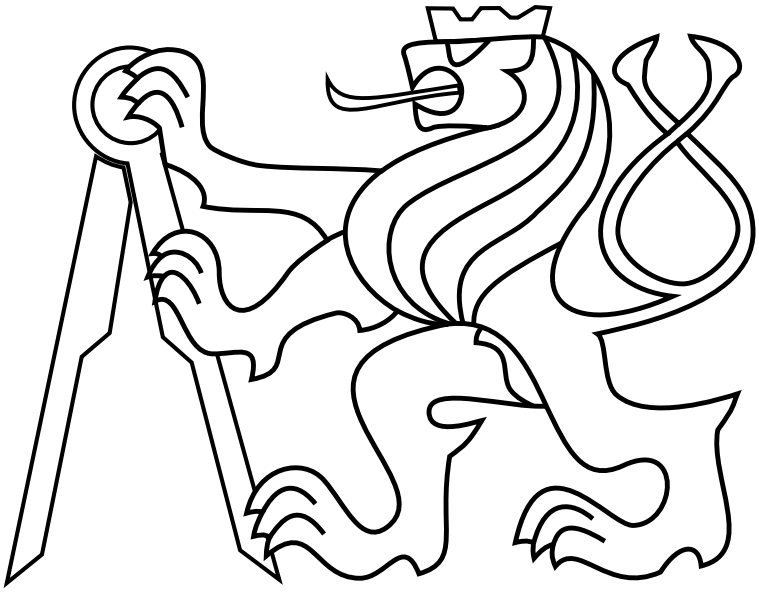
CZECH TECHNICAL UNIVERSITY IN PRAGUE

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: **24.05.2019**

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 advisor 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 a helicopter in three ways. The first is a Joystick Control, the second sets trajectory in Google Maps, and the last one is indoor navigation, for which a precise map is needed. The work is based on the results of Multi-Robotic Systems group from the Department of Cybernetics at Czech Technical University in Prague.

Keywords

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

Abstrakt

Práce je zaměřená na ovládání bezpilotni helikoptéry 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 significant technological progress. Smartphones are the most available and the most powerful ever. Almost everybody on Earth has a mobile phone. On the other side of development are autonomous vehicles, which have the most significant progress in the last ten years. This progress is in water, ground, and also aerial vehicles. It led to the decision to implement an application for smartphone, that would be able to control UAV (unmanned aerial vehicle). This thesis is a part of a bigger project. The main focus of the project is to develop applications that increase the automation in many specializations. Other applications are created as theses of other students. Nowadays, the most popular operating systems for mobile applications are Android and iOS. The majority of the thesis in this project are created for Android devices. This thesis is for iOS devices. Additional specifications are very variable because every project can have a different focus. The focuses are, for instance, agriculture, where UAV will map the field and decide where it should be engraved, and processing in storage halls, which is operated by humans now. In each sphere can the autonomous vehicles improve the processes and reduce the price of tasks.

The UAV is a remotely controlled, autonomous flying vehicle. It has from 4 to 8 propellers, in some applications the count could be higher. Propellers are mainly powered by electric motors. The power supply is a battery or a set of batteries mounted to the vehicle. A vehicle with four propellers is called quadcopter and with six propellers is called hexacopter. This thesis has been tested on hexacopters, developed by the students from Czech Technical University in Prague.

Hexacopters that are used for testing contains a motherboard with installed Ubuntu system and running Robot operating system (ROS). ROS is an open-source middleware, which is used for research and development of various types of robot applications. Before deploying to real drones, it will be tested on Gazebo (Robot simulator). There are models of hexacopters in Gazebo. In order to test the application on a computer, is needs to have Ubuntu operating system, ROS, and Gazebo that can simulate the behavior of a real drone.

A communication tool between iOS device and the hexacopter or a test computer is Secure Shell (SSH). SSH communicates over Wi-Fi. For this purpose, the open source library NMSSH from Cocoapods package has been used. NMSSH is an SSH wrapper for iOS devices.

Mobile applications for iOS devices are developed in XCode IDE. The application was developed in the Swift programming language. Older applications and libraries, such as NMSSH library, are developed in the older language – Objective-C.

Applications should be implemented with an architecture. An architecture called B-VIPER was used for this project. B-VIPER is an architecture primarily designed for massive projects, such as big bank applications and social networks. The reason why ar-

chitectures are used is a better testability, reuseability, and extendability.

2 Problem Definition

The whole application was developed for and tested on the UAV provided by the Multi-Robot systems group (MRS) [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 a middle-ware that controls everything on the board. Complete description of the helicopter is in Chapter 3. The communication with this board can be done in many ways. This work uses Wi-Fi for the communication between the UAV and the mobile device. The communication is overall described in Chapter 6.2.

ROS allows testing of the UAV on the computer with a simulator of the real world called Gazebo. Gazebo is described in Chapter 4.3. Every experiment can be run in Gazebo before the real flying. That is recommended because an error on the real helicopter can be fatal.

As stated in the assignment of this thesis, the application should be running on the operating system iOS. Implementation of mobile applications for iOS can be done in many programming languages. This application was developed in the programming language Swift. Applications running operating system iOS can only be run on the Apple devices. Therefore, the first problem of this work was solving the combination of Ubuntu, used for running ROS, and MacOS, used for running the IDE. The solution of this problem was either the virtualization of the Ubuntu operating system, or the utilization of the dual boot. Attempts of this simplification are described in Chapter 5.

This work does not contain the construction of an indoor map. That is part of another thesis.

3 Unmanned Helicopter

Figure 1: Micro Aerial Vehicle

In this thesis the major part is a UAV, invented on the Department of Cybernetics by the group Multi-robotic Systems. A requirement of this work is only one UAV. Set of these vehicles is used for student's research and development. For instance 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 the built-in very precise GPS locator. As a result, the drone has better commanding options, for example, specified movement or rotation.

The most used types of UAV are quad-copters with four propellers. However, this thesis uses helicopter is hexacopter, which means that it has six propellers. Reason for having six propellers is preventing expensive parts from being damaged due to the motor or propeller failure. The internal system is able to stabilize helicopter safely land with the malfunction of a motor, which is not possible with quad-copter. Hexa-copters are also used for carrying heavier packages. Six motors provide more power to take off and to fly.

For this project, UAV contains base components that are installed on UAV developed by MRS. These components are the computer with ROS, battery as a power supply, six electric motors with propellers, differential GPS, standard GPS, rangefinder and Garmin rangefinder. Standard GPS is installed on UAV for the case of differential GPS malfunction. Garmin rangefinder is installed for the case of rangefinder malfunction. Differential GPS has better accuracy than the standard GPS.

3.1 UAV

According to the definition, Unmanned aerial vehicle is a vehicle without any passenger or driver onboard. It can be controlled autonomously or remotely. It can carry a 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 main experiments of multi-robotic system group. As mentioned in the introduction, every experiment has numerous tests in a simulator before real visualization. Every program can still fail on real hardware. For this reason, the Collision avoidance system [6] has been developed. This system can avoid wrongly planned trajectory, that can lead to a possible collision. The system uses a multi-master system, where every vehicle knows about each other. Thanks to the multi-master system can the Collison avoidance system predict the future position of every aerial vehicle at the same time and in case of a possible collision, can change the trajectory of drones so that they won't crash.

3.3 Requirements for UAV in this Project

The UAV in this project will use basic systems prepared in default settings. The drone has all sensors for self-stabilizing. Another connected component is the GPS device, that cares about every localization. That device is also present in every helicopter. The localization is not used in standard GPS coordinates, but in the Universal Transverse Mercator (UTM). The UTM coordinates are better described in Chapter 6.2. The last necessary component is Wi-Fi. Each drone has to 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.

4 ROS

ROS [7] means Robot Operating System. It is a middle-ware used by many developers for testing and implementation of algorithms for robots. It covers small uncomplicated robot projects going over our drones, ending with smart robots with extensive 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 the ROS ecosystem. ROS has a big community of developers. Its community has about 1500 members, and there are thousands of threads on their Wikipedia page.

4.1 Uav_core

Uav_core is a package implemented by students of FEE CTU in Prague from Department of Cybernetics. The package implements control methods of UAVs and UGVs (Unmanned aerial vehicles and Unmanned ground vehicles). This thesis focuses on UAV. Therefore UGVs will not be described. Lots of packages have been implemented around uav_core controlling simulated devices, that are used on real UAVs. This thesis uses accessories such as cameras, RFID locators used in this bachelor thesis [8] and many similar components. Uav_core package is used for control of helicopter in this project.

4.1.1 Uav_core commands

Commands for a drone can be executed on the terminal on the computer. It implements few basic commands [9]. The most important functions are moving to absolute or relative positions complexly or divided by axes. Also, the trajectory for a drone can be set. The setting of relative positioning is simple. The unit of each parameter is one meter. For setting absolute position is the work harder. Every command uses UTM coordinates that are usually offset for adjusting the zero position to the center of the testing area. This will be the last step in communication with UAV. The search for a communication tool for an iOS device is described in chapter 6.2.

4.2 UTM coordinates

The Universal Transverse Mercator coordinate system (UTM) is unlike 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 the northern hemisphere [10]. With the UTM was at the first time defined one meter, because the distance from the equator to North pole is 10 000 km. As a result, is that Easting and Northing coordinates are always in meters.

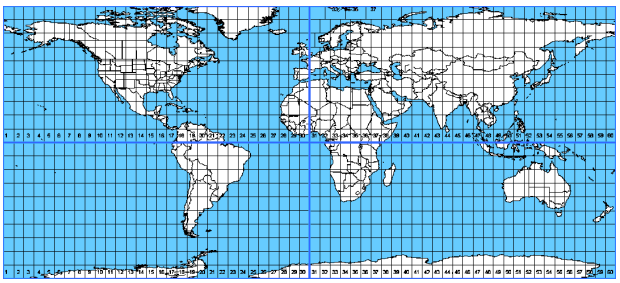


Figure 2: UTM Zones [1]

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 middleware that processes every control of the drone, but for simulation is there Gazebo, robot simulation tool. The gazebo can simulate dynamic systems, robots, sensors such a camera. It can also simulate clouds. For this thesis, the models of UAVs has been prepared. These models typically contain the same camera as the real drone.

Figure 3: Screenshot from gazebo [2]

5 ROS on MacOS

Best way to develop and implement iOS application is to use XCode - recommended IDE for iOS development. XCode can only run on MacOS and on some illegal copies, for example, Hackintosh. When we want to go via the legal way, we can only use the MacOS. Therefore, the implementation requires the usage of ROS that requires Ubuntu operating system and XCode IDE that requires MacOS. To make development more comfortable and straightforward, one computer that runs both operating systems was used. The possible ways how to achieve this are listed in the following sections.

5.1 Dual Boot

The first option which can remove the problem of two machines is to use a dual boot. Dual boot splits the memory into two parts and installs the different operating system in each part. However, Apple devices have a warranty, and by installing other OS on it, the guarantee would have expired. Likewise, installing Ubuntu on an Apple device is not so easy, because these devices are well protected. Even if all problems are resolved, restarting the computer after updating the mobile application to start Ubuntu and ROS will take much time.

5.2 Boot Camp

Lots of Mac users use Boot Camp to run Windows operating system, parallel to MacOS. That is the cleanest way to start the different operating system on an Apple device. And Windows is very stable, because all Apple configurations are known, and there are not any differences with different setups. Unfortunately Boot camp is not compatible with Ubuntu, or another Linux derives.

5.3 Virtual machines – Parallels, VMWare, Docker, VirtualBox

5.3.1 Parallels Desktop

Parallels Inc. is presented as a global leader of the cross-platform solutions. And their product Parallels Desktop [11] is a program for Mac, that can virtualize Ubuntu and other operating systems. With this type of virtualization, it was not possible to run ROS.

5.3.2 Docker

Unlike other listed options, Docker [12] comes with the method that divides all apps into containers. Docker is very often used as a development product for testing new of applications on the "clear systems." It guarantees high security, portability, and flexibility. However, Docker was not compatible with the graphical interface Gazebo and, therefore, could not be used.

5.3.3 VirtualBox

VirtualBox [13] is one of the most used virtualization product, developed by Oracle. Many Linux distributions were tested on VirtualBox. The first tested was classic Ubuntu, which was recommended for uav_core. Running ROS was successful. However, it has about five frames per second (FPS) in Gazebo simulation, but not so stable and easy to develop. After all these tests and research, it was discovered that VirtualBox with Ubuntu has a problem 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. Next derives from Linux, for example, Lubuntu, Xubuntu was tested. These types have the best rating for this type of applications. On both operating systems, better fluency and higher FPS was observed. Problem was in repeated installations, where 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. Neither of tested virtual machines proved to be suitable for running such types of operations, which ROS offers.

5.4 Conclusion of ROS on MacOS

The result of this research should be, for example, a package with the installation script, that will compile and install all necessary components of ROS with uav_core on MacOS. After all tests and tries, it was decided to stop with this way and move back to two devices and implement a mobile app on MacOS and simulate drone on PC with Ubuntu. The result is not very good, because there have to be two computers for development.

6 Mobile App

6.1 Operating system

At the department of cybernetics, four apps were developed this year, out of which three of them were developed for operating system Android. The application for this thesis was developed for iOS as an operating system.

6.1.1 Android Applications

Applications for system Android could be developed in a 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 the stability and efficiency of their application. All my colleagues, which has developed an Android mobile application used Java because they had much experience with Java, and it would be tough 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 the iOS development, the most used programming language was Objective-C usually shortened as an ObjC. ObjC is a programming language based on C, which is one of most known programming languages worldwide [14]. The Xcode is highly recommended IDE developed by Apple Inc. Nowadays, Swift is mostly used language for development, which is similar to Kotlin, mentioned in the Android section, but Swift is older. The latest version of Swift is Swift 4.0. It is the first version that is backward compatible. Backward compatibility was one of the major issues of Swift, while 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. The major pros and cons are described below.

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

Important to be mentioned is a description of application distribution. Both operating systems have application stores, places where users can download applications. Store for Android is Google Play and for iOS AppStore. Android developers have to pay 30 USD to be able to distribute applications on Google Play. A developer gets an unlimited license without expirations for distributing Android applications. Another option for distribution to share a file with *apk* format that file is automatically generated from Android Studio. Every Android user can easily install that file.

IOS application distribution is harder. For distributing on AppStore, it is necessary to have a license. This license costs 100 USD and has a one-year validity. The reasons why this license is so expensive are Apple policy and Apple review team. After the application is uploaded to AppStore, developers must wait for a review. The review is done by one of the Apple review teams. The team checks the completeness of the application, unauthorized system flows, and forbidden themes. The review should decrease the count of applications of poor quality with security problems.

6.2 Communication

As mentioned in the ROS section of this thesis, there is a set of useful scripts to control the drone. With this option comes possibility to connect the app with the drone or simulator via SSH. NMSSH library[15] was used or this communication. Library NMSSH can be downloaded in three ways. First is manual download from GitHub and insert classes into the project. The second one is via Carthage. Carthage[16] is dependency manager, which helps developers to use third party libraries to increase the quality of their applications. Last way is to use a dependency manager called CocoaPods. CocoaPods is the most used dependency manager [17], and it was used for this application. With NMSSH library, the application can start connection via SSH, log in as an arbitrary user, send bash commands, and receive responses for these commands.

6.3 Architecture

In the begging of developing mobile applications, architecture or structure of code was not resolved. The readability and reusability in bigger apps became a problem. For developing of a readable, reusable app, it needs to hold self-descriptive and clean code. For this purpose, app architectures are used. For this project, the B-VIPER architecture was used. It is 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.

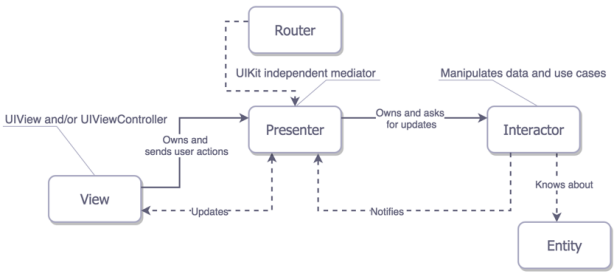


Figure 4: VIPER architecture [3]

6.4 Design

All screens use native basic native graphics components. Two specific views are used. The first one is a joystick from CDJoystick library[18]. This library was used only to joystick commanding. The second one is the Google Maps library for iOS[19].

6.4.1 Navigation Controller

The navigation controller is the controller for screen navigation. It controls transitions between screens, hides, and shows screes. There are two ways, how to work with the controller. The first one is to use the storyboards, that completely take care of the navigation controller. The second one is to take care of the navigation controller as a programmer.

6.4.2 Storyboard

The storyboard is a type of a file, which can be used for the implementation of the mobile application user interface. A benefit is that every change and new screens can be updated from one place. A disadvantages, for example, orientation in the storyboard of application with many screens. An important feature of storyboards as described in this Paper [20], is that it contains an opportunity to add a transition between screens. Routing is implemented with this transition. The storyboard is a file which can be edited in Xcode with tool Interface Builder. As convenient as it might be, programmers nowadays rarely use storyboards and prefer implementing the navigation themselves.

6.4.3 Programatically Implemented Layouts and Transitions

As mentioned in the description of Storyboards, a developer can use as the same language for implementing the layout as for implementing other parts of an application. Swift offers the same features as storyboards do. The view is not always set absolutely, but almost every view is positioned relatively. A benefit of these settings is that created layouts fit every resolution and sizes of iOS devices. Every screen can be designed separately. The exceptions are, for instance, preparing the layout of screens for tablets and mobile phones, this separation makes sense.

Navigation controller is represented by a class called *UINavigationController*. It contains various functions, such as *pushViewController(_ viewController: UIViewController, animated: Bool)* this function present a new UIViewController, optionaly animated. UIViewController is a 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

Open sourced external libs were used for more convenient development. In this application has been used dependency manager called CocoaPods [21]. This is mostly used dependency manager. From used libraries, I would like to mention Stevia Layout [22]. Stevia is a library that simplifies graphics implementation as part of an application directly in Swift instead of using storyboards.

7 Main Page

The first page contains three fields to insert the IP address of the controlled drone, the name of the user account and the password. A next field is a UAV number, which would specify the drone for the actual session. The layout of this page can be seen in Figure 5.

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

There are four types of communication - the Joystick commanding, the Google Maps trajectory, the Indoor map trajectory, and the Terminal communication. Every communication except terminal is described in its corresponding sections. A Terminal was implemented only for development purposes. It works as a classic terminal on a desktop computer. One input cares of commands and the output are responses from commands.

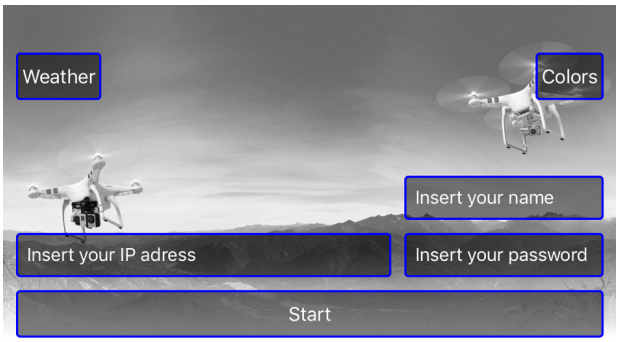


Figure 5: Login page to application

8 Joystick Commanding

The first control type is joystick commanding, which is a basic control.
Figure 6 captures the screen with two joysticks. The left one controls the rotation and height of drone. The right one is for moving forward, backward, left or right.

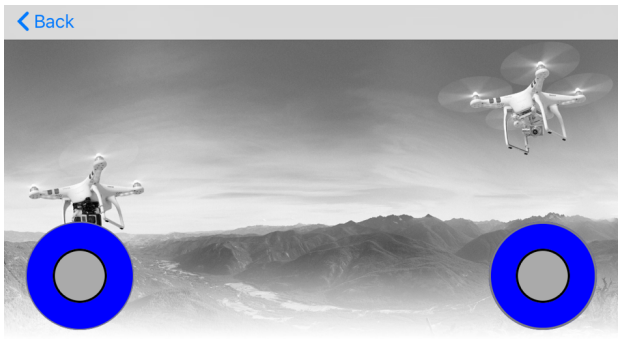


Figure 6: Joystick controlling screen

In the begging of the development, new issues with SSH communication settings arise. The NMSSH library uses non-interactive access. After a short research, a solution was found. Afterward, testing was started. Testing on the simulator was successful, but on the real drone, SSH showed that it is not useful.
The problem was in the speed of the SSH connection. The standard command takes about one second and one second was also a pause 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 the drone in real time is via sockets. In this way, the frequency can be increased from 1 Hz to about 100 Hz. This form of communication used colleagues from the same group for controlling UAV with Android devices. This project does not contain an implementation of communication via sockets.

9 Google Maps Trajectory

Next type of drone control is the Google Maps trajectory. This way can be the groundwork for more sophisticated planning algorithms.
On the Figure 7 is Google map. After a click on some place marker will be placed there. This marker represents one of the points where the drone has to fly. These points can be easily added and moved. Once the road id prepared, the user clicks on the square button on the left side of the screen. It approximates the trajectory to the smaller distance between each point and sends this trajectory to the drone. In the right top corner is button **goto**, which represents a command that sends the drone to his start position. If a drone is one the start position, button **start** can be tapped, and the drone will start to follow the set trajectory.

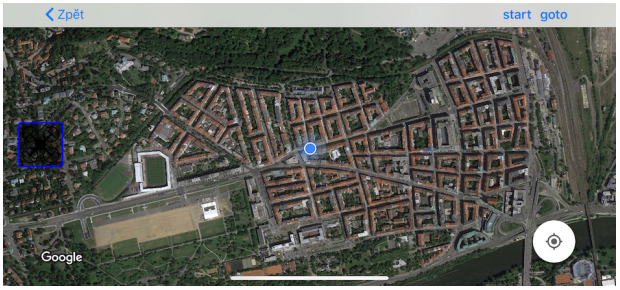


Figure 7: Map controlling screen

Few experiments were at the UAV camp, which is one week of testing all experiments for bachelor, magistrates, and doctoral students. These experiments will be described in the next section.

9.1 Installation Google Maps Library

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

9.1.1 Bundle ID

The bundle ID is a unique identifier for every application. This ID contains the region identification, the name of the company and name of the application all separated

by dots, e.g., *cz.fel.cvut.droneapp*. It can also consist of an environment. The application environment is a specification of the application. It 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 the client or the crowd testers, and release version goes only into AppStore.

10 Indoor Navigation

10.1 Overview

Last part of this project is an implementation of indoor navigation because the drone will be controlled the same way as in Google Maps. Implementation of this feature has some parts that are similar to the implementation of the Google Maps trajectory.

10.2 Preparation

The app needs a model for indoor navigation. 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 such as area mapping and object modeling.

10.3 Ways of Representation

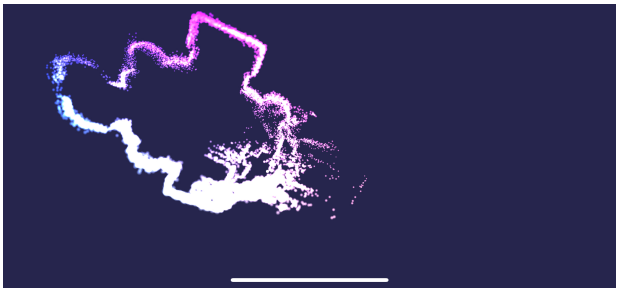


Figure 8: Screen with displayed Metal Kit

10.3.1 MetalKit

The first way to display point cloud data is to use MetalKit [23] 8 as a native iOS component. This component is available from iOS version 8.0. It is not complication because the minimal version of the app is 10.0, that corresponds to iOS development standards. These standards recommend the support at least the last two version of the operating system. The latest version 12.3 is from the year 2019. 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 the approximation, that will be described in the next section. The last and the biggest problem of MetalKit is user interaction. Three dimensions are very comfortable to observe and watch models and data, but very uncomfortable to insert points which will be necessary to plan trajectory.

10.3.2 UIBezierPath

UIBezierPath [24] is also a native Swift component. 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 points but draws lines. This problem can be easily resolved with some type of sorting by the nearest point. This procedure will be described in the next section. But again this way is not very straight for PCD representation.

10.3.3 View in View

The last way of representation of PCD is to use classic UIView [25]. UIView is a basic iOS application view component to develop an iOS application using Swift language and Xcode. It is available from the lowest version of the operating system, and it is easy to use. 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 the next section. After sorting all data will be separated into ten layers in the 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 a random axis. QuickSort 1 is used for sorting. Axis, which I had to choose is x. The second step is to decrease the density2 of points by the removing points, that are too 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 similar to the preparation for MetalKit. The first step would be to sort points in the z-axis. 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. The 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. Points have to be sorted by z-axis and separated into layers.

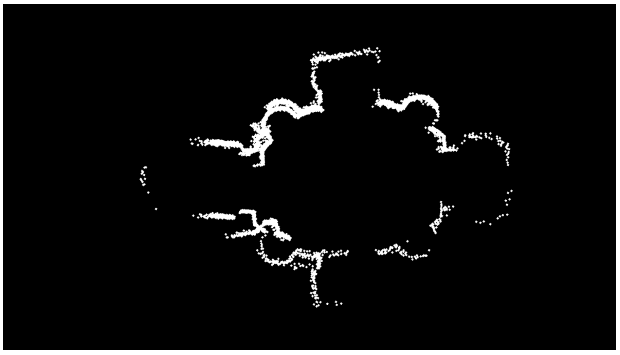


Figure 9: Points as view in view

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 the next section.

10.5 User Interaction and Tutorial

Screen with indoor navigations has two active blocks. The first one is the slider, which controls a 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 that the trajectory is properly set. In order to set the trajectory, the user has to long-press the screen in the desired place. The interaction is the same as in Google Maps. Replacing points is also similar, it is done by holding of a finger on the marker. Once the trajectory

is prepared, there are three buttons to set trajectory, go to start trajectory, and follow the trajectory.

11 Trajectory Preparations

Every selected trajectory have to be approximated, smoothed, and prepared for each environment. This process is described in the next sections.

11.1 General Preparation

After user interaction with an application, there is an array of the coordinates, which have three dimensions. In order to use it for ROS, the trajectory has to be 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 the tracker of the UAV is implemented to fly between every two points in a 0.02 seconds interval.

11.2 Preparation for simulator

Both implementations of navigation were tested in simulator. There is a library [26] for Gazebo, that simulates Google Maps behavior (coordinates and graphics), but it wasn't used in this project. Separated feature that synchronizes the real coordinates selected on an iOS device, to Gazebo coordinates, was implemented. Synchronization move virtual center of the Google coordinates to the actual position of the mobile device. Testing of indoor navigation in a simulator is very straightforward because points selected in mobile application intersect 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 (the 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 sending. 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, 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 the next parts. But the result of the joystick was not the best at last. The experiment in the simulator was so not so straight as the one on the real UAV. Communication via SSH was too slow, and the movement of UAV was discontinuous. This suggests that SSH connection cannot be used for the 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 the 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. The graph is processed in MATLAB. Screenshot of an iOS device with an inserted trajectory contains ten markers. The trajectory is only six markers in the center, remaining four points are borders of safety area, for flying with drones. 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

Tests of indoor navigation were executed neither on UAV nor on the simulator, the problem was in integration with Gazebo and ROS systems, where I was unable to render PCD. PCD is the format of points in 3D.

12.4 Problems and Possible Solution

After all experiments, few issues can be implemented in the next thesis or as the 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 several UAVs as user
- Implement node that can send captured data of the map to a mobile device

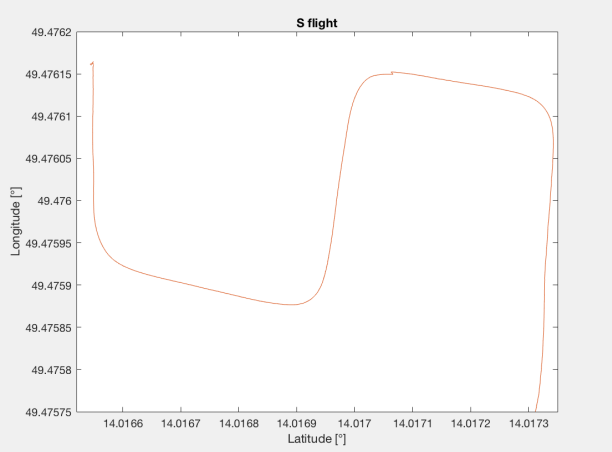


Figure 10: Longitude and latitude graph from experiment

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 the next development. Development of an 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 the 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 the 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] "Top 10 most popular languages," accessed: 2019-05-24. [Online]. Available: https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10

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

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

[17] "Choosing the right ios dependency manager," accessed: 2019-05-24. [Online]. Available: https://aimconsulting.com/insights/blog/choosing-the-right-ios-dependency-manager/

[18] "Cdjoystick github repo," accessed: 2018-04-15. [Online]. Available: https://github.com/Coledunsby/CDJoystick

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

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

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

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

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

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

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

[26] "Osm plug-in for gazebo/stage," accessed: 2019-05-24. [Online]. Available: https://github.com/l0g1x/gazebo_osm

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