views in view representation. Point have to be sorted by z-axis and separated into layers.

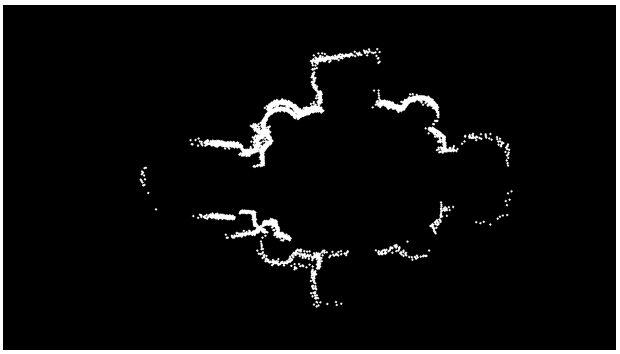


Figure 9: Points as view in view

Points represented as views in view is the final version of this part of the application. This solution has not the best design, but the design is not the main purpose. The main purpose should be user interaction. Interaction is described in next section.

10.5 User interaction and tutorial

Screen with indoor navigations has two active blocks, first one is the slider, which controls visible layer of PCD as an indoor map. The second block is just a map, which contains points in the selected layer. The map has gesture recognizers for zooming out and zooming on the map, rotating the map and move on the screen. This two should ensure good set the trajectory. For set trajectory user have to long press screen in the desired place as same as in Google Maps. Same is also replacing points, hold finger again on the marker

to replace marker. After prepared trajectory there are three buttons to set trajectory, go to start trajectory and follow the trajectory.

11 Trajectory preparations

As was mentioned in google maps and indoor navigation part of this thesis, every selected trajectory have to be approximated, smoothed and prepared for each environment. This process is described in next sections.

11.1 General preparation

After user interaction with an application, there is an array of coordinates, which have three dimensions. For the ROS have to be trajectory approximated. Each point of the trajectory has to be in maximum 8 cm distance. Reason for this requirement is that optimal UAV speed is 4 m/s and tracker of UAV is implemented to fly between every two points in 0.02 seconds.

11.2 Preparation for simulator

In simulator were tested both implementations of navigation. Google maps are very useless in a simulator because the coordinations accords to real GPS positions and it can not be practically tested in a simulator. Testing of indoor navigation in a simulator is very straight and easy because points selected in mobile application accord to coordinations shown in Gazebo.

11.3 Preparation for real flying

In real flying, there is more preparation for each part. The first step for Google Maps is to convert GPS longitude and latitude as are standard coordination used in Google Maps to UTM coordinations. For every testing session with real drones, are set some offset for coordinates. All these offsets are unique for every session, and they have to be set every time in an application. After adding these offset to all coordinates is trajectory approximated and send to UAV to process. Easier processes come for indoor navigation. Map, which was captured by UAV is in prepared with designated offset, and it is ready to use for send. Only changes are connected with representation for a mobile device.

12 Experiments

While this work was executed with few experiments, they were not always successful, unfortunately, but they were always rewarding. Results led to future improvements and completion.

12.1 Joystick commanding

The first part of the application was implemented mainly for testing purpose, to check if communication works, if UAV reacts, these features were successfully tested and gained experiences could be used in next parts. But the result of the joystick was not the best at last. Experiment in the simulator was so not so straight as the one on the real UAV. Communication via SSH was too slow, and movement of UAV was discontinuous. This suggests that SSH connection cannot be used for final product at least for a joystick.

12.2 Google Maps

Useful experiments were executed only on real UAV. There were two experiments. Both were about setting trajectory on a mobile phone and sending it into UAV. Video and screenshots of application are saved on DVD. In picture 10 and 11 are compared inserted points and trajectory from rosbag, which is storage collecting data from flight. Graph is processed in MATLAB. Results of experiments were in this case more successful than in the previous part, but problems that come from testing were, more about speeding up and improving development. Problems were with repeating setting of the number of UAV, offset of coordinates and for example implement take off function or battery status for UAV.

12.3 Indoor navigation

Unfortunately, tests of indoor navigation were not executed on real UAV, neither on simulator, problem was in work with Gazebo and ROS systems, where I was unable to render PCD. PCD is format of points in 3D.

12.4 Problems and possible solution

After all experiments, few issues can be implemented in next thesis or as next individual projects.

Known issues and Possible solutions

- Improve development of iOS - ROS applications by launching ROS on MacOS
- Replace SSH communication with socket direct socket communication with ROS
- Implement node for handling map offset
- Implement feature for a mobile application that can set number of UAVs as user
- Implement node that can send captured data of the map to a mobile device

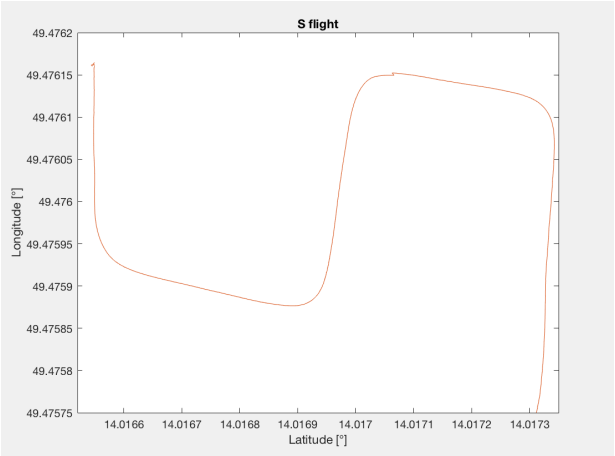


Figure 10: Longitude and latitude graph from experiment

Figure 11: Screenshot of inserted trajectory

13 Conclusion

At the beginning of this work was a goal to have ROS middle-ware running on MacOS, interface for the iOS device, which can communicate with ROS and command the UAV. That can be used for next development projects, that can, for example, extend the whole application, or focus on one part which, would be used for the final product. At the start of this thesis is described UAV, which was used for every real experiment. A helicopter has not any additional components against the traditional UAV in group multi-robotic system. Next was described used interfaces between application and UAV, that interface is ROS, very open and distributed system, that takes care of every communication on board. Experiments of running ROS on MacOS were tried on much virtualization products and with many parallels system, but no one was so good for next development. Development of application for iOS and UAV has been separated into two devices. In part called mobile application was in detail described differences between developing for Android and iOS devices in focus on the iOS device, where was development described more detailed because this work is aimed at iOS applications. In this section were also illustrated the algorithm for processing map. This algorithm is not very sophisticated, because the emphasis was not put on algorithms. Next part is clearly described experiments. These experiments were executed in a simulator and on real helicopter too. All attempts were in the end successful, but communication was not so optimal, because SSH is not enough fast for controlling UAV.

13.1 Next steps

Several tasks can be done in future. First one is to focus on virtualization of Ubuntu and set the best possible configuration to run Gazebo with ROS smoothly. Next possible way is to run ROS on MacOS, there is a possibility to download ROS working with MacOS, but it is only in experimental mode, and start with a blank project and little by little put in operation every implemented ROS package. Last most important and remarkable task is to communicate directly with ROS not by SSH using one of available CocoaPods libraries, there are many of them available on www.github.com.

References

[1] "Utm zones image," accessed: 2018-05-19. [Online]. Available: http://www.resurgentsoftware.com/images/UTM_WORLD.gif

[2] P. Petracek, "Screenshot from gazebo," 2017, accessed: 2018-05-24. [Online]. Available: http://mrs.felk.cvut.cz/data/students/petracekBP.pdf

[3] "Viper block diagram," accessed: 2018-04-15. [Online]. Available: https://cdn-images-1.medium.com/max/1600/1*0pN3BNTXfwKbf08lhwutag.png

[4] "Website of multi-robot systems group," accessed: 2018-04-15. [Online]. Available: http://mrs.felk.cvut.cz/people/martin-saska

[5] M. Saska, J. Mejia, D. Stipanovic, V. Vonasek, K. Schilling, and L. Preucil, "Control and Navigation in Manoeuvres of Formations of Unmanned Mobile Vehicles," *European Journal of Control*, vol. 19, no. 2, pp. 157–171, March 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0947358013000204

[6] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," 2018, (submitted to IEEE Robotics and Automation Letters).

[7] "Website of multi-robot systems group," accessed: 2018-03-20. [Online]. Available: http://www.ros.org/is-ros-for-me/

[8] M. Vrba, "Active searching of rfid chips by a group of relatively stabilized helicopters," 2016. [Online]. Available: http://mrs.felk.cvut.cz/data/students/vrbaBP.pdf

[9] "Commanding drone," accessed: 2018-04-15. [Online]. Available: https://mrs.felk.cvut.cz/gitlab/uav/uav_core/wikis/commanding_the_drone

[10] "Utm coordinates," accessed: 2018-05-19. [Online]. Available: http://www.resurgentsoftware.com/GeoMag/utm_coordinates.htm

[11] "About parallels," accessed: 2018-03-20. [Online]. Available: https://www.parallels.com/eu/about/

[12] "What is docker," accessed: 2018-03-20. [Online]. Available: https://www.docker.com/what-docker

[13] "Welcome to virtualbox.org," accessed: 2018-03-20. [Online]. Available: https://www.virtualbox.org/

[14] "Nmssh github repo," accessed: 2018-04-15. [Online]. Available: https://github.com/NMSSH/NMSSH

[15] "Carthage," accessed: 2018-04-15. [Online]. Available: https://github.com/Carthage/Carthage

[16] "Cdjoystick github repo." [Online]. Available: https://github.com/Coledunsby/CDJoystick,note=

[17] "Google maps library for ios," accessed: 2018-04-15. [Online]. Available: https://developers.google.com/maps/documentation/ios-sdk/intro

[18] J. Stefancik, "Mobile application development for ios in objective-c," 2015. [Online]. Available: https://theses.cz/id/zhfxzf/stefancik_bp.pdf

[19] "Cocoapods," accessed: 2018-04-15. [Online]. Available: https://cocoapods.org/about

[20] "Stevia layout," accessed: 2018-04-15. [Online]. Available: https://github.com/freshOS/Stevia

[21] Metalkit. Accessed: 2018-04-15. [Online]. Available: https://developer.apple.com/documentation/metalkit

[22] "UIBezierPath - UIKit | apple developer documentation," accessed: 2018-04-15. [Online]. Available: https://developer.apple.com/documentation/uikit/uibezierpath

[23] "Uiview," accessed: 2018-04-15. [Online]. Available: https://developer.apple.com/documentation/uikit/uiview

Appendix A DVD Content

In Table 1 are listed names of all root directories on DVD.

Directory name	**Description**
thesis	the thesis in pdf format
thesis_sources	latex source codes
app	source codes of application
media	videos and screen shots from experiments

Table 1: DVD Content