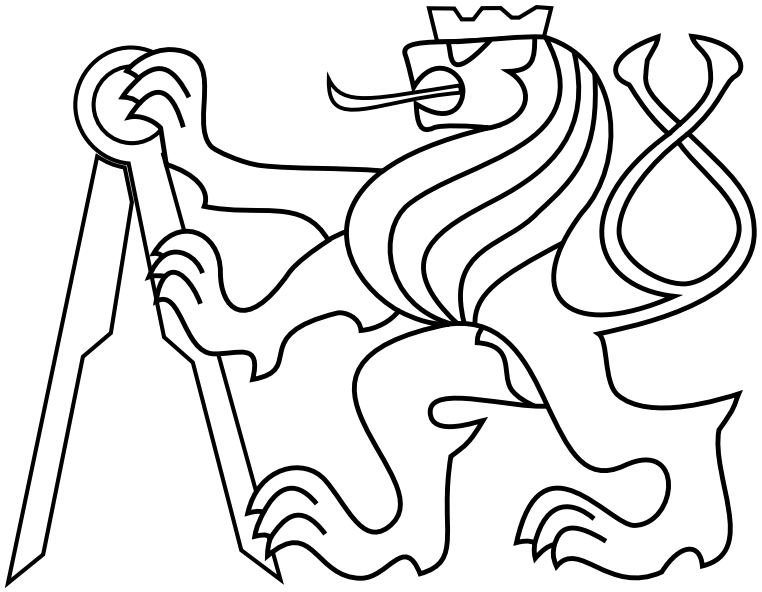
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Filip Bursik

Control and navigation of an unmanned helicopter using sensors in mobile phones

Department of Cybernetics

Thesis supervisor: **Dr. Martin Saska**

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Bursík Filip** Personal ID number: **457002**

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:

Control and navigation of an unmanned helicopter using sensors in mobile phones

Bachelor's thesis title in Czech:

Řízení a navigace bezpilotní helikoptéry s využitím senzorů mobilního telefonu

Guidelines:

The goal of the thesis is to design, implement, and experimentally verify in Gazebo simulator and real experiments an application for control and navigation of an Unmanned Aerial Vehicles (UAVs) in task of following an unauthorized helicopter flying in forbidden areas.
1. Design and implement an Android application for UAV control and navigation by positioning of the mobile phone into the direction of the flying target. Two different approaches will be implemented and experimentally evaluated. In the first one, operator keeps the unauthorized UAV in the center of a mobile phone camera, which defines a vector to follow by the controlled UAV. In the second one, an edge of the mobile phone is used as a direction finder to define this vector.
2. Design an interface that enables to control a UAV equipped with onboard computer with ROS (Robot Operating System) [1] by the mobile phone in real time.
3. Verify system functionalities in Gazebo and with a real platform in outdoor conditions. The goal of the experiment will be to get the UAV into proximity of the flying target to enable its localization by onboard cameras. The consequent target localization and possible elimination is not part of this thesis.

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] K. P. Valavanis, G. J. Vachtsevanos. Handbook of Unmanned Aerial Vehicles. Springer Publishing Company, 2014.
[3] V. Kumar, N. Michael. Opportunities and challenges with autonomous micro aerial vehicles. The International Journal of Robotics Research. Vol 31, Issue 11, pp. 1279 - 1291, 2012

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.

Prague, date ....

Acknowledgements

I would like to thank my classmates, who worked on the mobile version, too, for their advice. My supervisor Dr. Martin Saska for his patience, for his time and for his support and all his team Multi-robot Systems for help all over the term and especially on the camp, where he helped me a lot with finishing the application and final test on the drone and the measurement necessary to determine the accuracy of my application. And finally, my family for all the support and help during the term.

Abstract

The aim of the work is to design, implement, and experimentally verify in Gazebo simulator and real experiments an application for control and navigation of Unmanned Aerial Vehicles (UAVs) in an application of following an unauthorized helicopter flying in forbidden areas. Design and implement an Android application for UAV control and navigation by positioning of the mobile phone into the direction of the flying target. Two different approaches will be implemented and experimentally evaluated. In the first one, operator keeps the unauthorized UAV in the center of a mobile phone camera, which defines a vector to follow by the controlled UAV. In the second one, an edge of the mobile phone is used as a direction finder to define this vector. Implementation two different interfaces that enable to control a UAV equipped with the onboard computer with ROS (Robot Operating System) by the mobile phone in real time and describe them. Make verification of the system functionalities in Gazebo and with a real platform in outdoor conditions. The goal of the experiment will be to get the UAV into proximity of the flying target to enable its localization by onboard cameras.

Abstrakt

Cílem práce je navrhnout, realizovat a experimentálně ověřit v Gazebo simulátoru a na reálném experimentu, aplikaci pro ovládání a navigovaní bezpilotních vzdušných vozidel (UAVs) v aplikaci následování neoprávněné helikoptéry létající v zakázané oblasti. Navrhnout a realizovat Android aplikaci pro ovládání a navigaci UAV pomocí umístění mobilního telefonu směrem k létajícímu cíli. Navrhnuty a experimentálně vyhodnoceny budou dvě různé metody. V první metodě, operátor udržuje neoprávněné UAV ve středu kamery mobilního telefonu, který definuje vektor pro sledování, který bude sledovat řízené UAV. V druhé metodě, bude okraj telefonu použit jako vyhledávač směru pro definovaní vektoru. Realizace dvou rozdílných rozhraní, které umožňují ovládat UAV, vybavené palubním počítačem s ROSem, mobilním telefonem v reálném čase a popsat je. Udělat ověření funkčnosti systému v Gazebu a s reálnou platformou s venkovními podmínkami. Cíl experimentu bude dostat UAV do blízkosti létajícího cíle k umožnění lokalizaci pomocí palubních kamer.

Contents

List of Figures

1 Introduction

The popularity of helicopters has been grown up in recent years. They are favourite for their cheap components and possible modifications. Many industries offers assembled quadrocopter and on the other hand, Chines distributors offers components to assemble the helicopter. Industry such as DJI [9] offers quadrocopters with semi or full autonomous control system. Some of these helicopters are used for transporting packages. Alongside with this, helicopters are going to be used for more application in the future. There is still a lot of undiscovered possibilities. For example the controlling of an helicopter (marked as UAV in this work) using mobile phone is well-known issue.
In the past, for commanding helicopter was used radio communication with high frequency and range, but nowadays everyone has a smartphone. Thanks to this fact, smartphone is available stuff and use it for control the helicopter is more modern and provides more possibilities. Mobile phone is assembled of a lots of sensors and hardware for realization the system. However, not all these sensors are suitable for precise positioning. Since the helicopter is now available almost for everyone, it is necessary to eliminate possible threat. To make elimination easy, it's important to have well designed system for controlling the helicopter.
Three approaches to make control system: joystick, use mobile rotation and the last one: define drone position by coordinates system. Each one is different and use different techniques. Commonly used controlling system are joysticks. Left joystick for horizontal move and right joystick for rotation. Well designed technique for real-time controlling used also in radio communication with helicopter. Define the drone position is approach propose to send helicopter to defined position and not for real-time commanding. Use mobile rotation determined by mobile sensors is method similar to joysticks - designed for real-time control and brings a number of benefits. Such as: better awareness of final position.
Another vision is to use two phones. Both of them aim to the point and the final position of the helicopter is the intersection of both vectors. This solution requires more sophisticated algorithm. The helicopter has to know both vector and find the intersection and set it as final position or has to be connected to local saved, which stores the vectors. Purpose application of this concept is also localization and elimination of the unmanned helicopter. However, it is more difficult to control it.
These presented systems can be used for many utilities - simple control to record the video or take the photo. The second one is to localize the unmanned helicopter in forbidden area and get to proximity to make identification and elimination. Identification can be made by onboard camera and robot operating system installed on helicopter's PC. PC contains an algorithm for recognizing helicopter from the picture. When the helicopter spot unmanned helicopter, the elimination system can be started. Possible elimination is realized by throwing net, whose job is to prevent unmanned helicopter motors to rotate and the helicopter will fall down. Elimination is not part of this research.
Afterwards, the designed system, which was presented, was verified by simulation in Gazebo and deployed to real helicopter and made several experiments with Multi-robot Systems

Group at Department of Cybernetics, Faculty of Electrical Engineering (FEE), Czech Technical University in Prague (CTU). Final results and measurements are described below in this work.

(a) Helicopter on the ground, used for testing the system

(b) Helicopter in the air, used for testing the system

Figure 1: Helicopters used for system deployment

2 State of the art

Several applications using smartphone for controlling the UAV were presented in recent years [19] [20]. Also tutorial for [20] application [8]. They usually contain a lot of limitations for modifications. Source code for TurtleBot application can be found on GitHub [28], from these source codes comes some implementations in this work. Video showing application of [19] [30]. There are not so much research in this area. ROSJava used for communication is no longer supported, the documentation [27] is outdated and not so many examples can be found. Applications [19] [20] are usable for communication, but only for simple function, and they are not extendable for required features.
Research into interface between ROS and Android platform, has been made in BUT (Brno University of Technology) [38]. Goal of that master's thesis is find out the universal user interface for communication with ROS. Another master's thesis useful for this work [37]. The problem of interface implementation is detailed described in bachelor thesis [33] [34] [35] researchers from Multi-robot Systems CTU [22], where bachelor thesis by Filip Bulander [33] focuses on interface between iOS and ROS and Petr Jezkes's, Petr Kloucek's works focuses on interface between Android application and ROS. Approaches require front onboard camera as additional hardware. GPS have to be included in all UAVs, for control. However, most parts of the ROS is made by MRS [22] group. The Gazebo simulator, nodes and packages necessary for this work. This work is based on these designed features. List of thesis and publications can be found at their website [23]. However, most of the publications focus on research of formatting and controlling the swarm of UAVs, this work is made by cooperation [33] [34] [35], the goal of this thesis is not similar to this work, but the researchers focuses on the interface, too. More accurate, interface between mobile application and UAV's PC.
Designed communication is inspired by study [39]. This study shows the possible communications and their implementations in Android. Newer smartphones offers Wi-Fi, Bluetooth and NFC. The NFC is communication only to 4 centimeters. Bluetooth allows transmit data to distance 10 meters. Wi-Fi offers range up to 60 meters. Bluetooth and NFC showed to be unusable for real-time communication. Work doesn't contain details of these technologies. For connection to UAV, the phone needs to contain 5 GHz Wi-Fi module.
Work includes study of converting GPS coordinates (latitude, longitude) to UTM coordinate system. UTM coordinate system is used to navigate UAV. There are many documentations and implementations example to achieve that [1] [**?**]. For verification is used online convertor [13]. The Geoplaner is not so precise, but for verification it is satisfactory. Unfortunately, Geoplaner cannot read a decimal number in UTM coordinates.
Primary focus of this work is to present real applications of controlling UAV from mobile phone while taking real hardware and software limitations into consideration. This work introduces a possibility of interface and limitations in outdoor environment, where GPS is a reliable positioning system. Usage of GPS is necessary to work all the features of system. Furthermore, work is focused on finding the possible solution and comparing each them. UAV contains features necessary for the control and reading datasets from sensors. Safety

of mobile control is important part of the development and have to be researched before it is used in real application.

3 Problem definition

The purpose of the work is to design interface between mobile platform, implement it and finally verify it in simulation and real drone system. System is for navigating drone using the rotation of mobile phone, make simple Android application with primary functions such as: get information about UAV (battery voltage, battery percentage), command the UAV, control UAV using joystick and console for onboard PC. Research includes implementation of two methods for controlling UAV from the mobile phone, different communications between onboard PC and installed ROS. Includes verification these methods and communications technologies, in Gazebo simulator, after that test the best one on the real UAVs and compare the results, compute the accuracy of the methods, sensors, find the fastest and the most reliable communication.
Important is to understand Android system, which is selected, because of diversity and the possibilities that Android offers unlike others mobile OSs. This research is made for system to localization and elimination unauthorized helicopter flying in forbidden areas using an equipped onboard camera. By helping this system, it will be able to get close to unauthorized UAV and then can be made localization and elimination the UAV, but it's not a part of this research. To reach this goal, it have to be find out the accuracy of mobile sensors. The important criterion is speed and accuracy of the methods, which decides on usage in the real case and possibility to determinate the UAV final position. Position is calculated in mobile device. Calculated position is shared throw Wi-Fi to the UAV. This features offers ROS.
Used hardware is a smartphone, the helicopter with onboard PC (Installed Linux OS - Ubuntu and ROS), GPS for tracking the position, camera for detection of unmanned UAV and RTK for more precise positioning. GPS is positioning system to the determinate position, using satellites to achieve it. By knowing distance from three satellites, the position can be calculated. RTK is more accuracy version of GPS. GPS can determinate position with 2-4 meters accuracy. RTK can determinate with centimeters accuracy. However, smartphone doesn't contain the RTK, only GPS and Wi-Fi. GPS is limitation for getting exact position. Wi-Fi range ($\tilde{6}$0 meters) is the second limitation. Possible communication technology is Bluetooth. Every smartphone and Raspberry (UAV onboard PC) contains Bluetooth. In the case of real-time communication is not acceptable the range of the Bluetooth signal ($\tilde{1}$0 meters).
The assumption for exact positioning is to know the altitudes. First one - mobile altitude and second one - final position altitude. However, without an internet connection, it is not possible to find out final position altitude, the system assumes that mobile and the final position, where the UAV has to flight, are on the same altitude and make this approach the source of the inaccuracy. Altitude of smartphone can be detected by barometer. However, barometer is not included in every smartphone.
The aim is not to design final product, but find out possible approaches. Therefore, work contains accuracy measurements and prons and cons of the methods. Some of the methods appeared to be inappropriate. Android application is designed for androids with version

4.4 and higher. Android device has to contain all of the sensors, described in 5.3.2. The application is only for phone. Watches and tables are not supported, yet. Permissions for access camera, location and internets have to be granted for correct function. To connect to the UAV, the phone needs to have 5 GHz Wi-Fi module. Phones without this module cannot see drone's Wi-Fi.

3.1 Terms

ROS - provides libraries and tools to create robot application. Provide hardware abstraction. ROS is open source, meta-operating system running in our case on Linux.
ROSjava - library, which provides communication between ROS and java.
SSH - security communication over the network using TCP/IP.
Android - Mobile operating system.
Interface - manage connection and communication between two devices or programs.
Fragment - screen in Android screen. One Android screen has more fragments.
Layout - android application screen, written in XML.
Gazebo - 3D simulator for simulating robots and objects.
TCP/IP - family of protocols, contains set of protocols used for communication over the network.

4 Robot operating system (ROS)

ROS is a necessary part of this thesis. It's a building stone for the designed system. ROS is running on onboard PC and mobile phone is connected to that PC on the same network. Basic information about ROS can be found on Wikipedia [17]. For more detailed information - wiki directly for ROS called ROS wiki [26]. ROS is available for Linux, MacOS. Usage of ROS is rapidly growing. Either for university purposes or industrial purposes. Industries such as Shadow Robot[32], Meka[16] (no longer exists) or Robotics Equipment Corporation[31] are using ROS as a software platform for their customers to build on. Additionally, some of these industries are using ROS to build their applications. ROS is also used for research and development purposes, in universities and research facilities.

4.1 Introduction to ROS

Robot Operating System (ROS) is a collection of software frameworks for developing robot software. ROS is not a complete operating system. It provides services designed for heterogeneous computer cluster such as package management (4.3), hardware abstraction, low-level device control, message-passing between processes [17]. For implementation functionalities in ROS is recommended C++ and Python programming language. ROS helps to simulate sensors, motors, etc (Hardware abstraction). ROS can be run on Linux or Mac operating system. In this work is used Linux Mint for simulation, but onboard PC has Ubuntu, which is supported for ROS. UAV commands can be executed from the terminal of onboard PC or SSH to the PC and use ROS commands. Notice that the PC have to be on the same Wi-Fi.

4.2 Used function from ROS

ROS is composite of nodes and services. Each node communicates over the topic. The node can be subscriber or publisher. Publishers are publishing messages and subscribers are listening to this messages on current topics. This publishers and subscribers are called nodes. Another way to send command is over services. For commanding, UAV are used prepared services and topics. This topics and services were made by MRS group [22]. Pro preserving measured data is used *ROSBAG* util. This is a perfect tool to manage and store the topics. Only selected topics will be saved to the ROSBAG, saved *ROSBAGs* are used for later analysis. To get all these topics from *ROSBAG*, there are two options - first option is to use Matlab. Matlab can parse these bag and separate each topic to structure. Structures can be later displayed. Thanks to this fact, Matlab can print all store topic in dependence on the time. The second option - use Python script. *ROSBAG* allows storing the video from the camera.

4.3 Catkin

Catkin is a build system for ROS, used for building packages. Most information and tutorial to installation catkin can be found on ROS wiki [25]. Necessary system for building the UAV core and custom packages. Custom packages are written in C++ or Python (In this work is preferred C++). After a build, the launch files are created and used to starting the written code. Catkin is a part of ROS.

4.4 Gazebo

The Gazebo is a well-designed 3D simulator. The Gazebo offers tests for robots in complex indoor and outdoor environments. The gazebo is free. ROS has support for Gazebo. Gazebo allows simulating of multiple robots (UAV) in one world (map). For simulation in this work, is used only one UAV. Additional, object can be added to the world or completely all world can be changed. The Gazebo offers X, Y, Z coordinate system using for moving or positioning the objects.

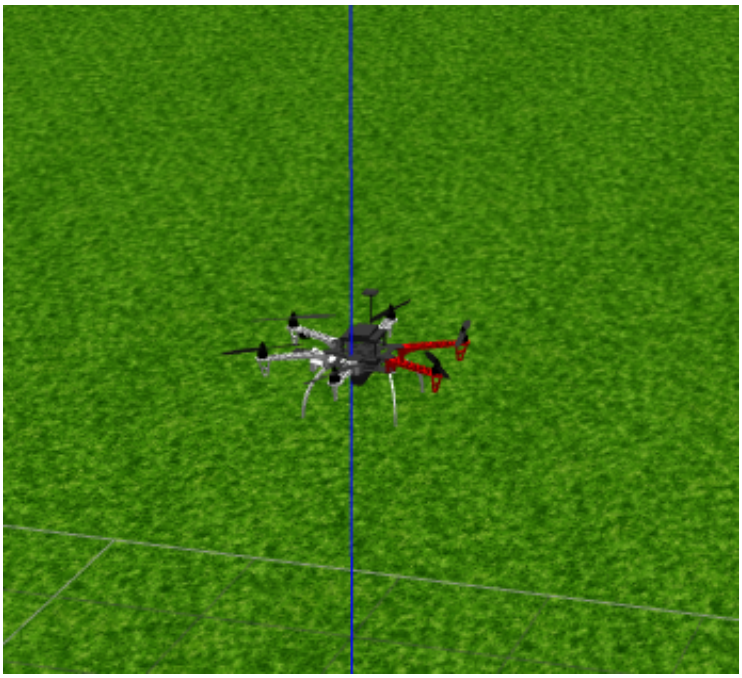


Figure 2: UAV designed in Gazebo simulator

5 Android application

Preferred mobile OS for this work is Android, because of the possibilities and libraries, which offers this OS. It's not necessary to use this OS, procedures and theories also work on iOS, but with differences in implementations, due to the diffuse creation of the application. Features of application: get battery state (voltage, percentage), control UAV using joystick, command UAV (positioning by UTM coordinate system), control UAV using mobile rotation (two methods of control described in 5.1, 5.2 and of course connection features. Testing of these features requires implementation of two applications. First one - intended for *SSH* communication, this technology is more explained in 6.1, second one - *ROSJava* explained in 6.2.

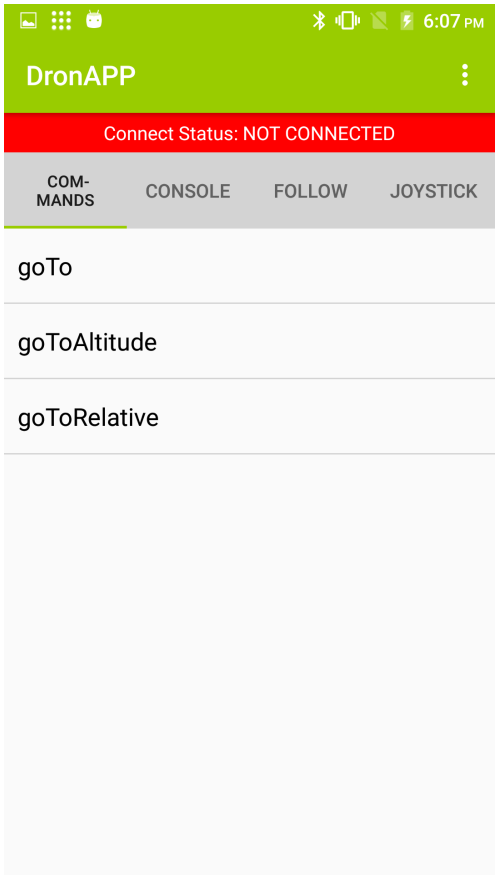


Figure 3: Screenshot of the main screen of the application

Each application is different in many ways, but architecture, compatibility and tests are almost the same for each one. The effort was to move most of the logic to the PC onboard, because of the difference between the approach, it is not possible to make a universal application.

5.1 Description of implementation of the first approach - Using the mobile camera to calculate following vector

Mobile camera approach use assembled camera in a smartphone and determine the final position by the center of the screen. Android offers native support for displaying camera image in the developed application. Sensors are responsible for getting the angles necessary for the approximation of the vector. More about sensors and their usage in section 5.3.2.
It's necessary to get points of the compass. It has to be also known the absolute elevation angle. From this two angles, the vector of the direction can be calculated and multiplied by the required distance. The distance is determined by the progress bar - slider. It maximum or minimum can be set optionally.
Mathematical equations used in the calculation of the vector:

$$x = cos(\beta)cos(\alpha), \quad y = sin(\alpha)cos(\beta), \quad z = sin(\beta) \tag{1}$$

This approach of calculation can be used in the simulation. For math operation is used default Math library, which Java offers. Pseudocode of implementation function, which calculates the direction vector:

Algorithm 1 Function for calculating vector in simulation environment

```
1: function getVector
2:     bearing
3:     elevation
4:     x = cos(elevation)cos(bearing)
5:     y = sin(bearing)sin(elevation)
6:     z = cos(elevation)
7:     return x, y, z
8: end function
```

The function takes two arguments - elevation angle and bearing angle. Elevation angle indicates the angle between floor and mobile tilt. Bearing angle indicates points of the compass. Returned vector is composite of calculated point [x,y,z].
However, for usage on real UAV, the calculated vector have to be converted to the GPS or UTM coordinates. More effective sight is to take only heigh from the vector (z) and X and Y value calculated from GPS position received from GPS module included in the smartphone. The position is moved by required bearing angle and distance. To move location is used function from StackOverflow [12]. The function takes location (latitude, longitude), distance, angle and it is also covered by unit tests. More about unit testing in section 5.5.1.
Mathematical equations:

$$x = asin(sin(latitude)cos(distanceFrac) + cos(latitude)sin(distanceFrac)cos(bearing)) \tag{2}$$

$$y = (longitude + (atan2(sin(bearing)sin(distanceFrac)cos(latitude), \\ cos(distanceFrac) - sin(latitude)sin(x)))) + 3\pi)\%(2\pi) - \pi \tag{3}$$

Bearing angle is the same angle as alpha in previous calculation. X is final latitude and Y is final longitude. *distanceFrac* is calculated from the moving distance and earthradisu (6 371km).
Important to notice is that this calculation can be source of the unaccuracy. Using float type can cause the difference between the expected value and result. This aspect cannot be eliminated. The change in the hundredth may shift the trailing position by the meter. Pseudocode of the algotitmh:

Algorithm 2 Function for calculating moved location

```
1: function moveLocation
2:     bearing
3:     distance
4:     myLatitude
5:     myLongitude
6:     radius = 6371
7:
8:     distance = (distance/1000)/radius
9:     bearing = toRadians(bearing)
10:    myLatitude = toRadians(myLatitude)
11:    myLongitude = toRadians(myLongitude)
12:
13:    finalLatitude = toDegrees(asin(sin(myLatitude)*cos(distance)+cos(myLatitude)*
   sin(distance) * cos(bearing)))
14:
15:    finalLongitude = myLongitude + atan2(sin(bearing) * sin(dist) *
   cos(myLatitude), cos(distance) − sin(myLatitude) * sin(finalLongitude))
16:    finalLongitude = toDegrees((finalLongitude + 3 * PI)%(2 * PI) − PI)
17:
18:    return finalLatitude, finalLongitude
19: end function
```

Returned value of this function is moved GPS location in form latitude (finalLatitude), longitude (finalLongitude). Function *toRadians* is used for converting angle from degrees to radians. Function *toDegrees* is used for convertion angle in radiands to degrees.

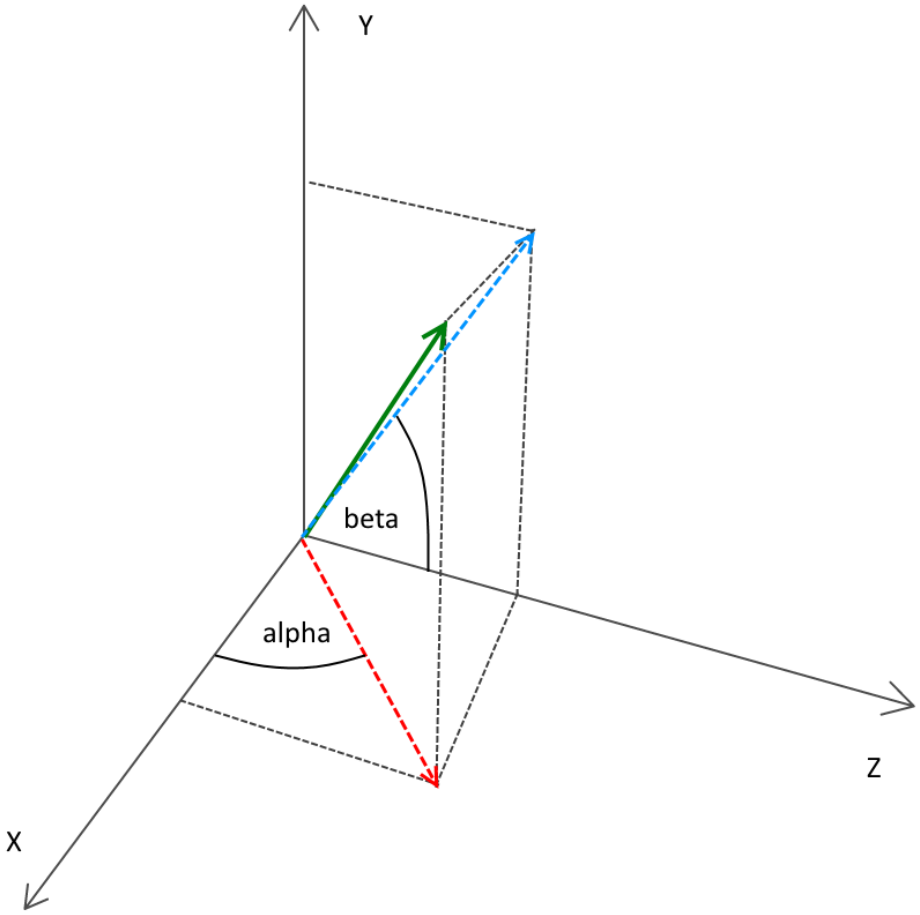


Figure 4: Definition of axis labels and angles labels [10]

To get these angles, mobile sensors have to be used. Used sensors, their accuracy and the pros and cons will be more explained in 5.3.2.

Figure 5: Screenshot of the camera following approach implemented in android application

Because of the showing camera image, this method needs better hardware equipment and quality camera.

5.2 Description of implementation of the second approach - Using mobile edge to calculate following vector

Using mobile edge is an approach where the vector is determined by the edge of the phone. Specifically, the right edge of the phone, when the phone is in the standard position of the display goes to the user.
The calculation of the vector is the same as is explained in 5.1. The algorithm to determinate angles using sensors is different against the first method. Algorithm can be found at [10].

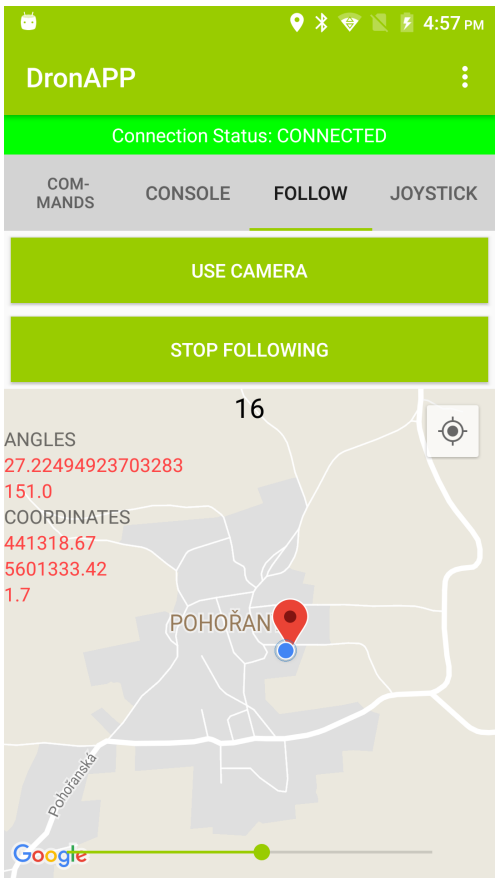


Figure 6: Screenshot of edge following approach implemented in android application

For better debugging during the development, the application contains google maps, where can be seen the calculated position of the UAV and also angles and UTM coordinates.

5.3 Used technologies and libraries

Android application is written in JAVA instead of Kotlin (programming language for Android, too). Java is compatible with Android and has better support for ROS that Kotlin. Layouts and fragments are programmed in XML language (native language for Android UIs). XML language is supported by Android studio. The necessary part of the Android application is a build system. The build system is responsible for libraries, dependencies, build version and used APIs. Android studio natively supports Gradle build system, which is enough for this work.

5.3.1 Libraries

Libraries are additional source code, written by someone else. To make implementation easier, these libraries are used in this work:
Butterknife - responsible for injection view from XML into the Android activities [2]
Dagger2 - library for dependency injection, very useful for making the correct architecture [14]
GoogleMaps - google maps library, used for maps, which helps with debugging [15]
Joystick library - necessary for control using the joysticks [3]
ROSJava - responsible for communication between ROS and mobile phone [27]
Espresso - used for UI testing [29] Each library is responsible for some part of the feature. Without *ROSJava* library, the communication between ROS and smartphone cannot be executed, etc...

5.3.2 Sensors

List of the mobile sensors necessary to the functionality of the system:
GPS - used for getting current position, the accuracy depends on the mobile phone and the position (indoor, outdoor, etc...)
Barometer - used for getting current pressure and calculate altitude, unfortunately not used, because this is a sensor, which is not ordinary in smartphones
Geomagnetic field sensor - used for getting points of the compass
Accelerometr, magnetic field sensor - used for getting values, used for calculating elevation angle
List of the UAV sensors necessary to the functionality of the system:
GPS - determinate UAV's position and used for controlling the UAV It should be noted that this list is valid if the UAV includes a system for control and regulate the flight.

5.4 Architecture

Architecture for this application is a modified MVP architecture (model, view, presenter). The application contains a logic part (model), the model layer is divided to three parts Domain, Application, and Resources. This layer is responsible for communication and storing application data. The presentation layer is responsible for formatting data and handles user events and view layer is simple UI, which is responsible only for showing data and forward the events to the presenter. This division into layers is critical regarding testing. More about testing is in 5.5.

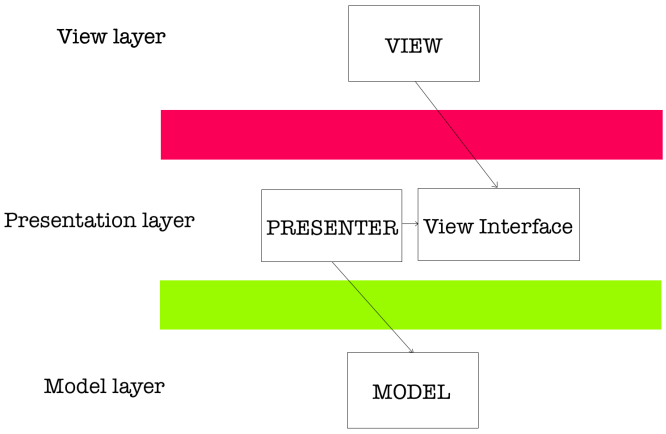


Figure 7: Diagram of the architecture layers

This is a diagram, where can be seen the dependencies between individual layers. Dependencies mean that only higher layer can have reference to the lower layer. When the view layer is changed, higher layer - presentation should not be changed. To reach one way of dependency, it has to be used dependency inversion principle, which is realized by the View interface. These theories are inspired by CleanCode book [36]. CleanCode book is about the rules and convinces respected to keep the clean code. For clean code, the concept is important to adhere that only view and platform layer are using external libraries, which are very hard to test and source of the possible errors. The division of the model to three layers is not necessary (when it's not divided, then this architecture is called MVP, and it's used all around the world for software implementation), but for making the clean code, this structure is adhered:

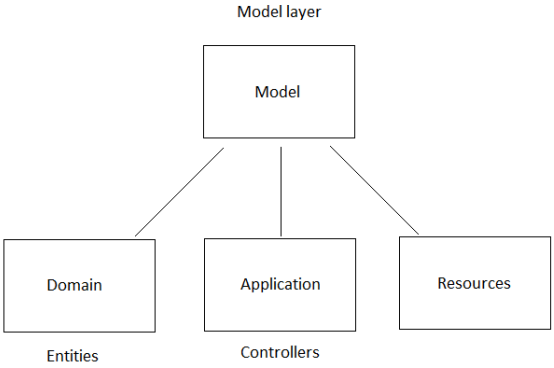


Figure 8: Diagram of the model layers

8 shows the division of the model layer into three layers. Domain, Application, and Resources. Resources are responsible for accessing the external resources. For example shared preferences, used for storing information in the device, external APIs, and libraries are implemented in Resources package. The application holds the application state. Manage data on the screen and call resources for data. These classes in application layer are called controllers. Domain layer contains only entities, using to store the data and pass an argument in some structure.
Dependency injection is reached by android library Dagger2 [14]. Callbacks are made by interfaces. It's good to use reactive programming for this architecture, but for this thesis, it was not the goal.

5.5 Unit, Android and UI testing

Most of the both application's code is covered by tests. The tests can be divided into three section - unit tests, android tests and UI tests. Each of the section tests the different part of the functionality.

5.5.1 Unit tests

Unit tests are used for testing small units of code. For example, the function used for moving GPS location by distance and angle. The test input (location, distance, angle) is made and output is checked. Given output is compared with expected output and if it's

correct, that doesn't mean that function is correct. Better is to test it for more inputs and test more outputs. These tests are used for example for moving GPS location. Arguments given to the function are latitude, longitude, distance to move, bearing angle. The function is called with this arguments and the test is checking the output value. If the output is correct, than the function is tested for more inputs. Another example of unit testing is converting the GPS coordinates to the UTM coordinates. This is the principle of unit tests. Testing simple functions.

5.5.2 Android tests

Android tests are testing the core of the application. The tests are split into three phases: given, when, then. The testing object is given, when something happens, then test if the result is correct. For this system of testing, are used fake inputs and outputs. For input it's called *Dummy* and for output *Spy*. These objects are just faking the input data and looking if the output is the correct form. The application contains the presenter, which make the correct string form for display in the view. The goal is to test it with android tests. The dummy is created and gives the presenter the data (in this case, it's controller). And it is also checked if the output string is in the correct form. For this checking the spy is created (View). The controller gives the battery info, this battery info is formatted in presenter and sent to view to display it. This all can be covered by android tests. This procedure can be applied to almost all layers, expect the view, which has the UI tests described in 5.5.3.

5.5.3 UI tests

UI testing is very sophisticated and complex. Every user interaction and event cannot be covered by the tests. The view is a layer, which is the most frequently changed layer - tests have to be modified, with every change in the view layer and this makes UI tests unstable and fragile. Some of the cases are covered by UI tests, such as click events, labels appearance. UI tests are generated by Espresso [29] framework. Tests are generated in Java language and it is possible to change them or write them manually.

6 Communication technologies

Communication technologies are one of the most important parts of the research. The criteriums for communication, between smartphone and ROS, are the speed and the reliability. Two possible solutions will be described, implemented and compared in this work. Part of the research will be practical experiences in the simulator and real experiment. Communications such as Bluetooth and NFC were abandoned, because of their range of the signal. The only solution is to use Wi-Fi and related communication protocols and technologies.

6.1 SSH communication

SSH is probably the easiest solution, which brings a lot of pros and also cons. The brief introduction to the SSH technology is written below. SSH is not designed to fast communication, which is needed for commanding the drone, but SSH is reliable in the way of transferred bits and authentification. On the other hand, authentication is not necessary property and it is more harm than good.

6.1.1 Introduction to the communication technology

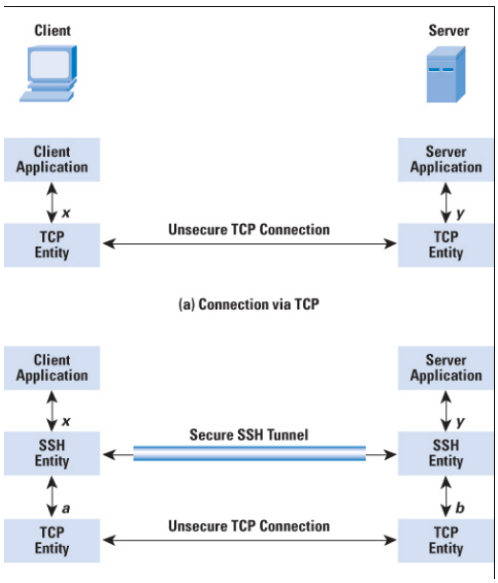


Figure 9: Explanation picture of SSH technology [11]

SSH is designated for a secured communication channel, using encrypted and TCP/IP protocols. SSH makes tunnel as can be seen on 9. SSH allows to send files and use shell

commands from the connected computer. That's the idea of this concept. Sending file is not used in this communication. SSH tunnel is one-way communication. Connected device can send the shell commands, but it cannot be reversed. The designed concept is to connect to the onboard PC using SSH and use prepared commands such as *goTo, goToRelative, goToAltitude*. These commands were designed by MRS group [22] on CTU. However, this concept of commanding is not designed to real-time communication, the authentication and the encrypting make it very slow. Second limitation - UAV executing the commands in the order in which they were received and the position is not overridden. This communication was tried in simulation, but for real usage it is unacceptable.

6.1.2 Implementation in the android application

Android doesn't have much libraries, which supports SSH. One of them is jonghough SSH library [4]. Jonghough library is in development stage and not all bugs are fixed, library can save the session (tunnel for communication), but cannot read the terminal output in controlling PC. Second library is JSch [18], one of the best SSH libraries for Android. For better understanding, the table of time spent to send one command, is shown below. Measured time of executing SSH command:

Number of the measurement	**Time [ms]**
1.	37
2.	35
3.	38
4.	169
5.	64
6.	36
7.	99
8.	2210
9.	2951
10.	34
11.	47
12.	45
13.	295
14.	117

Table 1: Lists of speed tests results of SSH communication

In measuring the mobile position doesn't change (change of the position may cause worse income signal - influence the results). The communication sometimes takes a long time that's unreliable and can make problems at the real experiment. Average spend time is 441.2 ms. Also depends on distance both of the devices from the router. Sometimes it

takes almost three seconds, which is unacceptable. Time spikes are caused by the authentification and the encrypting.

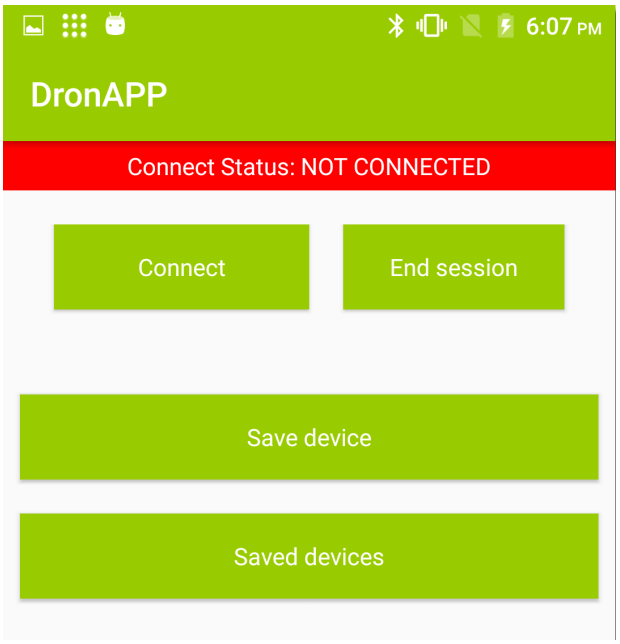


Figure 10: Screenshot of the settings layout

The screenshot of settings in the first application. Available functions are: connect to the onboard PC (it's necessary to know the username, hostname, password, and port), disconnect from the connected PC, save the credentials. The connection also takes some time, and it's indicated by the green/red bar under the toolbar (android tagged for the bar under the status bar, status bar - bar contains the time and system information). The indication bar is also on the main screen. Saved credentials are saved in application dictionary and they are saved until the application is installed.

6.1.3 Main problems and limitations of this technology

As can be seen in the introduction to 6.1, the issue is speed and the second issue is that the UAV finishs the location, which it received and the position is not overridden, that makes the move jerky, and UAV is badly controlled. This problem cannot be removed, because the command was not designed to control UAV in real-time. The spikes, recorded in 1 cannot be removed, too. The better approach is to use TCP/IP.

6.2 ROSJava communication

ROSJava is native Java implementation for ROS. Using subscribers, publishers and services to communicate with ROS master on UAV. Communication is based on publishers and subscribers. The publishers are publishing messages to some topic. Usually called /uavN/name_of_topic, where N is the number of the UAV. Subscribers are listening to this topics. All there subscribers and publishers are called nodes. This type of communication also used my college Stepan Kloucek and Petr Jezke in theirs Bachelor thesis [34] [35].

6.2.1 Introduction to the communication technology

Implementation includes two subscribers and two publishers. The subscriber is necessary for the battery information (voltage, percentage) and subscribing the position from the mobile phone. ROSJava doesn't support the custom message. The solution contains publisher with the custom message (in the format "x y z") and subscriber on the topic with this custom message. This message is parsed in the ROS subscriber and publish it to the desired position. In the real experiment, to the message was add additional information - calculated vector, to store this topic to the ROSBAG. Flowchart of the communication:

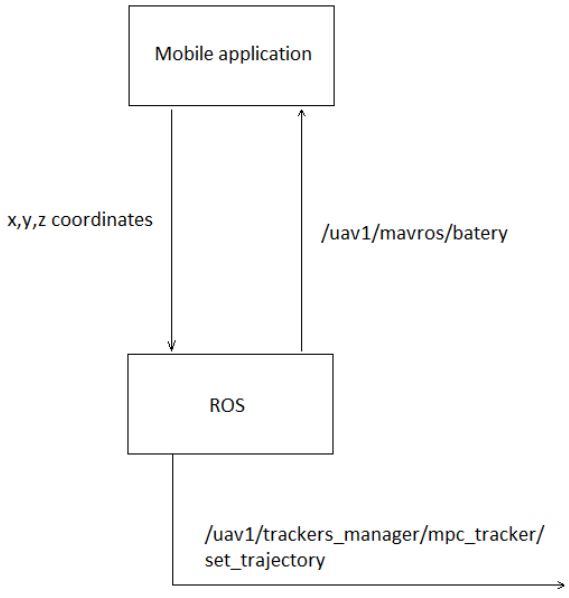


Figure 11: Flowchart of the used subscribers and publishers in ROS

11 shows the flow of the message, responsible for change the UAV position.

6.2.2 Implementation in the android application

ROSJava example code [27], used as template. Notice, that ROSJava is not longer supported in the development, but still some tutorial and guides can be found. The speed results to make comparation with the SSH technology 6.1:

Number of the measurement	**Time [ms]**
1.	13
2.	5
3.	5
4.	9
5.	8
6.	3
7.	28
8.	6
9.	1
10.	1
11.	14
12.	8
13.	1
14.	15

Table 2: Table of the speed tests results of ROSJava communication

Tested command - the goTo command. Publish the custom message to the MASTER ROS URI. There is a subscriber on topic /uav/set_position, which publishes the position to topic /uavN/trackers_manager/mpc_tracker/desired_position, where N is number of UAV. As can be seen, the commands take a small time to execute. Average spend time is 8.36 ms. And the spikes are not so obvious.

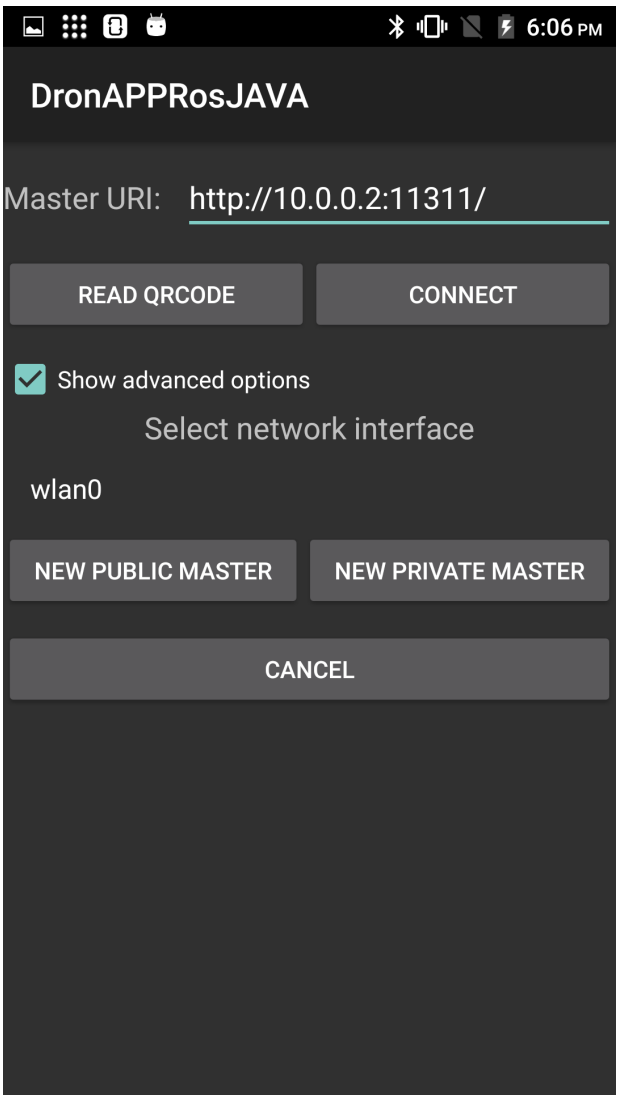


Figure 12: Shown screen of the connection screen of ROSJava application

12 shows the available functions of the connection using ROSJava. Available functions are: connect to the ROS Master by URL or read the QR code to connect. Advanced connection settings, allowing creating the new public master or private master. The port of the ROS master is 11311. The URI is the hostname of ROS Master on the same network. The screen is implemented by the library [27]. Can be changed by going to the source code, found in the important library.

6.2.3 Main problems and limitations of this technology

The problem of this communication is instability. Sometimes happens that the connection was refused and the only solution for this problem is to restart the application.

The restart of application involves reconnecting to the ROS MASTER. This problem is caused, by the low signal of Wi-Fi or unexpected bugs in the library. There is not another limitation or problem, occurred in the simulation or real experiment. The communication is fast and that is needed.

6.3 Compare of the ROSJava and SSH technology

The ROSJava is faster than SSH communication. SSH communication is, on the other hand, more stable. Communication using ROSjava takes the last sent position and fly there instead of SSH commands - fly to the sent location and queue the other commands. It is an inconvenience. The SSH is slow by 432.84 ms on average and it is suitable for settings trajectories, where the real-time control is not necessary. ROSJava is better for the real-time control.

7 Experiments and simulations

Experiments are separated into two types. The first type is testing on the simulator. On simulator are tested both of mentioned method and communications. The experiments were made with MRS group [22] in 16.4.-22.4.2018. The MRS provided the environment for simulation (Gazebo) with prepared features (such as UAV model) and also UAVs for real deployment. Experiments required one UAV with the hardware necessary only for flying. The goal of the experiments was to measure the accuracy and attempted to simulate the situation when the first one is controlled UAV and the second one is unauthorized UAV. Below are explained both types of testing, both communication and their behavior. The source code of the implemented application, used for experiments and simulations, can be found on bitbucket [6]. The source code for the ROS nodes can also be found in bitbucket repository [7].

7.1 Simulation

Simulation is used for verifying theories remembered in previous sections. Before real deployment, all methods and communications were tested on the simulator. Difference between real experiment and simulation is the procedure of calculating the vector. In the simulation, the mobile position is centered at point [0,0,0] and vector points UAV's position from this point. Coordinates system - x,y horizontal axis, z vertical axis. Tested function were commands (goTo, goToRelative, goToAltitude), console (terminal onboard PC), joysticks (using goToRelative command) and the edge method itself. SSH communication method is significantly slower than ROSJava. The times of speed can be seen in 6. Simulation is made in Gazebo simulator with one UAV with no special hardware used (such as special types of cameras of sensors are not necessary). The main purpose of the simulation is to test most of the implemented things to get awareness about how it will behave on real UAV.

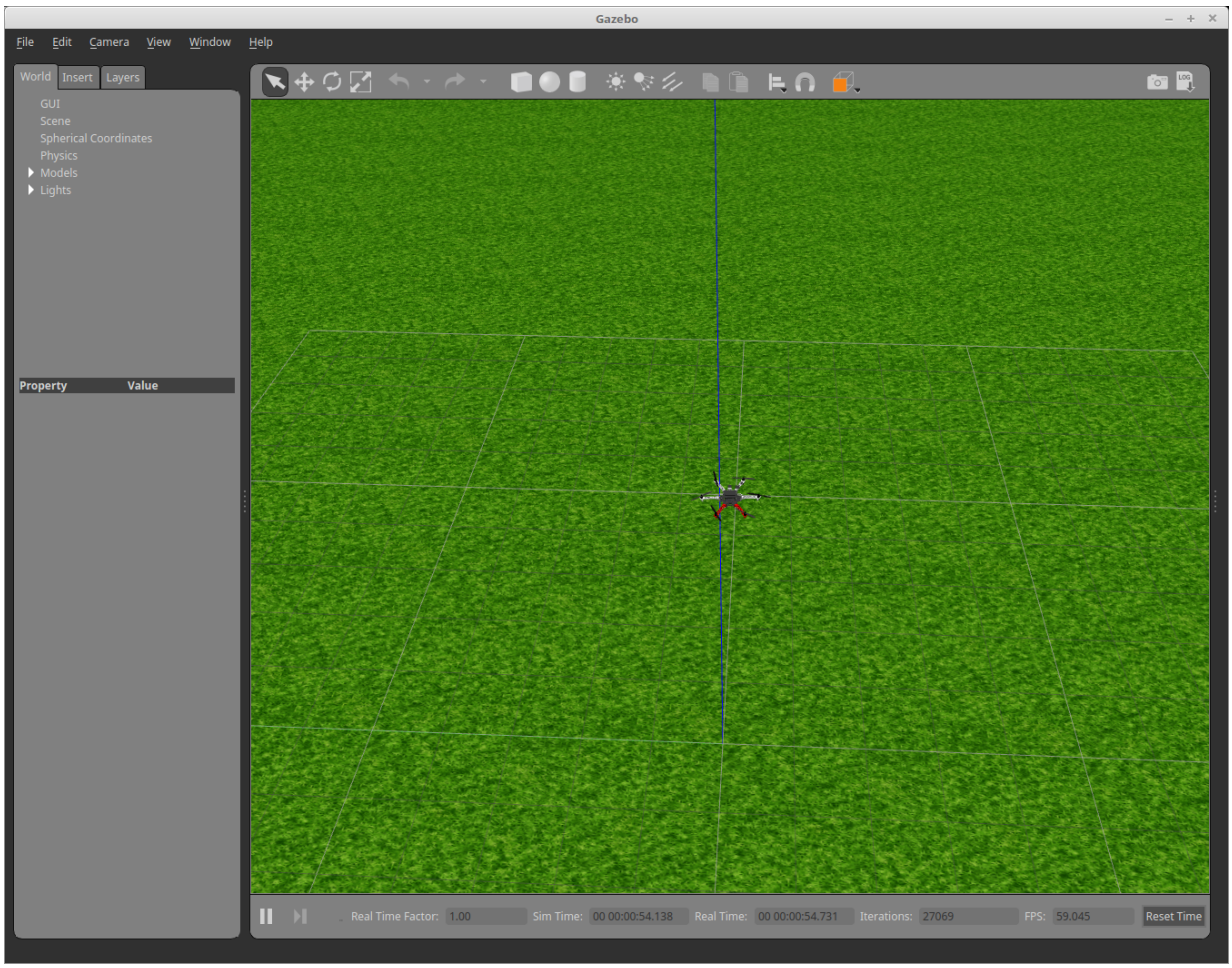


Figure 13: Gazebo 3D simulator - UI

13 shows user interface of Gazebo. The left bar contains hierarchical objects in the scene, bottom bar information about simulation (FPS, Simulation time, etc...) and top bar - utils for customizing view in the scene. The simple control allows customizing the simulation to make more complex simulations (for example - simulating swarm of UAVs and their behavior).

7.2 Experiments

The experiments are different than simulation. The vector cannot be calculated from user's position [0,0,0]. The mobile position has to be known by GPS integrated into the smartphone. The known position is moved by some distance, determined by the progress bar on the screen and rotated by the angle determined by the geomagnetic sensor to get points of the compass. These GPS coordinates are converted to UTM coordinates and deducted by offset for the camp place (camp, which was mentioned in the introduction,

was placed in **Cimelice u Orliku**). The converted coordinate is the final position send to UAV. Default zone for UAV is 33 U, easting, northing, attitude are calculated. It ought to be mentioned that for another zone than 33, the source code will not work. The better approach is to calculate this UTM in ROS. That it is possible to make the calculation of offset independent on the testing place. The altitude is gotten as *sin(elevation)*.

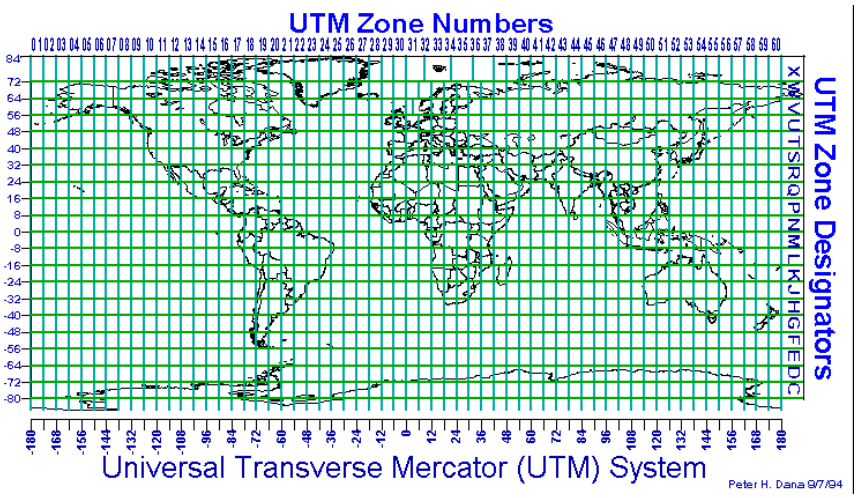


Figure 14: Zones used for UTM coordinate system [24]

14 shows the zones around the world. The Czech republic, exactly the place, where the real experiments were made (Cimelice u Orliku), is places in 33 zone.
In source codes can be found algoritmh for converting GPS to UTM. Pseudocode of the algoritmh:

Algorithm 3 Function for converting GPS location to UTM coordinates

```
1: function convertGPStoUTM
2:     Latitude
3:     Longitude
4:
5:     Letter = getLetter(latitude)
6:     Easting = getEasting(Latitude, Longitude)
7:     Northing = getNorthing(Latitude, Longitude)
8:     if Letter <' M' then
9:         Northing = Northing + 10000000
10:    end if
11:    Northing = round(Northing * 100) * 0.01
12:    return Easting, Northing
13: end function
```

The function converts GPS location in form latitude, longitude to UTM coordinates... Takes one argument - location and return UTM location. Function called *getLetter* is used for getting the letter by the latitude coordinate for UTM coordinates. Functions *getEasting*, *getNorthing* are implemented by complex equations. The zone is set to 33, which is the zone for the Czech Republic. The complete algorithm can be found on this website [1]. Experiments were separated into three parts. The first part was the simple test, communication, settings and other utilities. The second part was accuracy test, which is described in detail in 8. The goal was to measure the data necessary to later calculation of accuracy.

Figure 15: Snapshot of the accuracy experiment

The third and main experiment was catching another UAV. Two UAVs, first controlled by the mobile phone, second by radio broadcast. The goal was to catch the UAV by phone.

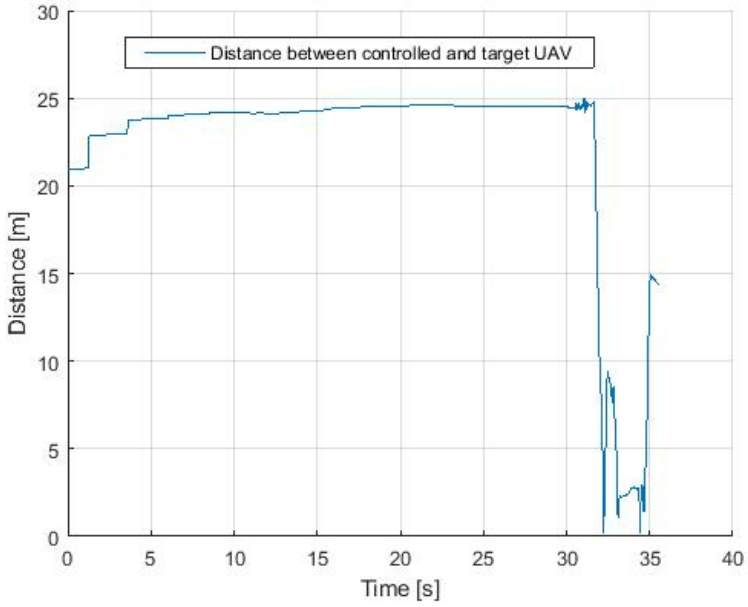


Figure 16: Graph of distance between target and controlled UAV

This graph shows the UAV coming to target UAV. The spike shows collision avoidance system, the UAV with the higher number (priority) flip the UAV with the lower number. It was possible to get close to the target to localize the target and eventually elimination. All these experiments can be seen on the final video, which can be found in DVD or MRS website [22].

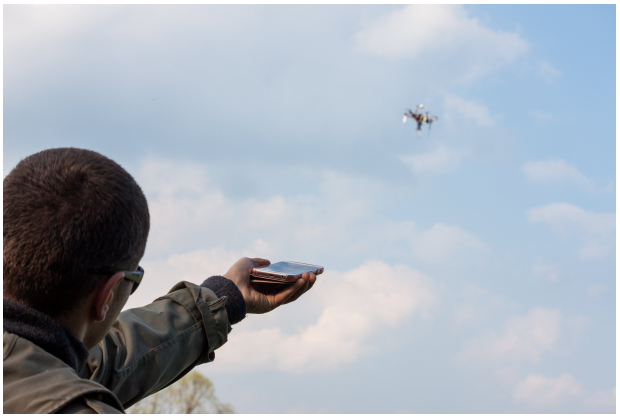


Figure 17: Snapshot of testing at the camp

7.3 Compare of both approaches - based on testing experiences

Both methods are implemented and tested in the simulator or on real UAV. Edge following was easier for implementation, better for debugging, because of Google map, used to display UAV location. There was easier to calculate the angles from the sensor values, but camera direction is more comfortable for control. For the camera's direction is hard to calculate the elevation and the points of the compass, caused by harder rotations. This cause the bugs and the camera method wasn't tested on real UAV. Angle detection in camera direction method is more complicated because of non-standard rotation of smartphone. Standart rotation of the smartphone is the rotation which is used in mobile edge method.

7.4 Results

To summarize the experiment and simulation, the simulation is different than the real experiment. The simulation doesn't contain the aspects necessary to be included for correct function, such as - inaccuracies in converting coordinates or inaccuracies in mobile sensors. Camera direction method is testable only at the simulator because it is not stable. The SSH communication is also not suitable for the real-time controlling. Mobile edge method was successful in testing and all recorded data are in the video. Measured data were used in section 8 to calculate accuracy.

8 Accuracy of the mobile direction

Three experiments were made to the determinate accuracy of mobile edge aiming method, for three different positions of phone and five different positions of the target UAV, but only two of them are usable for evaluation of results - in the third measurement, the connection between smartphone and UAV was lost in the middle of the flight. Calculation of accuracy contains the comparison of to the GPS location given by the mobile phone and the RTK (satellite navigation technique - used for the determinate position). RTK is more accurate than GPS position gotten from the mobile phone. The accuracy of the mobile GPS depends on the module used on the phone. Every mobile can have a different accuracy of the GPS. It equally applies to the other sesnsors: magnetic field, orientation sensor, and accelerometer. One general inaccuracy cannot be precisely determined for these sensors. Used mobile for the real experiment and determination of the accuracy of the GPS module was Samsung Galaxy J5. Matlab scripts used for the calculation described equations are stored in the bitbucket repository [5].

8.1 Measured data

First position of the phone - position gotten from the RTK: latitude - 49.475503, longitude - 14.017012 and the position from the mobile GPS: latitude - 49.47550, longitude - 14.01701.
Calculation of the GPS coordinates gotten from the RTK. Important to notice that the RTK coordinates are added by the offset, where the experiment were executed. More about experiments was described in 7.2.
RTK coordinates: northing = 0.0, easting = -29.5 and zone 33 U. The experiment place's offset have to by added to this coordinates: *northing* = 428789.2 + 0.0 = 428789.2, *easting* = 5480811.2 -29.5 = 5480772.7. This final coordinates was converted to the GPS using geoplaner [13] and the results are lat = 49.475503, lng = 14.017012.
The distance between these two GPS locations is 0.3635 m. The GPS module included in the mobile phones gives the location 0.3635 m away from the exact position.

Target UAV position [Lat, Lng]	**Controlled UAV position [Lat, Lng]**
14.0165130, 49.4761779	14.0170524, 49.4756287
14.0170812, 49.4761239	14.0171873, 49.4756646
14.0173715, 49.4760696	14.0172651, 49.4756654
14.0175096, 49.4760162	14.0172716, 49.4756644
14.0175516, 49.4759800	14.0172715, 49.4756649

Table 3: Measured data

From the smartphone position, target UAV and controlled UAV the assumed location can be calculated, From the assumed location and controlled UAV location, the distance

can be calculated. Assumed position means, the position in the direction of the phone aiming. For this calculation is used Matlab script, which can be found on the bitbucket repository [5]. This script is using algorithm from *MathWorks* [21]. The general process of the calculating assumed position:

$$v = \frac{d}{||u||}u + s \tag{4}$$

Where v is assummend point, d is the distance of vector from the user position to the controlled UAV position, u vector from the user position to the target UAV position and s is user position.

Assumed position [Lat, Lng]	**Distance [m]**
[14.0169335, 49.4756091]	13.4
[14.0170384, 49.4757399]	18.4
[14.0171731, 49.4757569]	14.2
[14.0172247, 49.4757224]	8.1
[14.0172411, 49.4757055]	5.5

Table 4: Measured data

The distance is between *assumed position* and *target UAV position*.

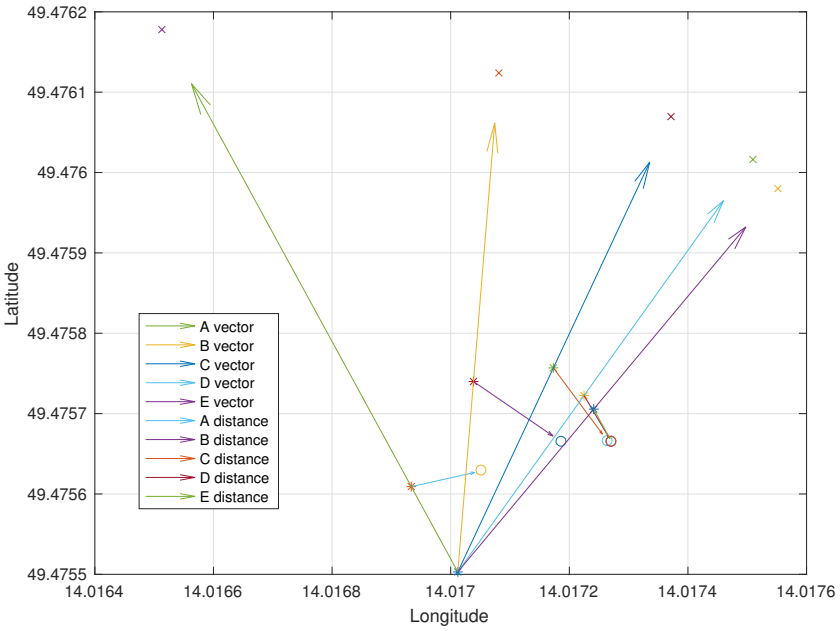


Figure 18: 2D graph of the target and controlled UAV and the mobile position, first flight

18 shows the vectors to the target UAVs and the distances between the assumed position and controlled UAV position. The vector from symbol ***** and **x** show the direction between mobile direction and target UAV. It makes sense of the mobile aiming. The vector between **.** and **o** shows the distance between the assumed position and the controlled UAV position. A-E vectors and A-E distances. A distance belongs to A vector, etc... The mobile phone is in the static position, calculated at the beginning of **First position of the phone**.
In average the distance between the assumed and the required position is 11.88 m.

Second position of the phone - position gotten from the RTK: latitude - 49.475482, longitude - 14.01713 and the position from the mobile GPS: latitude - 49.475479, longitude - 14.01714.
For calculating the exact position given from the RTK, the same approach as in **First position of the phone** was used.
The distance between these two GPS locations is 0.7958 m. The GPS module included in the mobile phones gives the location 0.7958 m away from the exact position.

Target UAV position [Lat, Lng]	**Controlled UAV position [Lat, Lng]**
14.0165148, 49.4761787	14.0169267, 49.4756741
14.0170845, 49.4761292	14.0170849, 49.4757147
14.0173759, 49.4760776	14.0171983, 49.4756906
14.0175178, 49.4760207	14.0172495, 49.4756725
14.0175677, 49.4759815	14.0172763, 49.4756615

Table 5: Measured data

From the smartphone position, target UAV and controlled UAV the assumed location can be calculated, From the assumed location and the controlled UAV location, the distance can be calculated. Assumed position means, the position in the direction of the phone aiming. For this calculation is used Matlab script, which can be found on the bitbucket repository [5]. This script is using algorithm from *MathWorks* [21]. The general process of the calculating assumed position:

$$v = \frac{d}{||u||}u + s \tag{5}$$

Where v is assummend point, d is the distance of vector from the user position to the controlled UAV position, u vector from the user position to the target UAV position and s is user position.
This approach is the same as in the calculation of **First position of the phone**.

Assumed position [Lat, Lng]	Distance [m]
[14.0168974, 49.4756563]	3.8
[14.0170379, 49.4757252]	5.3
[14.0171540, 49.4757255]	6.2
[14.0172165, 49.4757105]	5.5
[14.0172460, 49.4757028]	5.6

Table 6: Measured data

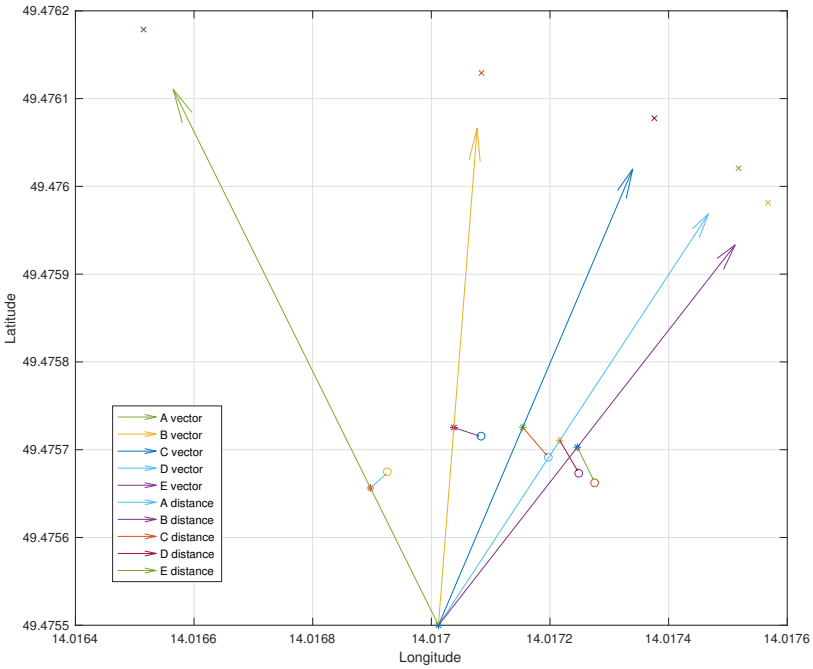


Figure 19: 2D graph of target and controlled UAV and the mobile position, second flight

19 shows the vectors to target UAVs and the distances between the assumed position and the controlled UAV position. The vector from symbol * and **x** show the direction between the mobile direction and the target UAV. It makes sense of the mobile aiming. The vector between **.** and **o** shows the distance between the assumed position and the controlled UAV position. A-E vectors and A-E distances. A distance belongs to A vector, etc... The mobile phone is in the static position, calculated at the beginning of **Second position of the phone**.
In average the distance between assumed and the required position is 5.28 m.
There is not included attitude inaccuracy, because there is no way to get the attitude of the smartphone without the third party API. Altitude from GPS is very inaccurate, and it is unusable for controlling. The way to get the accurate altitude is from the barometer, which is not included in old phones.

8.2 Results

As can be seen above, the method is not precise. Irregularities are caused by the GPS module. First GPS difference is 0.3635 m and second 0.7958 m. That's enough to make mistakes in final positions. Other reasons for inaccuracy position are sensors. Sensors accuracy is different for every phone. Some phones are using cheap sensors and some expensive phone sensors, usage of the more expensive phone can lead to more precise positions and precise determination of the final position. The diversity of mobile GPS modules is the aspect of the difference between the expected and the calculated location. The first test flight result of the distance between assumed and controlled UAV position is in average 11.88 m and the second 5.28 m. This inaccuracy is also caused by the human factor (incorrectly targeting the smartphone).

9 Conclusion

There is a discussion of the problem of controlling UAV using mobile phone tilt in this work. Description of the used hardware, technologies and libraries used to complete this research.
Work on this thesis is part of the system used for the localization and the elimination unauthorized UAV in the forbidden locations. This system is not part of this thesis, but research is completely ready to implement to it.
At first, the SSH communication was designed and implemented and found that it is not the best approach. ROSJava solution, was the second communication designed and implemented, the pron and cons are explained in 6.2. ROSJava communication took a lot of time to study about it, because it not supported for a long time. After a lot of studies the examples were found and used as a template.
As can be seen in 6.2, the main advantage of ROSJava communication is speed. The speed is the decisive limit of usage. The low speed of SSH communication makes it unusable in practice. ROSJava communication also contains some inconveniences such as frequently lots of the connections.
All these designed and implemented theories were tried on camp on real UAV with MRS group. More about the the camp in 7.2. For the real test was used only mobile edge method. Camera direction method has a lot of bugs and sometimes the UAV was send to the undefined location and used communication for the real experiment was ROSJava. The results were successful, and the necessary data used for calculation accuracy were measured. More about the accuracy in 8. The section contains the graphs and all explanations of calculations.
The main experiment was try to catch another helicopter. Implemented system will be connected to the system to localize and eliminate unauthorized UAV in the forbidden location. The experiment was successful, and the controlled UAV was able to get to close distance to the target UAV, fortunately the UAVs contain collision avoidance and the UAVs did not strike.
To summarize, two communications were implemented for two methods of aiming. One is usable for the real usage and one have a few problems. Both methods and communication was tested and made the results. The results are, that the GPS module on tested phone (Samsung Galaxy J5) is not accurate, results can be found in 8.2. To fix this, the better GPS module is needed. The data are measured only on one smartphone with one GPS module. Unfortunately, no other smartphones were not included in this research.
For camera direction, was made a the research and it was found that it is not usable, yet. There have to be fixed random generation nonsense values. Sometimes the sensors return the value different by 300 degrees and it can cause the accident of the UAV. The random generation nonsense values can be caused by the mobile sensors, but the solution is to make filter to filter these random values. Simple solution of the filter was implemented, but it needs more complex algorithm.
In both solution the research had to deal with random values from GPS. This problem was

fixed by saving the last position and calculating the difference between the current and the last position. If the distance is larger than 5 meters, the GPS location is not usable and is discarded. This filter nonsense values from the GPS module. Nonsense values from the GPS module are caused for example, when phone is under the the roof, the GPS doesn't have enough signal to determine correct location.
Without precise sensors is hard to determined the precise location. The low hardware is the biggest limitation to make this system best of all. Unfortunately the tested Android smartphone has a low hardware and to find the better one was not possible, but the experiment with a low hardware phone was successful and this research found all the resources of defects and eliminate all possible.

References

[1] Algoritmh for converting the gps location to the utm coordinates. https://stackoverflow.com/questions/176137/java-convert-lat-lon-to-utm. Accessed: 2018-02-10.

[2] Android library for injecting the views to the activities. http://jakewharton.github.io/butterknife/. Accessed: 2017-10-09.

[3] Android library for the virtual joysticks. https://github.com/controlwear/virtual-joystick-android. Accessed: 2017-10-11.

[4] Android library used for the connection via ssh. https://github.com/jonghough/AndroidSSH. Accessed: 2017-10-02.

[5] Bitbucket repository containing the source code for matlab scripts, used for the accuracy calculation and the rosbag parse. https://bitbucket.org/filipbursik5/commanddron-matlab/src. Accessed: 2018-05-22.

[6] Bitbucket repository containing the source code for the android application. https://bitbucket.org/filipbursik5/commanddron-android/src. Accessed: 2018-05-22.

[7] Bitbucket repository containing the source code for the ros nodes. https://mrs.felk.cvut.cz/gitlab/bursifil/mobilecontroll. Accessed: 2018-05-22.

[8] Desription of the android control on the turtle bot example. http://wiki.ros.org/turtlebot/Tutorials/AndroidControl. Accessed: 2018-05-24.

[9] Dji phantom specification. https://www.dji.com/phantom. Accessed: 2018-05-20.

[10] Explanation of the calculating elevating angle on the android phone. https://stackoverflow.com/questions/5284493/get-angle-of-elevation. Accessed: 2017-11-12.

[11] Explanation of the ssh technology. https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-46/124-ssh.html. Accessed: 2018-02-02.

[12] Forum, where can be found the resource for moving the gps location. https://stackoverflow.com/questions/10119479/calculating-lat-and-long-from-bearing-and-distance. Accessed: 2018-01-09.

[13] Geoplaner used for the converting the utm coordinates to the gps coordinates. https://www.geoplaner.com/. Accessed: 2018-05-10.

[14] Github library for the dependency injection. https://github.com/google/dagger. Accessed: 2017-10-09.

[15] Google maps api. https://developers.google.com/maps/documentation/android-sdk/intro. Accessed: 2017-10-09.

[16] Informations about the meka robotics, no longer exist industry, using the ros. https://en.wikipedia.org/wiki/Meka_Robotics. Accessed: 2018-05-21.

[17] Informations about the ros. https://en.wikipedia.org/wiki/Robot_Operating_System. Accessed: 2017-09-02.

[18] Library for the ssh communication. http://www.jcraft.com/jsch/. Accessed: 2017-10-09.

[19] Link to download ros control, android application. https://play.google.com/store/apps/details?id=com.robotca.ControlApp&hl=en_US. Accessed: 2018-05-23.

[20] Link to download the android application for turtlebot. https://play.google.com/store/apps/details?id=org.ros.android.controllerSample. Accessed: 2018-05-23.

[21] Matlab script used for calculating the distance between two gps locations. https://www.mathworks.com/matlabcentral/fileexchange/38812-latlon-distance. Accessed: 2018-04-05.

[22] Multi-robot systems group. http://mrs.felk.cvut.cz/. Accessed: 2017-09-23.

[23] Multi-robot systems group list of thesis and publications. http://mrs.felk.cvut.cz/people/martin-saska. Accessed: 2017-08-02.

[24] Picture showing the zones for the utm coordinate system around the world. http://www.jaworski.ca/utmzones.htm. Accessed: 2017-10-09.

[25] Ros catkin. http://wiki.ros.org/catkin. Accessed: 2017-09-02.

[26] Ros wiki. http://wiki.ros.org. Accessed: 2018-01-20.

[27] Rosjava. https://github.com/rosjava/rosjava_core. Accessed: 2017-10-09.

[28] Source code of the turtlebot application. https://github.com/turtlebot/turtlebot_android. Accessed: 2018-05-24.

[29] Ui testing android library. https://developer.android.com/training/testing/espresso/. Accessed: 2017-12-01.

[30] Video showing funcionality of the ros control application. https://www.youtube.com/watch?v=1MdV2NdT2w4. Accessed: 2018-05-24.

[31] Website of the robotics equipment corporation using the robot operating system (ros). `http://servicerobotics.eu/`. Accessed: 2018-05-21.

[32] Website of the shadow robot industry using using the robot operating system (ros). `https://www.shadowrobot.com/`. Accessed: 2018-05-21.

[33] Filip Bulander. Interface ios for control of an unmanned helicopter in ros. 2018.

[34] Petr Jezke. Autonomous helicopter control by a mobile phone with android for precision agriculture. 2018.

[35] Stepan Kloucek. Android application for control of an unmanned helicopter carrying a bird repeller. 2018.

[36] Robert C. Martin. *Clean Code - a Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009.

[37] Simon Puligny. Ros interface and urdf parser for webots. page 48, 2014.

[38] Vojtech Robotka. Interaktivni rozhrani pro vzdaleneho robota pro android. page 49, 2014.

[39] Lazaros Stamatiadis. Mobile application for controlling the embedded system. page 50, 2018.

Appendix A CD Content

In Table 7 are listed names of all root directories on CD.

Directory name	**Description**
thesis	the thesis in pdf format
thesis\sources	latex source codes
thesis\sources\chapters	latex source codes for chapters
thesis\sources\fig	pictures used for figs
video_experiments	phantom_videos
app_sources	source code of the application
app	apks for installation
matlab_sources	Matlab scripts
ros_source	source code of the ros nodes

Table 7: CD Content

Appendix B List of abbreviations

In Table 8 are listed abbreviations used in this thesis.

Abbreviation	Meaning
API	application programming interface
UI	user interface
ROS	robot operating system
SSH	secure shell
UAV	unmanned aerial vehicle
PC	personal computer
OS	operating system
MVP	model view presenter
GPS	Global Position System
TCP/IP	transmission control protocol/internet protocol
RTK	real-time kinematic
URI	uniform resource identifier
CTU	Czech Technical University in Prague
BUT	Brno University of Technology
MRS	Multi-Robot Systems group at FEE CTU
FEE	Faculty of Electrical Engineering
UTM	Universal Transverse Mercator

Table 8: Lists of abbreviations