Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS

Robert Husák

Quadrotor 3D Intuitive Flying

Department of Software and Computer Science Education

Supervisor of the bachelor thesis:	Mgr. Tomáš Plch
Study programme:	Computer Science
Specialization:	Programming

Prague 2014

I would like to thank my supervisor, Mgr. Tomáš Plch for his guidance and advice. I am also grateful for the support of other members of Autonomous Aerial Drone research group held in the university.

Název práce: 3D intuitivní řízení quadrotoru

Autor: Robert Husák

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Tomáš Plch

Abstrakt: R-UAV (angl. Rotor-based Unmanned Aerial Vehicle - rotorové bezpilotní letouny) je robotická platforma schopná vertikálního vzletu a přistání. Obvyklý způsob jejich ovládání je pomocí systému "dvou páček", který je častý o modelů lodí a letadel řízených přes rádiové spojení. Ukazuje se však, že v oblasti R-UAV je tato metoda ovládání pro uživatele obtížná na pochopení a osvojení. Proto jsme vyzkoušeli jiná ovládací zařízení: smartphone, gamepad, joystick, 3DConnexion SpaceNavigator a Novint Falcon. 19 dobrovolníků je používalo k plnění dvou jednoduchých navigačních úkolů pomocí AR.Drone, malého R-UAV od firmy Parrot. Na základě zkušeností pilotů s řízením R-UAV jsme je rozdělili do dvou skupin. U každého zařízení jsme měřili čas potřebný k plnění jednotlivých úkolů, počet nechtěných kolizí s překážkami a také míru spokojenosti pilotů s tímto zařízením. Na základě výsledků tohoto experimentu se ukázalo, že joystick a smartphone byly nejméně intuitivní a efektivní pro všechny piloty, zejména pro nezkušené. Gamepad se zdá být použitelný zejména pro zkušené piloty, nezkušení měli problém s rozdělením ovládání mezi dvě ruce, jak jsme očekávali. Jako nejlepší zařízení pro nezkušené piloty se ukázaly být 3DConnexion SpaceNavigator a Novint Falcon.

Klíčová slova: quadrotor, R-UAV, UAV, ovládání, vzdušný, drone, 3D pilotování, 3D myš, Novint Falcon, smartphone, gamepad, joystick

Title: Quadrotor 3D Intuitive Flying

Author: Robert Husák

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Tomáš Plch

Abstract: The Rotor-based Unmanned Aerial Vehicle (R-UAV) is a robotic aerial platform capable of Vertical Take-Off and Landing. It is commonly controlled by a "two stick" interface common to radio controlled models of ships and airplanes. However, this method of control proves to be difficult to learn and master in the domain of R-UAV. Therefore, we examined other control devices: smartphone, gamepad, joystick, 3DConnection SpaceNavigator and Novint Falcon. They were used by 19 volunteers to perform two simple navigation scenarios with AR.Drone, a small R-UAV from Parrot. We separated the pilots into two groups, based on their experience with the R-UAV remote control. We measured the time needed for each device to fulfill every task, including the number of unwanted collisions with obstacles, as well as the pilot's satisfaction with it. According to the results of the experiment, joystick and smartphone proved to be less intuitive and effective among all pilots, especially the inexperienced. Gamepad seemed to be most usable for experienced pilots, although the inexperienced had problems distributing control between two hands, as we expected. 3DConnexion, SpaceNavigator and Novint Falcon were rated the best devices for inexperienced users.

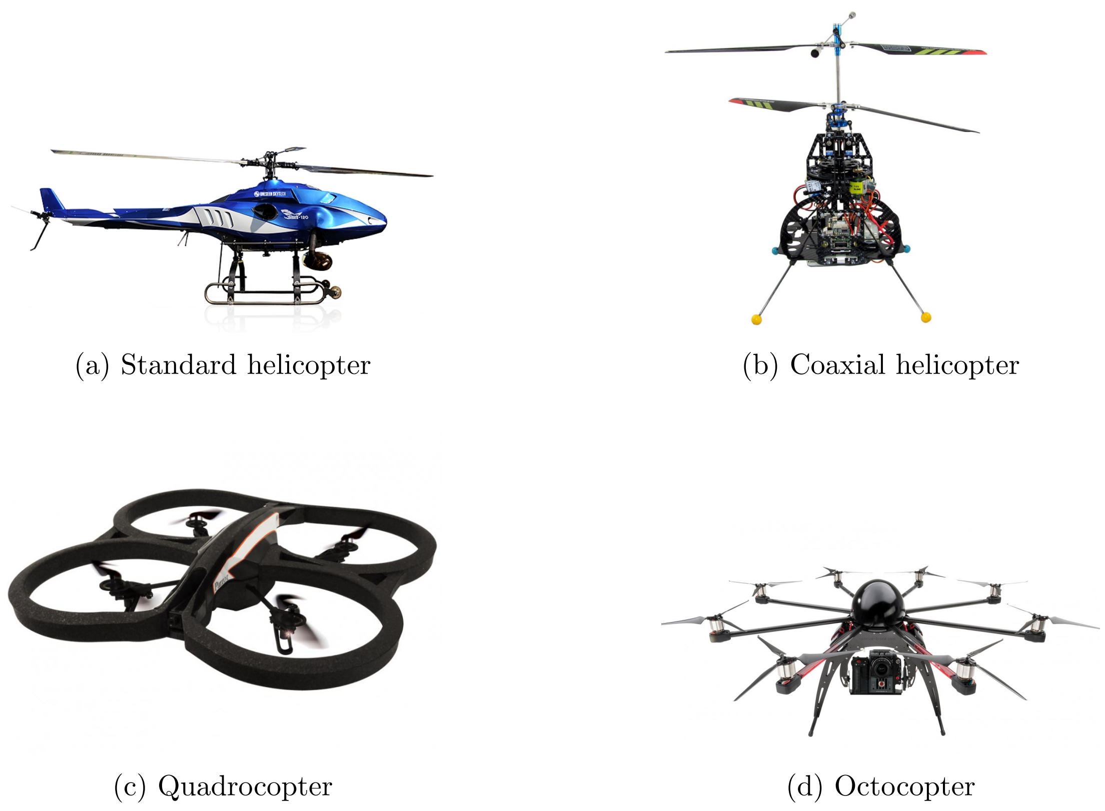
Keywords: quadrotor, R-UAV, UAV, control, aerial, drone, 3D piloting, 3D mouse, Novint Falcon, smartphone, gamepad, joystick

Contents

Introduction

The Unmanned Aerial Vehicle (UAV) platform has recently become a frequently discussed topic, mostly in connection with their use by the US Army. The term *unmanned* refers to the fact that the vehicle doesn't carry a pilot. Instead, its control is distributed between its on-board computer and a Ground Control Station. Under this definition we might imagine either a building with sophisticated pilot user interfaces or a simple, light-weight device carried by the pilot himself. In many scenarios, UAVs prove to have significant advantages over manned aircraft, the most important of these being safety of the crew and economics of the operation. [1] The most common types of UAV airframe are fixed-wing and rotary-wing.

UAVs with fixed-wing configurations resemble ordinary airplanes. They fly by jet engines or propellers and are mostly useful for long-range operations in open space. However, unable to hover in a constant position, they are not useful for close-range operations taking place at low altitude and/or over complicated terrain. Moreover, as they are only capable of Horizontal Take-Off and Landing (HTOL), their use in places without appropriate 'runways' is complicated.


(a) Standard helicopter (b) Coaxial helicopter

(c) Quadrocopter (d) Octocopter

Figure 1: The most common R-UAV rotor configurations

In this thesis, we will focus at the second type of configuration: rotary-wing. This platform is commonly referred to as Rotorcraft-based UAV (R-UAV and flies by the lift of the rotating wings (rotors). Due to physical laws, each rotor is both a source of lift and torque affecting the vehicle. Therefore, the configuration of the rotors must be very carefully designed. R-UAV piloting is done through changes of the rotors' lift, as some rotors are able to tilt their individual blades. Figure 1 shows several types of rotor configurations. The best known is the standard helicopter (1a), but coaxial helicopters (1b), quadrocopters (1c), hexacopters and octocopters (1d) are becoming more and more popular.

R-UAVs are used in both military and civilian applications. The primary military applications are surveillance, reconnaissance, combat and testing of new weapon systems. Non-military applications, apart from surveillance, include industrial systems inspection, border patrol, rescue missions, fire prevention, terrain mapping and agriculture monitoring. [11] They are also used by film makers and photographers and many universities use them for education, research and the development of new technologies and algorithms. We should also not forget their pure entertainment use, either by hobbyists building them by themselves or customers buying from a manufacturer.

They have numerous advantages over the fixed-wing alternative. Regarding their use in complicated terrain, the most important are capability of Vertical Take-Off and Landing (VTOL), as well as hovering in constant position. They also offer better stabilization and wider variety of manoeuver.

As for disadvantages, R-UAVs are slower and less fuel-effective for long-range flight and as the rotors can be very complex, they are more likely to malfunction. Apart from these technical issues, there is also a problem regarding the user interface for remote pilots. Because many R-UAVs are radio controlled (RC), they are designed to work with standard RC transmitters. Their main user-interface elements are two sticks controlled by the thumbs, as seen on Figure 2a. Both sticks can be rotated horizontally and vertically, giving four axes of control. Usually the right stick is mapped to vertical movement and rotation around vertical axes, while the left stick controls horizontal rotation axes. Beginners are often confused by this mechanism, because it is not intuitive. Moreover, the learning curve for this type of control is steep and, until it is mastered, vehicle crashes are the norm. [6]

The goal of this thesis is to solve the problems of R-UAV remote human control. In order to do that, we will select several other devices that are commonly available on the market and examine how intuitive they are for users. All such devices are shown in Figure 2.

With the growing popularity of smartphones it is small wonder they are currently used as the primary control-device for several hobby R-UAVs. [8] [2] A smartphone can use its multi-touch color display to show the R-UAV video and accept commands from the user. Some smartphones may utilize a gyroscope or compass to enhance the user experience. Those are reasons why we included it in our list.

The second device examined is a *gamepad*. Although its user interface is rather similar to the one the RC transmitter provides, it is very popular among video-game players. The latter applies to a *joystick* as well. Moreover, it is used mainly for flight simulators and its variations appear in cockpits of helicopters.

Next devices are two that represent new controllers *designed* to work in 3D, which have recently become known to computer-game players and 3D graphic designers. 3DConnexion SpaceNavigator, often called the *3D mouse*, is a button that fits into the palm and can be rotated and translated into all three degrees.

Novint Falcon is an advanced 3D positioning device allowing force feedback

(a) RC Transmitter

(b) Smartphone

(c) Gamepad

(d) Joystick

(e) 3D mouse

(f) Novint Falcon

Figure 2: R-UAV possible remote control devices

on all those three axes. In other words, it has the ability to exert a force against the user. Although it is relatively cumbersome due to its size and need of a power supply, we expect it to provide an interesting user experience.

Organization

The thesis consists of three parts: Analysis, Implementation and Evaluation. In Analysis, we summarize existing research in this area, inspect R-UAV platforms and provide information about particular control devices. Those topics are contained in Chapter 1, 2 and 3 respectively. Chapter 4 summarizes and specifies the problem we intend to solve.

The software part of the problem solution is described in Implementation, which is divided into five chapters. Chapter 5 summarizes the proposed solution. The remaining chapters describe the implementation of the particular parts of the application: Glue component system, AR.Drone component, control devices components and the user interface.

Evaluation explains how the software implemented in the previous parts is used and consists of three chapters. In Chapter 10, all the evaluation scenarios and criteria are described. Results measured are listed and commented upon in Chapter 11. The last chapter contains the interpretation of results and plans for the future.

The content of the enclosed CD is described in the Attachments section.

Part I

Analysis

1. Related Work

As the use of the R-UAV platform grew rapidly in the last two decades, it is no wonder that it is a popular subject for research among universities and scientists all around the world. For every research group interested in this topic it is common to build their own R-UAV, usually an electric quadrocopter. As a first step, they construct the hardware portion of the vehicle, make decisions regarding the batteries used, along with motors, propellers and other components. Then they implement its basic functionality, such as stabilization, hovering in a constant position, taking-off and landing. From this point on, the vehicle can serve as a platform for testing of many varied algorithms, including obstacle avoidance, path finding, flying in a group, etc.

All the steps mentioned are not trivial and require a deep understanding of the platform particularities. As a starting point for this purpose, we recommend the book *Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy* [11]. Among others, it provides hints for R-UAV construction and implementing algorithms for control. Moreover, it summarizes the current possibilities for UAVs and identifies challenges for the future.

The authors see implementing a robust way to control and cooperate swarms of UAVs as their most important task. Other aims include improvements in flight time, sensors and autonomous navigation algorithms. According to the authors, UAVs must be autonomous enough to allow their human operators to assign them high level-commands. An example of such a command is: "Fly to a destination provided by GPS coordinates, unobserved and perform reconnaissance over a 10 kilometer perimeter area."

It seems that the book provides a true image of current research, because there is an abundance of projects in the aforementioned areas. On the other hand, it is hard to find a project concerned with user satisfaction with the low-level control of UAVs, particularly R-UAVs. An exception is the article *Development of RC helicopter control skill study support system in consideration of user interface*, by Junichi Kunieda and Yukinobu Hoshino. [6] They understand how difficult the RC helicopter control is for beginners and propose ways to simplify the learning process. Among others they used a head mounted display to transfer images from the on-board camera and sensors, which literally provided the user a better insight into the flight of the helicopter.

The approach of this thesis is different. We believe that the whole 'two stick' mechanism is an inappropriate way to fly R-UAVs and we want to try other control devices more suitable for beginners. After we select the devices, several volunteers will use them to perform certain navigation scenarios. From their feedback we will be able to decide which devices seem to be the most convenient.

2. R-UAV Characteristics

As the R-UAV platform is a broad topic, we will concentrate only on its fundamentals. In the beginning, we describe the particular mechanical parts responsible for flying and the configurations of assembling them together to build a functional unit. The example of the quadrocopter configuration and methods of performing manoeuvers in the air will be demonstrated. The last section of this chapter provides an overview of an on-board controller, which is the device responsible for fulfilling navigation commands.

2.1 Parts

The most important mechanical parts of the R-UAV necessary for flying are a mainframe, engines and rotors. The mainframe serves as a skeleton, upon which all the other parts of the R-UAV are attached. It is expected to endure vibrations, centrifugal forces, hard landings and other inconveniences. Therefore, its material is expected to be strong and durable; on the other hand, it must be light enough to enable the vehicle to fly. For some types of engines, the mainframe material must also withstand a substantial level of heat.

The most common metal used for this purpose is aluminum, as it is inexpensive, heat resistant and strong. It is too heavy to be used on small models, but for larger ones is suitable. On the other hand, when exposed to very high frequency vibrations it may develop stress cracks and is prone to bend unnoticeably in cases of hard landings or crashes. Apart from aluminum, special alloys also may be used, especially for military purposes.

Plastic does not suffer from the weaknesses of aluminum, thanks to its flexibility. Nevertheless, it is weaker and not resistant to heat, which limits its use to small R-UAVs powered by certain “cold” engines such as electric motors.

Probably the best available material is carbon-fiber. It is stronger and lighter than all those mentioned and also withstands heat and vibration. As such, it is also the most expensive of the materials mentioned. As a cheaper substitute, fiberglass can be used, which is about twice as weak. [9]

Another important aspect of every R-UAV is the particular energy it uses to spin the rotors. Especially for large vehicles, engines powered by bio-chemical fuels, such as petroleum fuels or diesel, are common. Those engines are divided into two categories: piston engines and gas-turbine engines. Piston engines resemble those known in cars, as they use the combustion of the fuel to move the pistons. The pistons are arranged in such a way as to allow them to spin an attached shaft responsible for turning the rotor. An analysis of particular types of piston engines can be found in [1, p. 102].

Gas-turbine engines use a different mechanism, based on the continuous streaming of air. Air is compressed, mixed with the fuel and ignited. While in the case of airplanes the resulting energy is directly transformed into the drag of the vehicle, in rotorcraft it is used to rotate the shaft. This type of engine is often called a turboshaft.

Electric motors are completely different types of engines. Compared to engines based on bio-chemical fuels, they have the advantage of being the quietest and with a smaller thermal signature. However, when considering the proportion of weight and capacity of batteries, they are usable only for small R-UAVs with flight times measured in tens of minutes. [1]

The last part of the R-UAV to be examined is the rotors. A rotor consists of a central mast to which particular blades are attached. The shape of the blades is similar to that of an airplane wing. Therefore, when they rotate, they draw the air above them down. As a reaction, the vehicle is pushed upwards by the opposite force, called thrust. Apart from the force, the rotor is also the cause of torque in a direction opposite to its rotation. Further explanations of the physics laws behind those mechanisms can be found in [3].

Some rotors are able to variably change the pitch of their blades, to either increase or decrease the thrust they cause. For this operation a component is used called swashplate. It is a circular plate surrounding the base of the mast, around which linkages rotate to particular rotor blades parallel to the mast. When the swashplate is elevated or lowered, the pitch of all blades is altered, which is called 'collective blade control.' It is also possible to tilt the swashplate at certain angles. In that case, the pitch of the blades (and consequently the thrust) will differ among the particular angles of rotation, causing the vector of the resultant thrust to tilt. This mechanism is called 'cyclic blade control.'

When the blades of the rotor cannot change their pitch, the only method of adjusting the rotor thrust is to alter the speed of its rotation. Fixed-pitch rotors are sometimes referred to as propellers, although strictly speaking they are two distinct concepts. Propeller blades are generally shorter and made from hard materials, whereas rotor blades are expected to bend when not rotating.

2.2 Configurations

In this section we mention the most common configurations of the parts used in the R-UAV platform, as shown in Figure 2.1. Probably the best known is the helicopter (2.1a). At the top of its main body a main rotor is attached, capable of both collective and cyclic blade control. In order to suppress the torque of the main rotor, a tail-rotor with collective blade control capability is attached. It is responsible for rotation around the vertical axis, whereas the main rotor causes all the other types of movement. The tail-rotor is probably the greatest weakness of this configuration, because it is vulnerable and consumes energy inefficiently. Furthermore, it is a source of asymmetry, complicating control of the helicopter.

Those disadvantages are eliminated in the case of a coaxial rotor configuration (2.1b). The tail-rotor is missing and repaced by a second main rotor, stacked on the first and rotating in an opposite direction. The only downside of this configuration is the increased mechanical complexity of the rotors.

Quadrocopters are a solution without rotors, using variable-pitch blades, sometimes called quad-rotors (2.1c). Their mainframe is shaped as a cross with four propellers attached to its tips, while each pair of opposing propellers rotate the opposite direction in order to nullify the torque they cause. Each propeller has its own motor and its thrust depends only upon rotation speed. This approach brings both ease of construction and maintenance and is suited mostly for small electronic models. In order to enhance the reliability and stability of the vehicle, several pairs of propellers can be added, which forms the basis of the hexacopter

(a) Standard helicopter

(b) Coaxial helicopter

(c) Quadrocopter

(d) Octocopter

Figure 2.1: The most common R-UAV rotor configurations

and octocopter (2.1d).

2.3 Manoeuvers

In order to explore available manoeuvers of the R-UAV platform, we chose the quadrocopter as a representative. At first, we designate the parameters which we wish to measure and affect. Those are the three-dimensional velocity vector (v_x, v_y, v_z)and the three angles of rotations called pitch, roll and yaw. As shown in Figure 2.2, pitch corresponds to a y axis, roll to an x axis and yaw to a z axis. All parameters are measured at the mass center of the air vehicle. Vertical velocity can also be referred to as gaz. This name is preferred in the thesis.

Movement of the quadrocopter is accomplished entirely by adjusting the thrust of the rotors. We will assume the quadrocopter is hovering in an ideal environment with no turbulence. When all propellers are provided the same thrust, the quadrocopter either increases, decreases, or hovers at a constant altitude. If the given thrusts of the rotors differ, certain rotation angles will change.

How to separately change the pitch, roll, yaw and gaz is shown in Figure 2.3. When we increase or decrease the thrust of all propellers simultaneously, we alter only the vertical speed. To change pitch or roll, we must affect an unbalance between the corresponding opposite pairs of propellers. On the other hand, unbalance between the thrust of the propellers on the first and second diagonal causes the quadrocopter to perform yaw rotation. In order to change more parameters simultaneously, we may combine those schemes.

Changing the angles affects horizontal velocity – changing the pitch affects

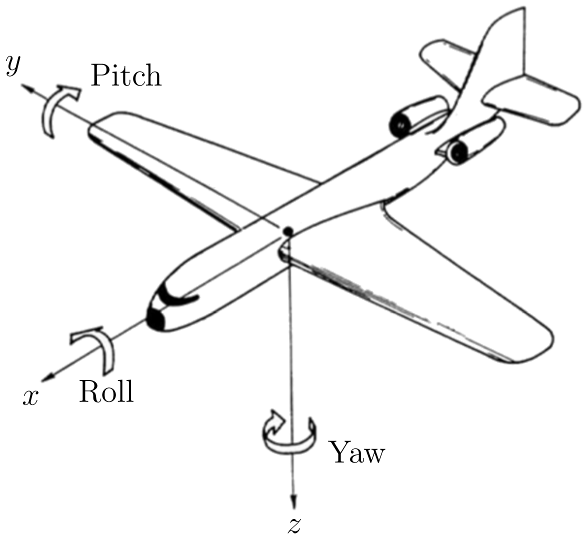


Figure 2.2: Rotation angles of an air vehicle

v_x and the roll affects v_y. Their dependency is not linear and varies for different types of quadrocopters. [5, p. 9]

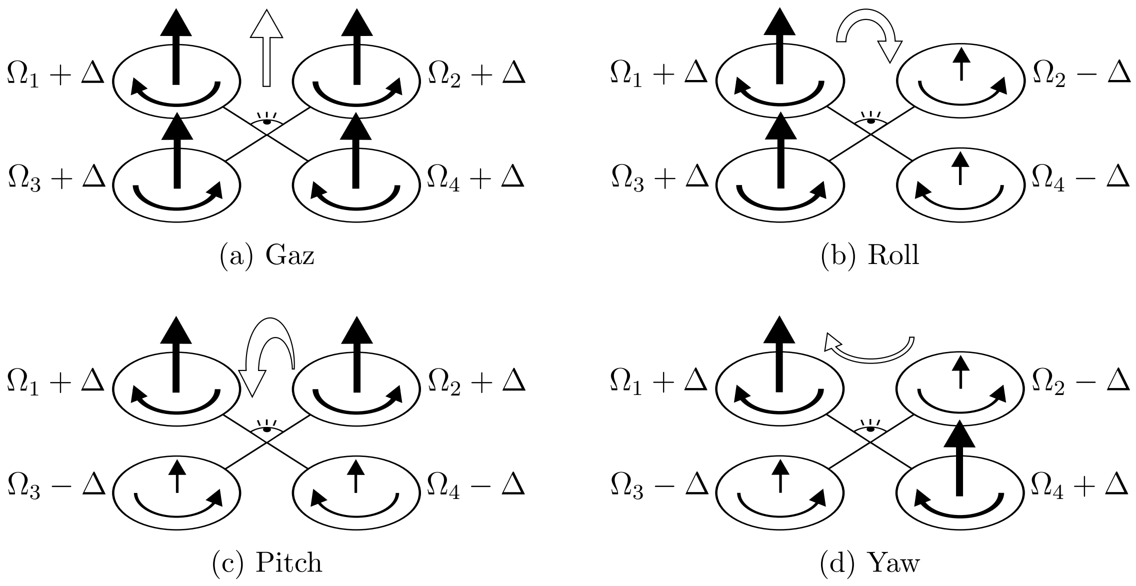


Figure 2.3: Adjusting gaz, roll, pitch and yaw by modifying propellers' angular velocities
Symbols Ω_1, Ω_2, Ω_3 and Ω_4 denote the angular velocity of the propellers before the operation and Δ denotes the amount of their change.

2.4 On-board Controller

The example of the quadrocopter will also be used to present the on-board controller, which is in fact the "brain" of the R-UAV. It is responsible for interpreting commands received from the pilot and usually also provides a certain level of autonomy. The lowest level is the ability to stabilize the vehicle, while higher levels include following a route given by GPS coordinates, detecting and avoiding obstacles or performing reconnaissance over a certain area. Because in this thesis we are not concerned with autonomy, we will only focus on stabilization ability.

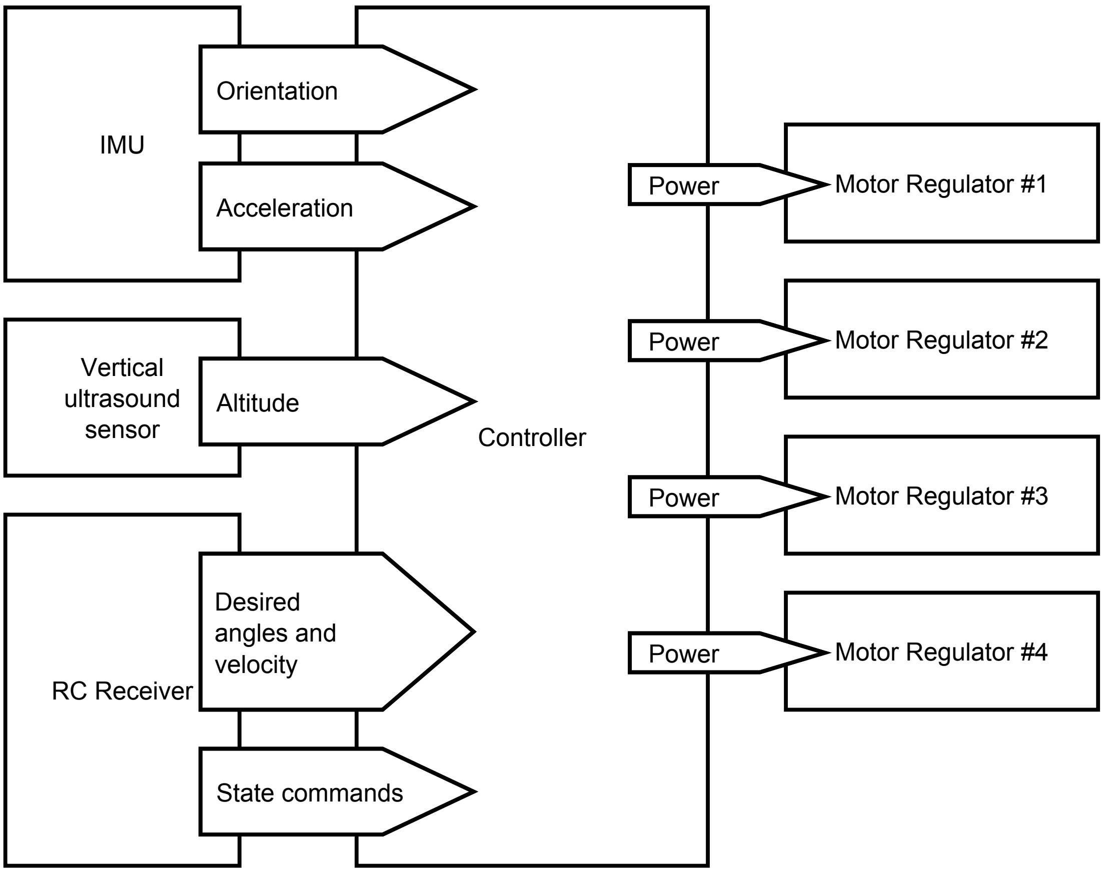


Figure 2.4: On-board controller schema
Communication between the on-board controller and other parts of the R-UAV

In Figure 2.4 shows how the controller is usually interconnected with other parts of the vehicle. The Inertial Measurement Unit (IMU) is a set of sensors responsible for determining actual acceleration and orientation of the R-UAV. Other sensors may be available, for example pressure and ultrasound sensors to measure altitude. Using image-recognition techniques, it is also possible to gather information from a camera (commonly used to make velocity measurements) more accurate. Another input of the controller is commands from the pilot. Usually every command comprises desired values of the four aforementioned flight parameters: pitch, roll, yaw and gaz.

The task of the on-board controller is then to adjust the thrust of the propellers according to the comparison of current and desired state, stabilizing the quadrocopter in that state. A proportional-integral-derivative (PID) controller, known from industrial control systems, is commonly used as the implemented algorithm. Other advanced approaches, such as the linear-quadratic regulator (LQR) and H-infinity (H_∞) can also be used; hovever, according to the study in [11], PID is sufficient for non-aggressive flights.

Besides its stabilization capability, the controller can also launch pre-programmed sequences. We might imagine them as collections of instructions stored in the quadrocopter's firmware. Those instructions describe actions during the sequence, e.g. how to change propeller thrust, or how to react to the gyroscope or altitude.

Probably the most important of the pre-programmed sequences is take-off. When the quadrocopter hovers just a few centimeters above the ground, it is exhibit to huge turbulence. Therefore, the thrust of the propellers must be raised for a short period. After the quadrocopter is airborne, the thrust can be diminished.

Another important sequence is landing. The thrust of the propellers must be lowered in order to descend, but not so radically as to result in a crash. Then it must recognize when it is on the ground, and turn off its rotors.

As we mentioned before, some quadrocopters are capable of determining their horizontal velocity (v_x, v_y) and position. On those, a position-holding sequence can be implemented. It uses simple pitch and roll adjustments to move or stop the quadrocopter on a horizontal plane. On some quadrocopters, a variety of other pre-programmed sentences may be available, such as a vertical flip, vibrations, a simulated rotor malfunction, etc. [8, p. 81]

3. Control Devices

Having reviewed the fundamentals of the R-UAV structure and behavior, we will analyze several devices available on the market that can be utilized to control R-UAVs. For every device we will inspect its technical requirements, the interface it provides to the user and summarize its advantages and disadvantages. Inspected devices are: RC transmitter, smartphone, gamepad, joystick, 3D Connexion Space Navigator and Novint Falcon.

3.1 RC Transmitter

When Radio Control (RC) is used as a communication method, it usually consists of a receiver on the R-UAV side and a transmitter on the user side. We use the term 'transmitter,' even though some of those devices are also capable of receiving data from the R-UAV.

The common structure of a hobby RC transmitter is described in Figure 3.1. The antenna (A) is used to send signals to the receiver on the R-UAV side. The user generally holds the transmitter in his hands or has it attached to his body by a belt. He uses thumbs or a combination of thumbs and index fingers to move the sticks (B) in horizontal and vertical directions. Both sticks affect two flight parameters. Commonly, the left stick sets roll and pitch, whereas the right stick controls yaw and gaz. All four axes have corresponding scroll bars (C) used to adjust the sensitivity of the user input.

Although the RC transmitter is probably today's most popular R-UAV control device, it is quite unintuitive and difficult to understand for beginners.

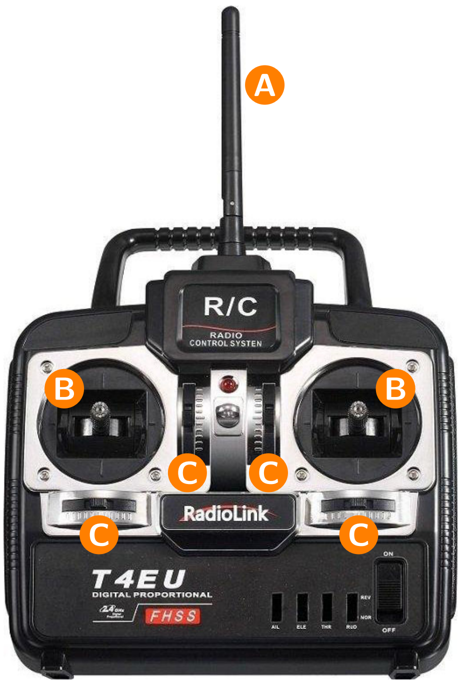


Figure 3.1: F03353 RadioLink RC Transmitter
(A) Antenna, (B) Sticks, (C) Scroll bars

3.2 Smartphone

We understand this term to define a mobile phone equipped with a multi-touch screen and capable of installing additional applications. The smartphone can be used to control R-UAVs when the both sides contain the same wireless communication technology, such as Wi-Fi or Bluetooth.

In Figure 3.2 we show a screenshot from AR.FreeFlight, an application to control the Parrot AR.Drone. On the smartphone screen (A) we see the picture from the AR.Drone camera. It is overlaid by the Take-off/Land toggle button (B) and two virtual handles (C).

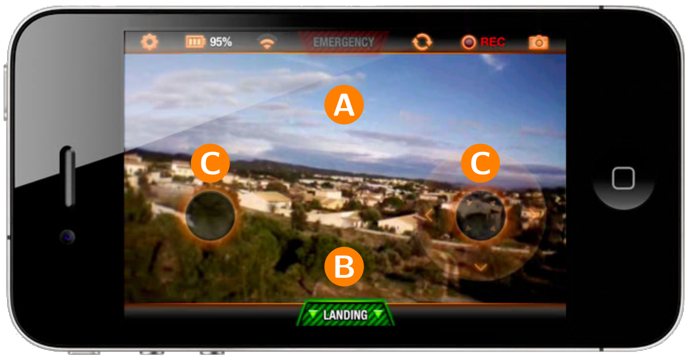


Figure 3.2: Ar.Drone FreeFlight application on Apple iPhone
(A) Picture from the camera, (B) Take-off/Land toggle button, (C) Virtual handlers

Functionality differs according to the user-settings and the capabilities of the smartphone. In a basic case, the handles are used just as the two sticks on the RC transmitter. When the smartphone is equipped with a gyroscope, its tilt can be used to control the pitch and roll, replacing the function of the left handle. If both the AR.Drone and the smartphone have a compass[1], the user can use the *absolute control*. This means that the given navigation commands are not relative to the AR.Drone orientation, but to the orientation of the smartphone.

Both the use of the smartphone gyroscope and compass enhance the user experience. However, we do not consider the yaw and gaz control as very intuitive, as it still resembles the RC transmitter. Another problem is that the view from the R-UAV camera is hidden behind the user's thumbs. This can be solved by using a tablet instead of a smartphone. An advantage of the smartphone over the RC transmitter is that the author of the piloting application can easily extend it to use more of its features, such as vibrations.

3.3 Gamepad

The gamepad is a controller originally used to play games on a computer or a video game console. Even though some gamepads use Bluetooth connection instead of wired, R-UAVs do not usually have an option to be controlled directly

[1]AR.Drone 2.0 contains a magnetometer, which is an extended version of the compass. The first version of AR.Drone does not include any of these.

by the gamepad. Instead, it is only a user interface for the computer or another device communicating with the vehicle.

A typical modern gamepad is described in Figure 3.3. Similarly to the RC transmitter, it is held in both hands and mostly the thumbs are used to provide input. The gamepad features the directional pad (A) on the left side, the set of four action buttons forming a cross (B) on the right side and two sticks in the center (C). [12]

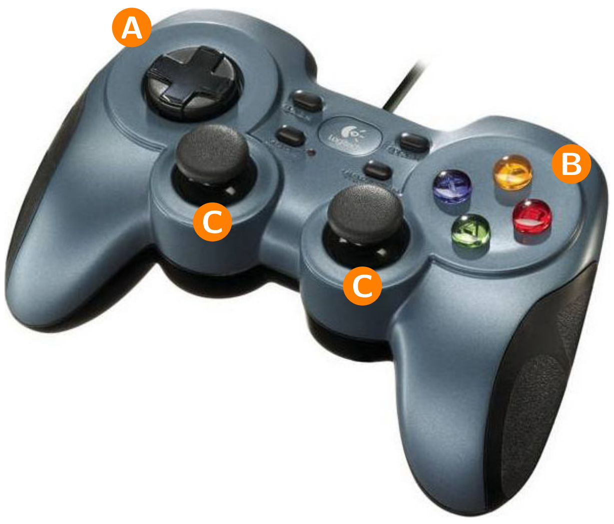


Figure 3.3: Logitech Rumble Gamepad F510
(A) Directional pad, (B) Action buttons, (C) Sticks

The most popular method of using the gamepad to control an R-UAV is to use the sticks in similar fashion to those on the RC transmitter. As some gamepads provide information about the pressure needed to push particular buttons, the directional pad and action buttons could also be used to control the flight parameters.

From the user interface point of view, using a gamepad instead of the RC controller provides no essential advantage, as the interfaces are quite similar. The main technical requirement - the presence of the computer - brings both advantages and disadvantages. On the one hand, the programmer can use its features (big screen and high computing power) to affect more advanced techniques, such as image recognition. On the other hand, the user must carry a computer everywhere he wants to fly the R-UAV. All the devices listed below are dependent on a computer.

3.4 Joystick

Alongside the gamepad, the joystick is the most widespread game controller. The professional variants of the joystick are used as the primary human interface controllers on many vehicles, including aircraft.

A common joystick can be seen in Figure 3.4. It consists of the stick (A) attached to the base (B). Depending upon joystick type, the user can rotate the stick in two or three degrees of freedom, which is the primary input. The action buttons (C) can be attached to both parts and some joysticks also have a *throttle* (D) - the one-dimensional input. Special joysticks are further capable of vibrations and *force feedback* - the ability to exert some force on the stick.

The joystick may probably be the best choice for players of aircraft simulator games, because they are accustomed to using it. Moreover, if the joystick supports rotation in the three degrees of freedom and contains the throttle, the angles and gaz of the R-UAV can be quite intuitively mapped to the orientation of the stick. Regarding the need of a computer, the joystick has the same benefits and drawbacks as the gamepad.

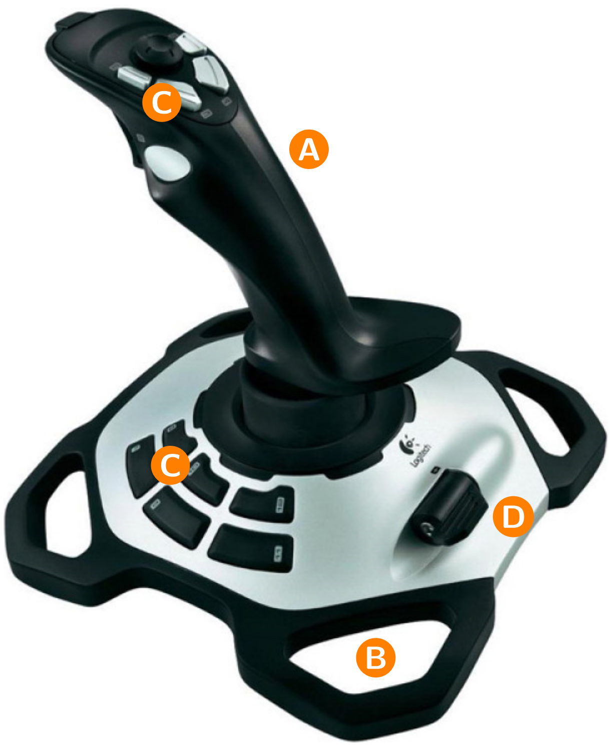


Figure 3.4: Logitech Extreme 3D Pro joystick
(A) Stick, (B) Base, (C) Action buttons, (D) Throttle

3.5 3D Connexion SpaceNavigator

People who work in a 3D environment, such as 3D graphic designers, encounter many difficulties when using only a mouse and keyboard. One of those is navigation in space. To ease this task, the 3D Connexion company created several products known as 3D mice; namely SpacePilot Pro, SpaceMouse Pro, SpaceMouse® Wireless and SpaceNavigator. They mostly differ in their number of additional buttons, although the main principle of control remains the same.

As shown in Figure 3.5, SpaceNavigator is a light-weight device resembling a button that fits in the palm. The handle (A) sitting on the base (B) can be both simultaneously rotated and translated in three dimensions. There are also two buttons (C) available, optimized to be pushed by thumb and little finger.

If we implement the control in a way that the R-UAV copies the rotation from the device, it will allow the user to virtually 'hold the R-UAV in the hand.' Therefore, we expect this device to be more suitable for R-UAV control than all those

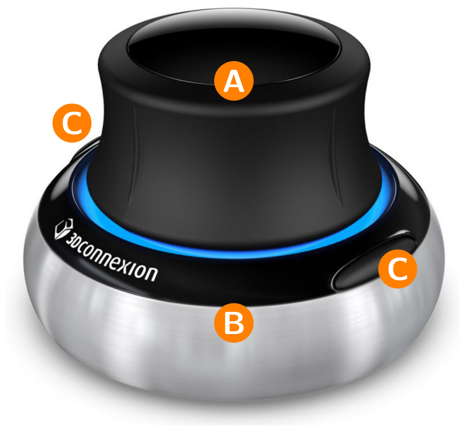


Figure 3.5: 3D Connexion SpaceNavigator.
(A) Handle, (B) Base, (C) Buttons

previously mentioned. The only inconvenience is the inability to communicate directly with the R-UAV.

3.6 Novint Falcon

In order to enhance computer games player experience, the Novint Company chose to use the sense of touch. The Novint Falcon is a *haptic* device, meaning it is possible to both receive input according to the user's manipulation and to exert some force against the user. In order to work, it needs to be connected both to a computer and power supply.

As we see in Figure 3.6, the Novint Falcon consists of a grip (A) attached by three arms to a body (B) that stands on a base (C). At the top of the grip, four buttons are located. The standard grip shown in the picture can be replaced by a pistol-grip, which is used mostly for playing computer games.

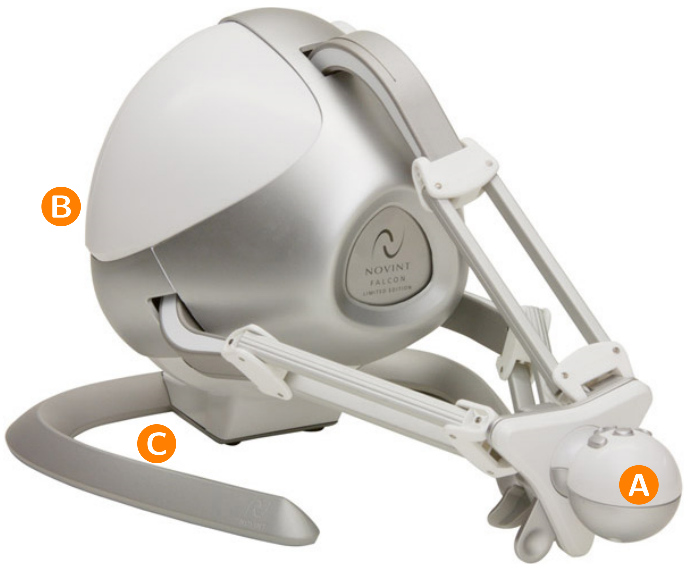


Figure 3.6: Novint Falcon.
(A) Handle, (B) Body, (C) Base

The Novint Falcon has big potential as a R-UAV control device, because it allows the user to operate in much larger spaces than the 3D mouse. Also, as a haptic device, it can provide an interactive experience. The most significant problem with the device is that it needs to be connected to a power supply, which makes it cumbersome to use outdoors.

4. Problem Specification

As we see, the R-UAV is a very specific aerial platform with several unique abilities, such as hovering in constant position and VTOL. Moreover, it offers better stabilization and provides greater freedom of movement in space than the fixed-wing alternative. The most common way to remotely control an R-UAV is to repeatedly send it the desired flight parameters: pitch, roll, yaw and gaz. Its on-board computer is then responsible for altering the thrust of particular propellers or rotors in order to meet the requirements. Some R-UAVs are also capable of launching preprogrammed sequences upon request, such as take-offs and landings.

In order to control the R-UAV with a device, it must provide a way for the pilot to independently manipulate all four flight parameters and launch the preprogrammed sequences. We summarized a list of several devices that are commonly available on the market and meet those requirements. The RC transmitter is the industrial standard for small remotely controlled vehicles; however, with its 'two sticks' interface it is not very intuitive for beginners. We also chose the smartphone, which became popular in this field because of its ability to control the Parrot Ar.Drone. The gamepad and the joystick also appear on our list, as they represent standard computer game controllers known to many users.

Another type of controllers are the new 3D controllers, which have recently become known to computer game players and 3D graphic designers. From among those, we chose the Novint Falcon and 3DConnexion SpaceNavigator. The Novint Falcon is an advanced 3D positioning device allowing force feedback on all those three axes. The 3DConnexion SpaceNavigator (often called 3D mouse), is a button that fits into the palm and can be rotated and translated in all three degrees.

As we stated in the goal of this thesis, we want to solve the problems that arose from the extensive use of the "two sticks" mechanism with the R-UAV remote human control, which proved not to be very intuitive for beginners. We will summarize the tasks that need to be done in order to accomplish that goal:

First, we need to choose a particular R-UAV that will serve as a testing platform. The next important step is to propose control devices suitable to control the R-UAV.

Second, we will build an application that allows us to use the particular devices to control the R-UAV. The application must be extendable, enabling the addition of other control devices in future. Also, it must provide the user a method for configuring the parameters of the controlling device, such as buttons and axes mapping, sensitivity etc.

Last, we will design an experiment in which a group of volunteers will fly the R-UAV, using the chosen controllers. We need to design several navigation scenarios for these pilots and choose what quantities to measure among all the flights. After performing the evaluation experiment, we must interpret the results, essentially comparing the devices and deciding which devices to recommend as a future method of R-UAV control.

Part II

Implementation

5. Proposed Solution

We selected Parrot AR.Drone as an R-UAV to test our controlling mechanism. It is a widely known small size quadrocopter, originally created as a toy. However, for its convenient price and availability it became popular across universities and among technical enthusiasts. Unlike the majority of hobby quadrocopters, it utilizes Wi-Fi as a base communication protocol. Its most common remote controller is smartphone, but a custom control mechanism running on a computer can also be implemented, because the communication protocol is public.

Thanks to that, we can utilize all the devices that can be connected to a computer. Namely, we selected those previously mentioned: gamepad, joystick, 3D mouse, Novint Falcon and the smartphone, which is supported locally. We only skipped the RC transmitter, as it is very complicated to connect either to Ar.Drone or the computer. However, the mechanism of 'two sticks' is also available on the gamepad, so the lack of RC transmitter will not have a negative inpact on the trustworthiness of the research.

We understand Ar.Drone and the control devices as separate entities, where each entity can provide output and consume input. The purpose of the application is to interconnect their inputs and outputs, providing a simple user interface for piloting.

Therefore, we decided to build the application upon a component system. Every component in the system will be capable of providing a configuration and data interface. The configuration interface consists of a set of configuration parameters, whereas the data interface comprises unique typed data inputs and outputs. We call them sinks and sources respectively, as these terms are common in this area. The component system to instantiates particular components, configures them and connects their data interfaces, enabling simple aggregation capabilities. The component system was named Glue, as it is a tool to join together particular parts of the application.

The next step is to create the Glue components, representing individual entities: the Ar.Drone component, control devices components, control mapping components and the user interface component.

The Ar.Drone component represents a handler for the vehicle, automatically connecting to it and then providing information concerning connection status, telemetry and the stream from the on-board camera. The sinks of this component will accept commands to alter drone flight parameters and change its state, such as in landing or take-off.

Most of the control device components contain only sources. Their structure depends on the number and composition of the axes each device provides. One source on every component also provides information concerning pressed buttons. As Novint Falcon is a haptic device, its component will contain a sink that enables sending it a three dimensional vector representing the force to perform.

For every control device component there is a corresponding control mapping component. It obtains user input from the device and translates it into commands for the drone component. The method to accomplish this is highly configurable.

The user interface component is based on the Qt Quick cross-platform framework. As a configuration parameter it will expect a file in QML format, describing the user interface in a declarative way. The user will see the video from the drone on-board camera, the actual drone telemetry and the battery status. We will also provide a way to see the list of available devices and select the one to be used to control the drone.

When the application is assembled, we will proceed to the experiment. Its detailed plan and the results obtained are written below in the third part of the thesis. The current part contains details concerning the testing application implementation.

6. Glue Component System

Glue was created with simplicity in mind. Its purpose was to enable a programmer to easily write particular components with clearly specified interfaces and then interconnect them to form the resulting application.

As a trade-off for that it is expected to consume some additional CPU and memory, especially during the application startup. It is designed to run only in one process, not distributed accross several processes or even devices.

It is programmed in C++11, currently the most recent standard of the C++ programming language. It is dependent on several Boost libraries, mainly on Boost.Asio and Boost.Signals2. Because this thesis is one of the projects of the AADrone group, it uses their common libraries for event-logging and mathematical operations.

6.1 Entities

In Figure 6.1 we illustrate a simple example of how the system may be used. On the left side there is a component representing a handler to an elevator, whereas on the right side we have two components to manage its altitude. The `Elevator` provides the last measured altitude and accepts the desired altitude, with the possibility of setting the frequency of communication with the actual device. The `Higher` and `Lower` components simply receive the measured altitude, add or subtract a value to it respectively and send it back. They are both arranged in a way that disallows their use simultaneously, so only one must be explicitly chosen.

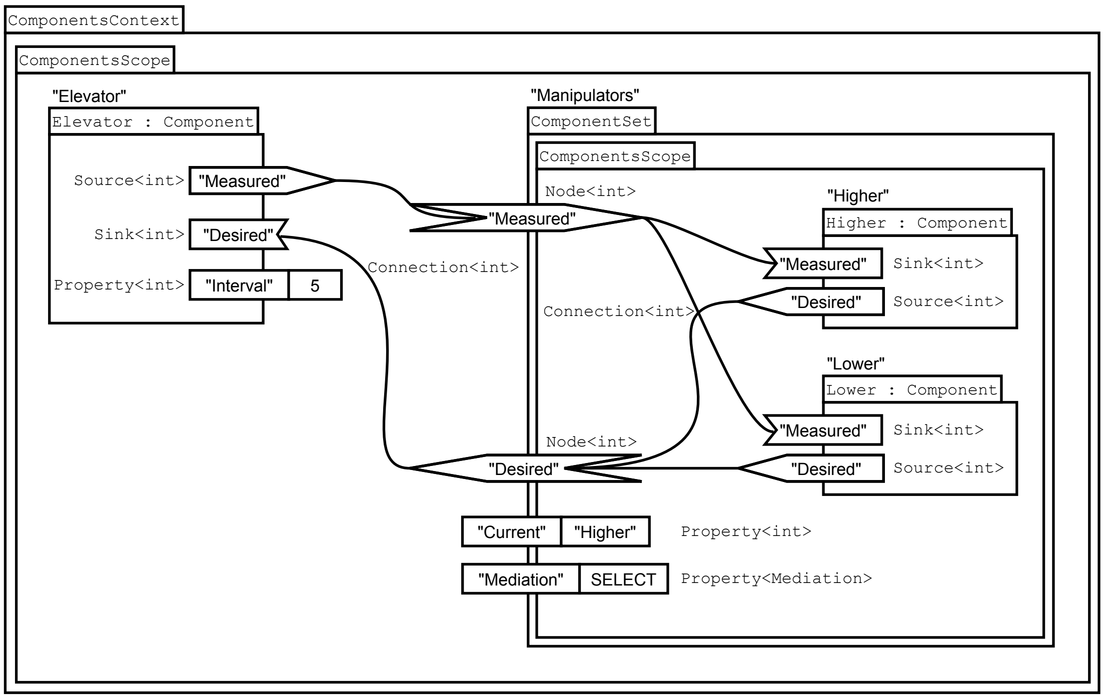


Figure 6.1: An example of Glue use

We will inspect the most important Glue classes present in this example. `ComponentsContext` serves as a container for all the other entities. It manages their construction and contains a `boost::asio::io_service` instance, through which it takes care of the execution flow.

`ComponentsScope` may be understood as a name-scope for components and their sources and sinks. The names of those sources and sinks may collide; however, every component in the scope must be listed under a unique name.

`Component` represents an application fragment with a particular function. Programmers are encouraged to use the `io_service` object provided by the context when performing non-blocking asynchronous operations. This should fit the common tasks for which the components are created, such as waiting for user input or sending data to an external device. Operations that may possibly take a lot of time to complete are better moved to a separate thread, in order not to block the data flow in the component system. Components mostly communicate with the rest of the application using their endpoints.

Every `Endpoint` provides the possibility to be marked as required. It can either be active or inactive and how these states are implemented differs among the subclasses. `Endpoint` is a superclass of `Propery`, `Source` and `Sink`.

Properties can be used as a method for configuring the component, as well as to present certain information about it, which is not very often changed. Every property holds a nullable value of a specific type and is active when the value is set. Upon creation, it can be given a validation callback, so that not all the values of the given type are allowed for that property.

`Source` and `Sink` are closely related to each other, as their function is to transfer strongly typed data between the components. Every source can send data to multiple sinks and similarly, every sink can receive data from multiple sources. Both source and sink are considered active when they are connected to at least one of their counterparts.

To represent this relation we have the `Connection` class. Once established between a source and a sink, it enables data delivery. This may be marked as closed to block all data coming through, which could be used in certain aggregation scenarios.

For those purposes the `Node` class was also created, which inherits both from `Source` and `Sink`. Its function is simple: whenever it receives data from an incoming connection, it passes it to all the outgoing connections.

Those features are used by `ComponentSet`, a specific implementation of `Component`. As is apparent, it encapsulates its own `ComponentsScope`, into which components may be added. For every source and sink of the unique type and name the contained components create, `ComponentSet` creates a node and publishes it to both scopes. As in the example, it may happen that certain of these nodes have multiple connections in the inner scope. How to behave in this situation depends on the "Mediation" property. When set to `UNITE` mode, no restrictions are applied; but when set to `SELECT` mode, only connections from one component remain open and the rest are closed. Which component is active depends on the `Current` property, where its name is specified. Another possibility of `ComponentSet`, which is not shown in the picture, is to contain a `ComponentGenerator` class instance. If that is used, the generator takes care of creating components and adding them to the scope.

It may be unclear, how ComponentSet discovers that a component in its inner scope publishes an endpoint. Moreover, how it even *knows* that there was a component added to the inner scope. Thanks to Boost.Signals2 library it is rather simple, because the entities mentioned make similar events available as signals and anyone can subscribe a listener to them.

Another interesting topic is how we deal with the strongly typed endpoints. In the DataTypesContext class instance there is a map containing factories for the endpoints and connections of all the data types used. It is indexed by the std::type_index of those types. Whenever a component creates an endpoint, its type is checked against this map and if its factory is not yet present, it is added. In case of properties, there are also stored parsers, enabling encoding the property value from a string.

Similarly to that, ComponentsContext contains factories for components and component generators. However, those are indexed by their name and need to be added manually in order to use them.

6.2 Lifetime

We have already described Glue from the static perspective, now we will explain how it usually behaves throughout its lifetime. At first, a ComponentsContext class instance is created along with the main scope. Then there are basically two methods of adding components into the scope and initializing them: programmatically or declaratively.

In the case of the first approach, we must create all the components in the code and register them under their names in the scope. In order to alter their properties, we obtain them by their names from the components and set them to the appropriate values.

If we choose the declarative approach, we must provide the context the factories of all the components and component generators we expect to use. Then, we create an XmlLoader class instance and provide it an XML document in a specific format. This document describes how to instantiate the particular components and select the values to set up their properties.

Every component may be, throughout its lifetime, within four different states: *uninitialized*, *inactive*, *active* and *destroying*. Right after creation it is in the *uninitialized* state, meaning it has not been yet added to a scope and not yet created endpoints. When the component is added to a scope, the Initialize() virtual method is run, in which the component creates its endpoints (sinks, sources and properties). After initialization the component is in either an *inactive* or *active* state, depending on whether all the required endpoints are activated. If the component elects to destroy itself, it will change its state to *destroying*, causing its scope to destroy it.

Now we are at a point where all the components have been created, published their endpoints and their properties are set. The next step will be to interconnect their sources and sinks by calling ComponentsScope::ConnectEndpoints(). For the sake of simplicity, we decided to connect every pair with the same name and data type. It might be reasonable to provide a way to make this process more customizable, but for our needs this approach is sufficient.

By calling `ComponentsContext::Run()` we proceed to the main part of the application, where most communication is completed by transferring data between sources and sinks. It is also expected that a component may change the property value of another, as in the case of the 'CurrentItem' property of `ComponentSet`.

In order to terminate the application, `ComponentsContext::Stop()` must be called. The context will cancel all the pending asynchronous operations and destroy all underlying entities.

7. AR.Drone

Parrot AR.Drone is a light-weight quadrocopter intended for wide audience. In the following section, we provide general information about it. Next, we present the implementation of its Glue component.

7.1 Features

At the time of the writing, the latest version available is Ar.Drone 2.0. We will inspect its technical specifications, software capabilities and how an application communicates with it.

7.1.1 Hardware

As seen in Figure 7.1, Ar.Drone 2.0 can be decomposed into several independent parts. A mainframe (A) is made from carbon fibre tubes. Motors (B) have performance of 14.5 Watt and are capable of 28,500 revolutions per minute (RPM). They are attached to propellers (D) by gears (C). A main body (E) is protected by an expanded polypropylene shield and from this material are also made the indoor (Figure 7.1a) and outdoor (F) hull.

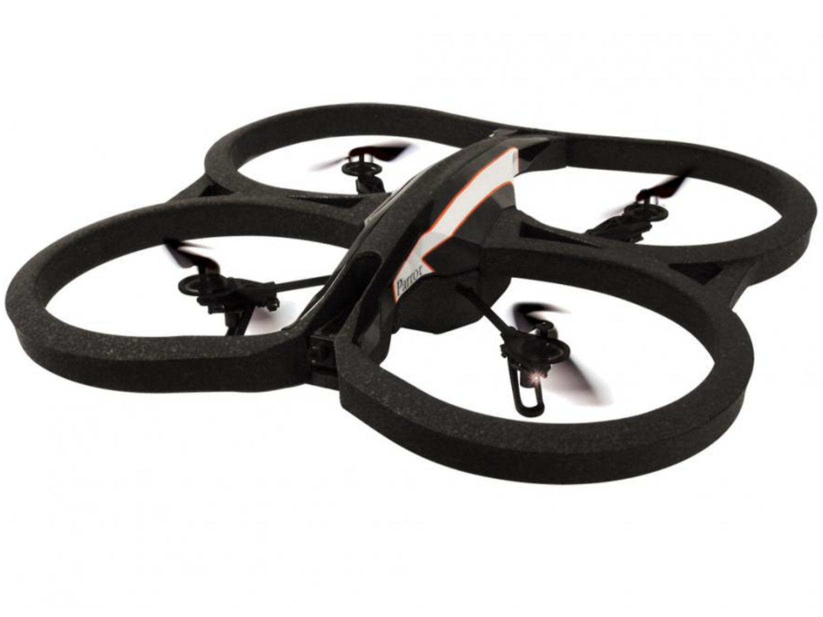
Apart from the rotation detection using IMU, it is also capable of detecting altitude, azimuth and horizontal speed. For altitude it contains an ultrasonic sensor (G) and pressure sensor. The azimuth is gathered from a three axis magnetometer. Its horizontal speed is acquired by a simple image recognition algorithm of the output from a bottom-facing camera. There is also a HD camera (H) in the front, with the maximum resolution of 1280 x 720 pixels.

We can also plug a storage device to the USB port in the main body. Storage can be used to save flight data and video stream and Parrot also sells a USB device that provides the AR.Drone 2.0 with a GPS. Communication with the control device is done via the Wi-Fi adapter. Computing power of the device is relatively high. Most of it is provided by 1GHz ARM processor and 1Gbit DDR2 RAM, there is also an 800MHz processor for video encoding.

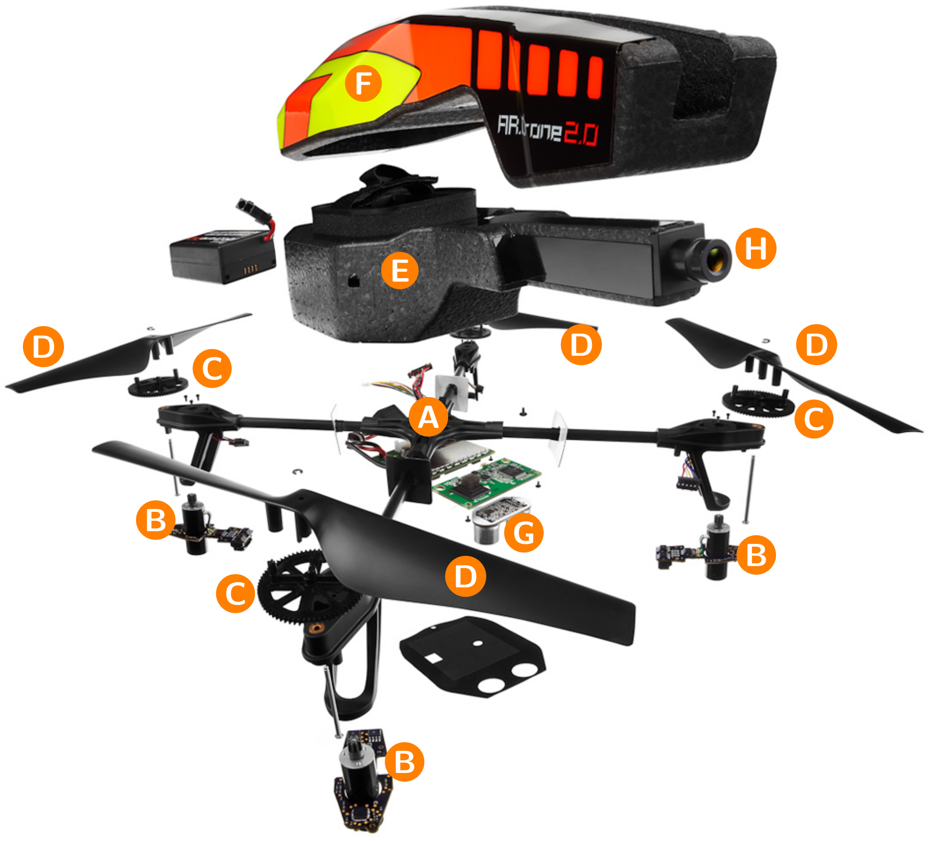
7.1.2 Software

The operating system uses a version of embedded Linux, modified by Parrot. The on-board controlling mechanism, communication with the control device and video processing are implemented as a Linux executable file. We will call the file *firmware*, although that name would fit the embedded Linux as well. Thus, a *firmware update* is actually just a replacement of the file. As Parrot does not provide the source code of the firmware, as this information may be convenient to people developing their own.

Apart from the stabilization, the original firmware is capable of take-off, hovering in one place, looping and landing. It is also responsible for communication with the control device.



(a) Indoor hull



(b) Outdoor hull, decomposed
(A) Main frame, (B) Motors, (C) Gears, (D) Propellers, (E) Main body, (F) Outdoor hull, (G) Ultrasonic sensor, (H) HD camera

Figure 7.1: Parrot AR.Drone 2.0

7.1.3 Communication

Communication with AR.Drone is done over Wi-Fi. It creates a network and expects the control device to connect to it. When the control device is connected, it obtains an IP address and can use ports 5554, 5555, 5556 and 5559 as communication channels.

On UDP port 5556 the control device sends *AT commands*. Those are text messages used to control flight angles, configuration and sensor calibration. Navigation data, such as status, angles and velocity are sent to the control device on UDP port 5554. A video stream is available on TCP port 5555. The protocols of all the channels are described in the AR.Drone Developer Guide [8].

As Parrot encourages developers to create applications working with AR.Drone, it provides *ARDroneTool*, a framework written in C and allowing the programmer to use callbacks and function calls rather than implementing all the protocols. Apple iOS application developers can also use the AR.Drone Engine, is a common base for all applications meant to control AR.Drone.

7.2 Component Implementation

For AR.Drone communication component we decided not to use the *ARDroneTool*, as it is a rather complex and cumbersome framework, into which we would have little insight. Instead, we created a simple stand-alone library implementing the given protocols. Its functionality is distributed between four classes: `ATCommandsSender`, `NavdataReceiver`, `ConnectionHelper` and `PaveVideoReceiver`. All utilize Boost.Asio and Boost.Signals2 libraries to behave as separate building blocks, providing subscriptions to asynchronous events.

As its name suggests, `ATCommandsSender` is used to send the AT commands to the drone. Available commands are represented by classes that inherit from `ATCommand`. If needed in the future, new commands can be easily added.

`NavdataReceiver` listens on the appropriate port in order to periodically receive navigation data. It provides an interface to subscribe to the event of a certain data receipt, for example to status or angles information.

`ConnectionHelper` utilizes both `ATCommandsSender` and `NavdataReceiver` to manage connection with the drone. It handles its initialization and maintenance, as well as restarts in case of disconnection.

For video frames transmission, AR.Drone uses Parrot Video Encapsulation (PaVE) format. That is where the name `PaveVideoReceiver` originates. It enables receipt of particular video frames, where every frame contains the information concerning the used codec, resolution, its position in the stream, etc.

`ARDroneGlueComponent` uses the mentioned classes to incorporate AR.Drone into Glue. In the following discussion we will inspect its interface. Its only property is `DroneState`, which holds information on the current state of the drone.

There is also a source of the same name, from which the state of the drone is sent whenever it changes. Other important navigation data also have their sources, namely `DroneAngles`, `DroneVelocity`, `DroneAltitude` and `DroneBattery`. `DroneCamera`, a source that provides the particular video frames encoded in the original codec.

The component contains two sinks. `DroneStateCommand` accepts commands to change the drone state, the most important being `TAKEOFF` and `LAND`. `DroneFlyCommand` is used to send the desired pitch, roll, yaw and gaz to the drone.

8. Control Device Components

The task of this chapter is to describe the method by which particular control devices are plugged into the component system. Every device is represented by at least two components. The first serves as a device driver, which implements device inputs and outputs to the component system. The second, called the control mapping component, provides methods to map axes and buttons of the device to commands for the drone.

In the first section we inspect the ‘driver’ components for the joystick, gamepad and 3D mouse. Their components are very similar, because all the devices mentioned belong to the USB human interface device (HID) class, whose characteristic is summarized there. Novint Falcon cannot be utilized like that; therefore, its component implementation is explained in the second section. The last section reveals how the control mapping component is implemented.

8.1 USB HID

USB HID class[10] is a part of the USB specification for keyboards, mice, game controllers and other human interface devices. Every device in this class is able to provide a descriptor, in which are described the protocols of all the possible reports to be sent between the device and its host. That allows the host to map every byte on every report to the corresponding feature of the device, such as rotation, translation, button press, etc.

Moreover, thanks to the HIDAPI library it is simple to work with these devices. The library is a lightweight cross-platform layer, working on Windows, Linux and Mac OS X. It enables the programmer to enumerate the available USB HID class devices, connect to them and listen to their reports.

An alternative approach to enable control devices usage in a cross-platform manner would be to use Simple DirectMedia Layer. It is a robust library created, among others, to provide simple and unified access to game controllers. However, we chose HIDAPI, as it provides finer control of communication with the devices. Furthermore, if we develop a custom controlling device in the future, it will be easier to add a support for it.

The only downside of this approach is that in order to create a general piloting application, working with any type of input device, we must implement the HID descriptor parser and create the endpoints dynamically. Although technically feasible, it is not the goal of this thesis. The application is created only as a tool for research and, as such, is not required to support all the joysticks and gamepads available on the market. Instead, we selected a specific model of every device type. As we know all its capabilities, we can ‘hardcode‘ the endpoints of its component.

Speaking of the components, we must mention the `HidGlueGenerator` class. When supplied to a `ComponentSet`, it fills it with components of the currently connected USB HID class devices. Every component contains a HIDAPI handler of the particular device. Whenever it receives a report from the device, it propagates the axes and buttons data to the component system. The explanation of the particular components follows.

As the joystick we chose Logitech Extreme 3D Pro (Figure 3.4), because it enables rotating the stick in three degrees of freedom. Also, it contains the throttle, giving altogether four axes to work with. Logitech Rumble Gamepad F510 (Figure 3.3) is the second chosen device. As 3D Connexion Space Navigator also belongs to the USB HID class, it is handled in this way as well.

8.2 Novint Falcon

Communication with Novint Falcon is done by using Novint Haptic Device Abstraction Layer (HDAL). [7] After the device is initialized, it creates a *servo* thread. Its purpose is to communicate with the device at the frequency of 1 kHz through callbacks that can be registered using HDAL. Callbacks can obtain the 3D position of the grip, status of the buttons and also provide the device a 3D vector of force to perform. This interface is published to the component system using the `FalconGlueHDAL` component.

In our application, we decided that without the user interaction, the grip will stay on a center position. Whenever it is moved away from it, it uses the force to center back. The intensity of the force grows linearly with the grip distance from the center, which forms a virtual 3D spring. However, such a spring alone would cause the grip to repeatedly bounce from side to side. In order to fix that, we also added a virtual damper, which determines the grip velocity and applies force in the opposite direction. Those two mechanisms proved to be sufficient for rough centering, because the center position then had a threshold of about 1.5 centimeter in every direction. Therefore, we also added a small amount of constant force targeting towards the center. That reduced the center position to the cube with the edge of about 4 millimeters. This behavior is implemented in the `FalconHandlerGlue` component.

8.3 Control Mapping

As the only aim of the control devices is to send commands to the drone, it would be possible to put the control mapping logic directly into their components. However, Glue allows us to perform a clear separation of concerns, which proved itself to be the right method for designing applications. That is why we put the mapping into a separate component called `DroneControl`.

Usually, components create all their endpoints when their `Initialize()` method is called. `DroneControl` creates just a portion of endpoints instead, namely `DroneStateCommand` and `DroneFlyCommand` sources to send the commands to the drone and a configuration property named `Mapping`. This property expects a list of device buttons mapped to particular drone commands and a list of device axes mapped to pitch, roll, yaw and gaz.

Axis mapping is slightly more sophisticated than button mapping, as we need to specify the transform function between the device and the drone command coordinates. First, we determine the axis of the device and the flight parameter to map it to. Second, we specify the maximum and minimum values of the axis, along with the threshold, e.g. the size of the 'dead zone.' These values will be used to map the data obtained to the interval of $[-1, 1]$. Finally, the magnitude of the result will be passed either to linear, quadratic or cubic function, according to the configuration.

When the `Mapping` property is set, the component creates the sinks for the buttons and axes input. Whenever it recognizes a state command from any of the buttons, it immediately sends the command through the `DroneStateCommand` source. In the case of axes, it performs a simple multiplex: the desired flight parameters are sent through the `DroneFlyCommand` source only when all four flight parameters are gathered.

Mappings of particular devices are quite straightforward. Gamepad utilizes its two sticks in the same way as the RC controller: left stick controls pitch and roll, whereas the right stick alters yaw and gaz. For taking-off and landing LB and RB buttons are used respectively. Joystick maps all the three rotation axes of the stick to the rotation of the vehicle, while gaz can be altered by throttle. The state commands mentioned are issued by buttons 1 and 2. The 3D mouse handles the rotation in the same way as the joystick; gaz is controlled by translation on the vertical axis. Left button is used to take-off and right button to land. In the case of the Novint Falcon, we use the x, y and z distance from the center to manipulate roll, pitch and gaz respectively. Yaw change is performed through the side buttons of the handle, while taking-off and landing through the center front and back buttons respectively.

9. User Interface

Qt is a robust cross-platform application framework currently developed by Digia, a Finnish software company. It comprises, among others, Qt Quick, a framework offering the possibility to build applications with rich graphical user interface (GUI). The GUI is usually created using a declarative QML language, utilizing Javascript for simple bindings, whereas the core application logic is implemented in C++.

For the integration into Glue, we created the `QmlLoader` component. A QML file specified in the `File` property is loaded together with any other QML files it depends upon.

In order to publish values from the component system to the Qt Quick, we utilize context properties. They allow publishing a `QVariant` or `QObject` instance to the Qt Quick runtime under a unique name, which can be then used by GUI bindings. `QVariant` acts as a union for the most common Qt data types, such as numbers, strings and vectors. When a new value for the property arrives, we overwrite the old value with it. `QObject` is the base class of all the classes used in the Qt framework. We use it to provide more complicated objects to the Qt Quick, for example `RawVideoSource`, through which we send video frames.

As we also wanted the GUI component to be configurable, it contains a `Sinks` property that allows specifying sinks to create on the component and the names under which they will be mapped to Qt Quick. During implementation of this feature we overcame an unpleasant problem. As the endpoints in our application are strongly typed, we would need to specify the type of every sink set up in this way. However, to determine the exact type from a string would be very problematic, as we would then need to have all the possible data types listed by their name in certain places. The `DataTypesContext` class provided by Glue does not suit this task, because it is filled dynamically and the `name()` obtained from the `type_info` is implementation-defined.

As a result, instead of directly creating the sinks, only *sink intentions* are created. Every sink intention observes the relevant component scope and, when a source with the given name appears, it creates the sink of the corresponding type, registers it in the scope under the same name and publishes it to Qt Quick runtime.

Another way the component interacts with the component system is by manipulating the `Current` property of a `ComponentSet`. This behavior can be specified in the `Switches` property, where particular switches are configured. Every switch must obtain the name of the component it is set to operate on and the string identifier under which to publish a `QtSwitch` class instance to the Qt Quick runtime. The `QtSwitch` class inherits from the `QAbstractListModel` class used to provide one-dimensional list to the Qt Quick runtime. When inserted into it, `QtSwitch` enables the GUI components to list the names of the `ComponentSet` children and to select one of them as active.

The final GUI created with the component for our application is shown in Figure 9.1. The screen serves the picture from the drone camera as the background (A). On the bottom line, we see the information concerning battery status (B), drone angles (C) and altitude (D). In the upper-right corner there is a drop-down

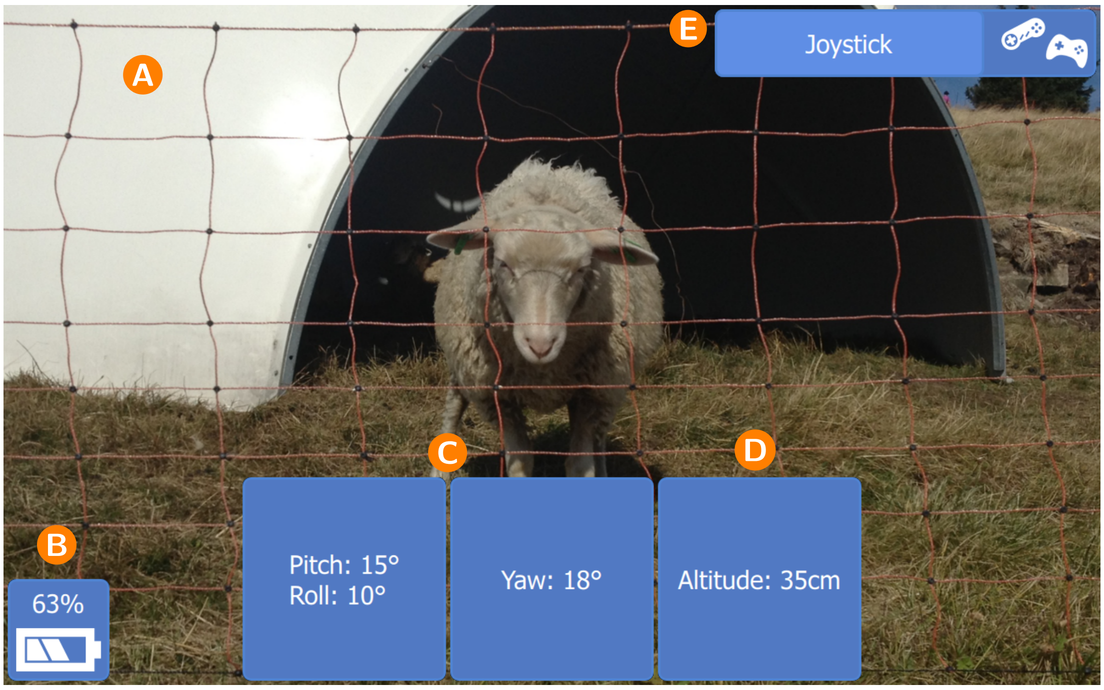


Figure 9.1: Screenshot from the piloting application
(A) Picture from drone camera, (B) Battery status, (C) Angles, (D) Altitude, (E) Control device selection

list of connected control devices (E), from which the user can choose the active one.

Part III

Evaluation

10. Experiment Design

In order to explore new ways of R-UAV remote control, we have proposed several devices comprising gamepad, joystick, 3DConnextion SpaceNavigator and Novint Falcon. We have also implemented a software solution that enables us to use those devices to control Ar.Drone quadrocopter. The next step of our research is to compare, how much intuitive and difficult to use the particular devices are.

Therefore we decided to design an experiment where will some volunteers use the devices to fullfil simple navigation tasks with the quadrocopter. As the volunteers we need to find people interested in some sort of remotely controlled toys. It is important to gather both experienced and inexperienced ones. Regarding the inexperienced ones, it is useful to compare how difficult is for them to learn to fly with the particular devices. On the other side, the experienced ones can compare the user experience to the one they are accustomed to.

In the next two sections we will describe what flying scenarios pilots need to perform and what are the measured variables in those scenarios.

10.1 Scenarios

Although R-UAVs are usually used in open land flying high over the ground, for the purpose of testing we decided to fly indoors only at about altitude of 1-2 meters. That will prevent the quadrocopter to suffer damage in case of collision with an obstacle or in case of falling to the ground; moreover, it will also allow the capacity of the battery to be used effectively. Also, the inexperienced pilots will more clearly see the position and orientation of the drone, allowing them to closely focus on the control mechanism.

10.1.1 Poles

As the name of this scenario suggests, the main objects are two vertically standing poles. Both are 2 meters high and they stand 4 meters from each other, in the middle of this distance there is a starting point. It is a board in size of 1x1 meter on which the quadrocopter is placed in the beginning. The pilot looks at this scene from such a fixed point so that he can see both the poles and the starting point and those three things do not overlap in his view. Several photographs of this scenario are shown in Figure 10.1.

The task of the pilot is to take off and repeatedly circle the first and the second pole, making an “8” symbol trajectory. Whenever he flies over the starting point after circling the second pole, it counts as the end of the one round and the beginning of the other round. He must perform 5 rounds totally for each device, with the possibility to try it first.

The pilot is advised to use mainly the pitch and roll to perform manoeuvres. The yaw and gaz manipulations should be used only to correct heading or height of the quadrocopter.

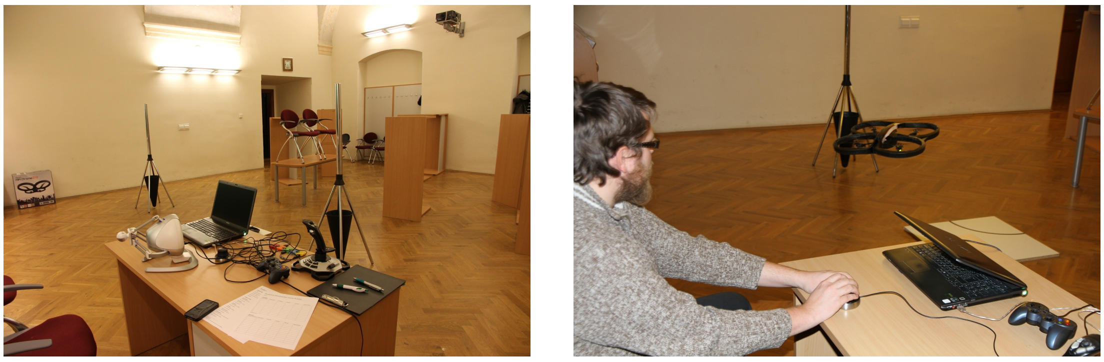


Figure 10.1: Poles
Pictures of the first scenario

10.1.2 Maze

This scenario is going to be more difficult than the previous one, as the pilot will be able to navigate only from the on-board camera of the quadrocopter. The pilot must take into account the delay of the camera picture and also the distortion caused by the camera lens.

As we can see in Figure 10.2 The ground plan of this scenario reminds a circuit about 15 meters long with an abundance of obstacles. They have many forms, some of them can be overcomed by flying through their center, others by flying over them, under them etc.

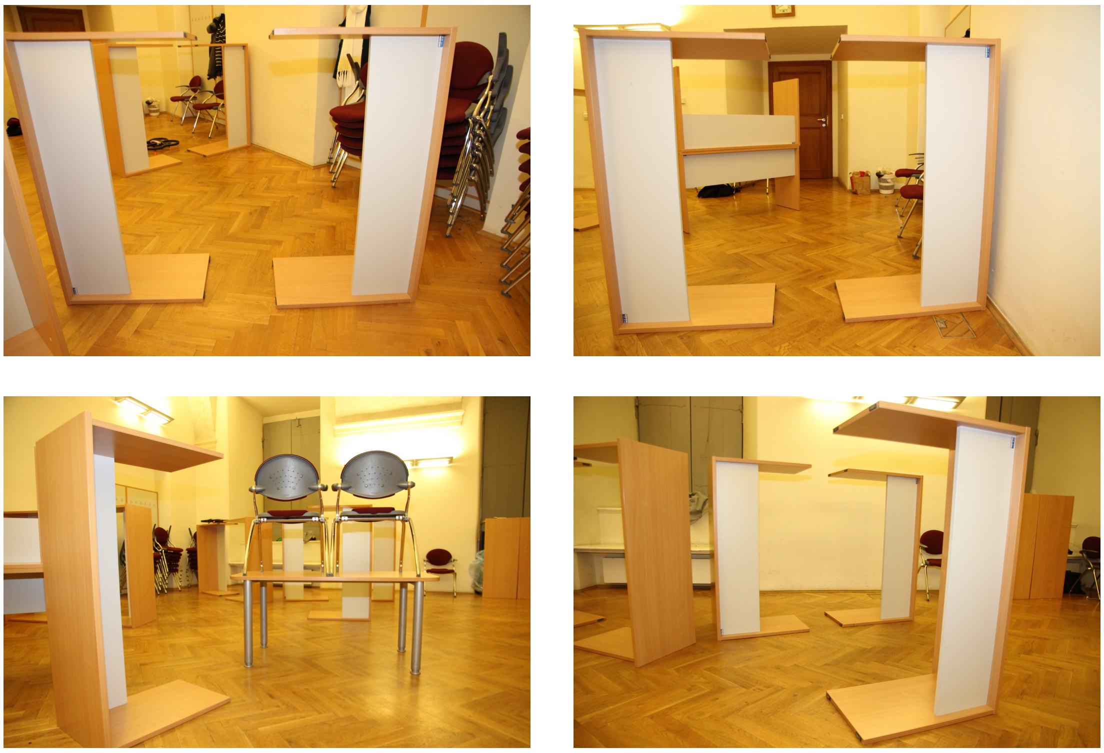


Figure 10.2: Maze
Walkthrough of the second scenario

The pilot must fly through the whole circuit. Every obstacle acts as a checkpoint, so that the pilot does not have to start from the beginning when he crashes.

For each device is enough to fly only one time; moreover, he does not have to use all the devices, but only the ones he selects. As we have only limited amount of time for every pilot, the maximum number of selected devices depends on how quickly he finishes the previous scenario.

10.2 Criteria

Criteria can be split into two categories: subjective and objective. We will start with the subjective ones. Probably the most important information about each device is the fact how intuitive it was for the pilots to use it. Therefore, they are asked to provide a rating in form of a number between 1 and 5 inclusively, 1 being the best and 5 being the worst. As this is the marking schema used in Czech educational system, all the pilots should be familiar with it and have similar measures. Along with the number they can optionally provide a written comment.

Although we do not have the RC transmitter in the list of control devices being tested, we added the option to provide rating for it. It is addressed to the pilots which have an experience with this device and are able to rate and comment it in the context of the other devices.

Regarding the objective criteria, we will measure three quantities: time needed to fullfil the task, number of collisions with obstacles and number of crashes. In the first scenario are those numbers counted for every round, in the second scenario for every checkpoint.

To the measured time we do not count the time needed to take off or to navigate to the starting position. As a collision we mean a situation when the quadrocopter hits an obstacle, but does not fall to the ground. If the quadrocopter falls to the ground or gets stucked in some position from which it needs to be rescued by hand, we count it as a crash. As the obstacles we count also walls, windows and other equipment of the room where the evaluation is taking place.

11. Results

Altogether, 19 volunteers attended the experiment, from which 11 had previous experience with remotely controlled quadrocopters or similar vehicles. The topic of this chapter presents results of the experiment, including subjective ratings of all the devices and objective ratings of flights from both scenarios. Most of the data presented are shown separately for three groups: all pilots, inexperienced pilots and experienced pilots.

11.1 Subjective rating

We obtained a number-rating of each device from every pilot and moreover, 8 experienced pilots also provided a rating for the RC transmitter. The majority of the pilots also gave text comments. Therefore, we can present a complete report of the subjective ratings for particular devices.

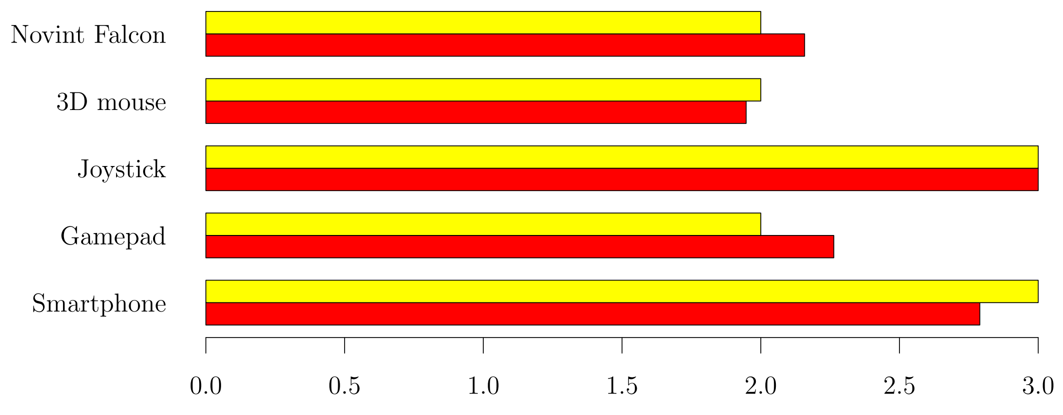
In Figure 11.2 we find the complete histogram of all the ratings provided by the pilots and means and medians for every device are summarized in Figure 11.1. As can be seen, the least popular is joystick, followed by smartphone. The remaining three devices have a similar average rating around 2, although their histograms differ. An interesting fact is that in the group of inexperienced pilots, 3D mouse and Novint Falcon clearly won. In the following paragraphs we will inspect each device more closely with regard to its marks and comments.

The *smartphone* was the second least popular device. This was mostly caused by bad ratings from inexperienced pilots, who did not find it very comfortable. Probably the most serious problem was the fact that the right stick is virtual, so the pilot does not feel to what extent it is shifted. Another complication appears when he must tilt the phone and simultaneously use the right stick: the display that must be touched is already in motion. On the other hand, the experienced pilots gave the smartphone much better ratings, because they became used to the mechanism relatively quickly.

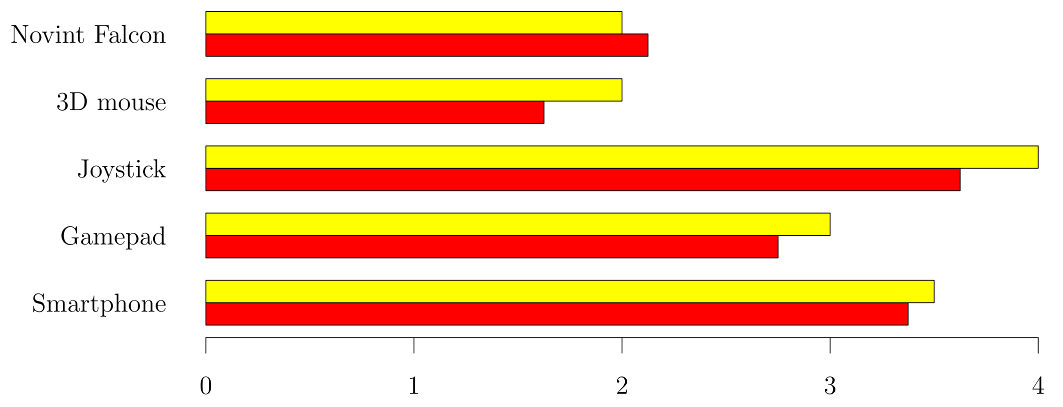
The *gamepad* was given an average rating. On one hand, the pilot can feel the rotation of both sticks. Also, the device is widely used for playing video games, so many pilots were accustomed to it. On the other hand, mostly for the beginners it was difficult to mentally distribute the control of one object into two hands, as we expected.

The *joystick* received the worst rating among all devices. Pitch, roll and yaw are controlled by one hand, which brings both advantages and disadvantages. Some pilots found it more intuitive than the previous controllers; however, that was outweighed by the fact that they were prone to alter the yaw by mistake. Moreover, as the gaz is altered by the throttle, it is problematic to find the position where the quadrocopter neither ascends nor descends.

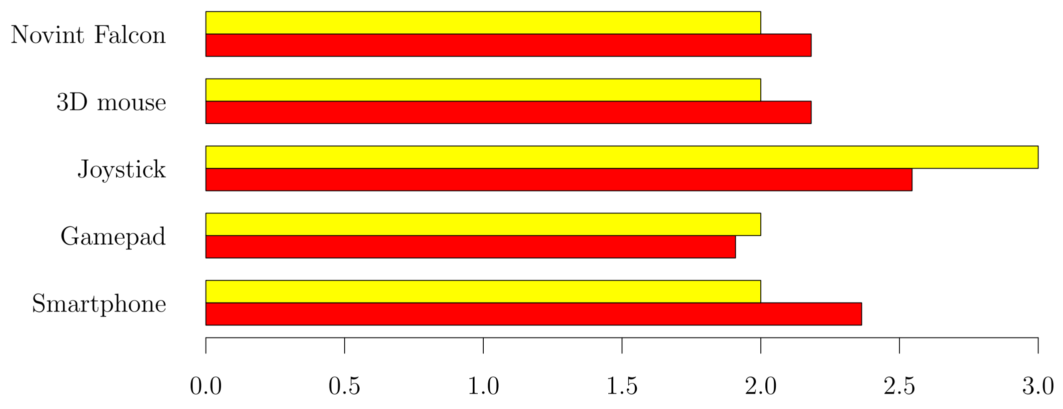
Although the control mechanism of the *3D mouse* is similar to the joystick, its rating was much more positive. From inexperienced pilots it received no worse mark than 2 and among the experienced there were only two pilots who gave lower marks. Supporters of this device mostly appreciate its intuitivity, sensitivity and the ability to combine several moves into one. When rotating the mouse, they performed shorter moves that with the joystick. The only downside was that



(a) All attendants

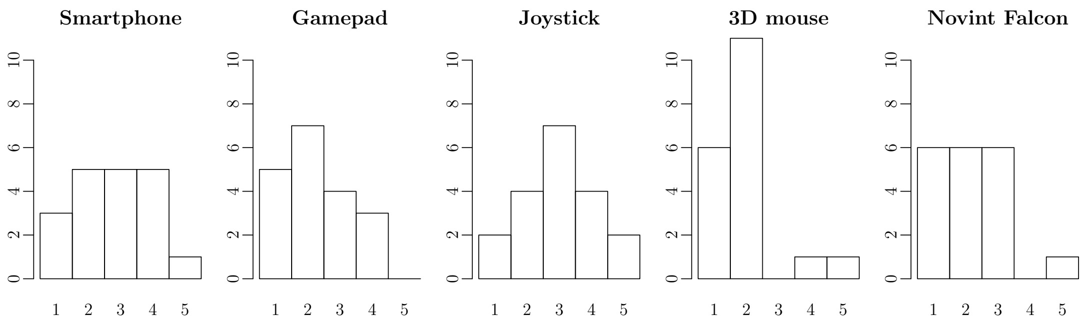


(b) Unexperienced attendants

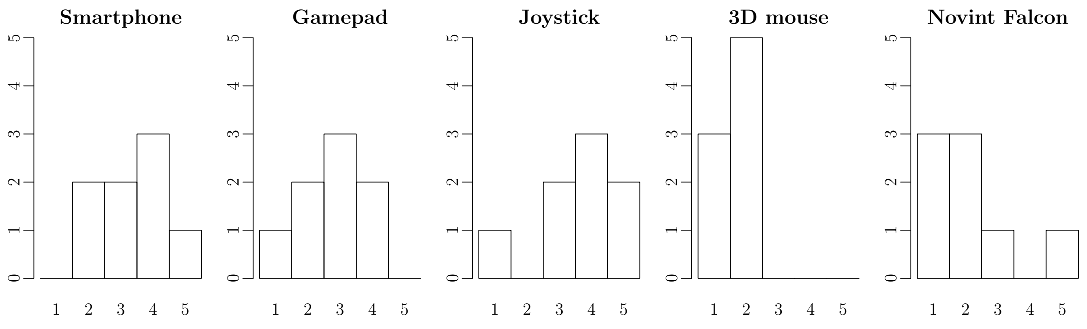


(c) Experienced attendants

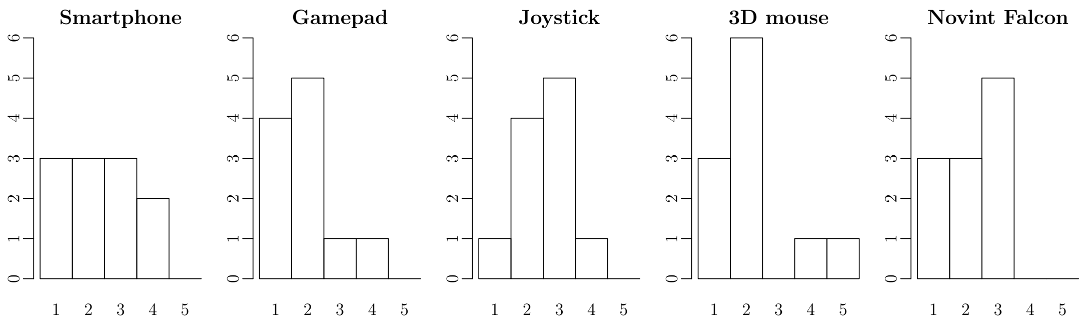
Figure 11.1: Subjective rating of particular control devices - mean and median
1 - best, 5 - worst



(a) All attendants



(b) Unexperienced attendants



(c) Experienced attendants

Figure 11.2: Subjective rating of particular control devices - histograms
1 - best, 5 - worst

some people altered the gaz by mistake. However, unlike with the joystick, this situation seldom occurred.

The *Novint Falcon* was rated as an average device among the experienced pilots, but the inexperienced rated it similarly to the 3D mouse. As they were not accustomed to a particular control mechanism, they accepted the Falcon, although it significantly differs from the other devices. Some pilots found it very intuitive; however, there were a few pilots who rated it too cumbersome.

The rating of the *RC controller* is merely orientational, as it was acquired by only a portion of attendants. However, it provided an insight to how experienced pilots see it compared with the tested devices. We observe that they rated it very high, its rating similar to the 3D mouse rating among the inexperienced pilots. That is caused by the fact that they are accustomed to it.

11.2 Poles

In this scenario, most of the pilots completed all five rounds. However, some of them refused to continue when they repeatedly crashed the quadrocopter after several rounds. Therefore, the total number of rounds performed differs slightly among the devices, in the range of 86 to 91. Because of that, we must keep in mind that all results should be normalized. We will present the rate of accidents and the time needed to complete all the laps.

Speaking of accidents, for every round we measured the count of collisions and crashes. As those quantities are closely related to each other, we decided to present them together. Because every crash is, in fact, an unsuccessful attempt to perform one round, for every device we defined the following variables:

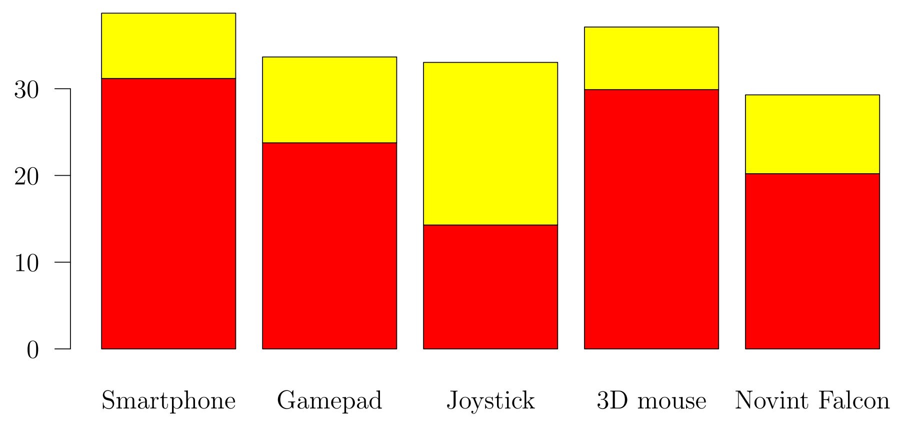
$$\text{total attempts} = \text{successful rounds} + \text{crashes}$$

$$\text{collision rate} = \frac{|\{x \in \text{rounds collisions}; x > 0\}|}{\text{total attempts}}$$

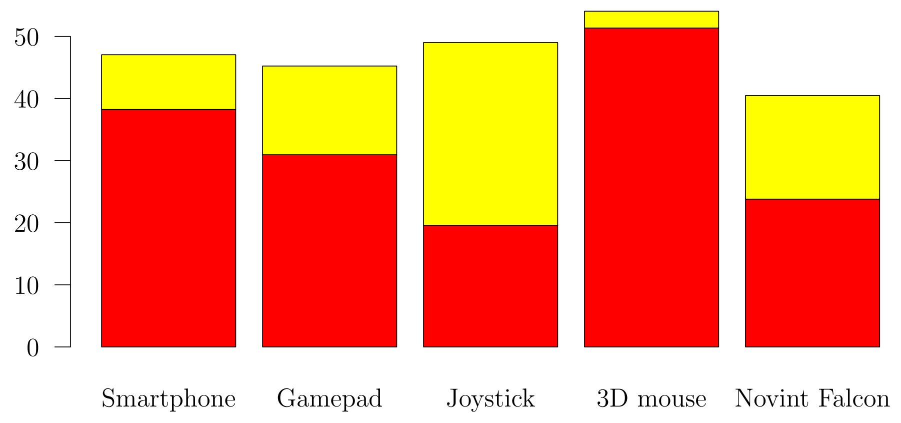
$$\text{crash rate} = \frac{\text{crashes}}{\text{total attempts}}$$

Collision rate and crash rate are depicted in Figure 11.3. As shown, the joystick had the highest crash rate among all devices, which corresponds to its poor subjective rating, mostly by inexperienced pilots. They also reached very interesting results with the 3D mouse. Although they collided with it in more than the half the cases, they mostly managed to stabilize the quadrocopter and avoid the crash. This phenomenon is similar to the Novint Falcon in the hands of experienced pilots.

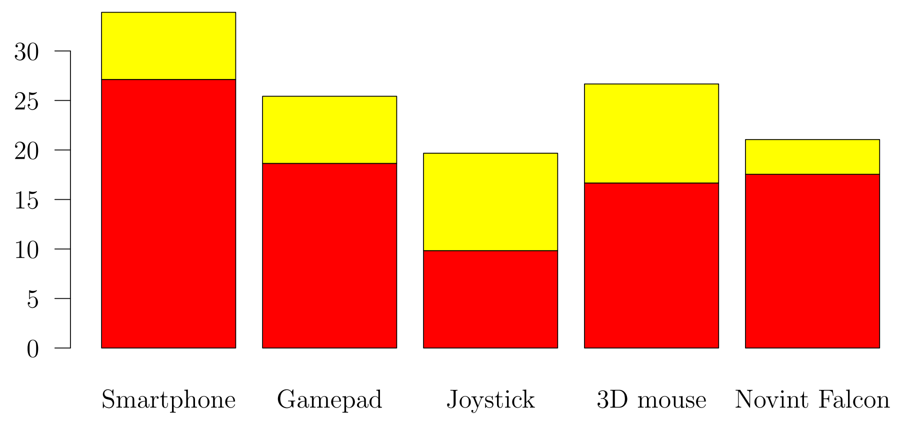
To present the measured round-times we decided to use a Beanplot [4], as it provides us information concerning their distribution. Results are presented in Figure 11.4. Again we can see, in the case of the joystick and smartphone, that several pilots had problems using them, which extended their round-times above 40 seconds. The time distributions of the remaining three devices are similar.



(a) All attendants

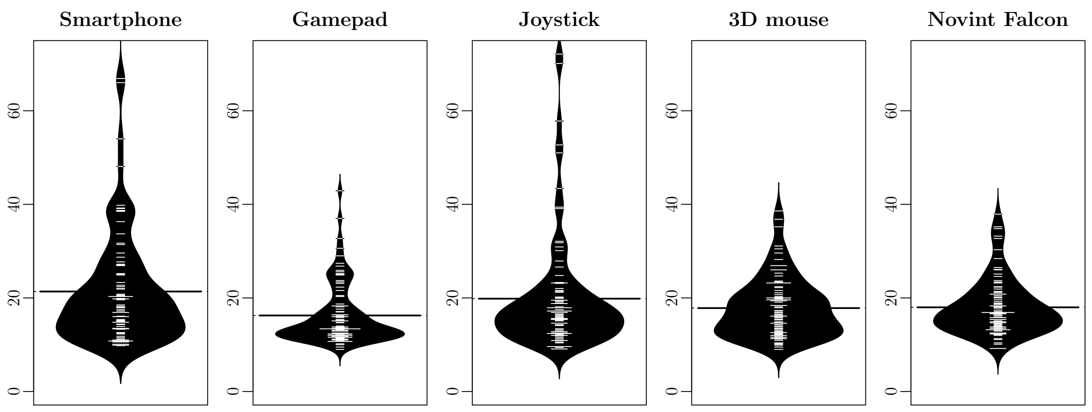


(b) Unexperienced attendants

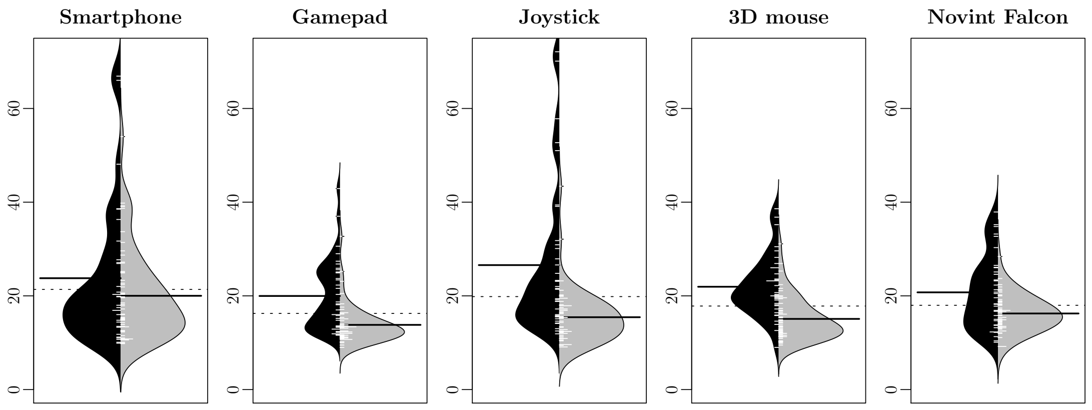


(c) Experienced attendants

Figure 11.3: Drone percentual accidents with particular devices



(a) All attendants



(b) Unexperienced and experienced attendants separated

Figure 11.4: Poles: Distribution of flight time per round in seconds

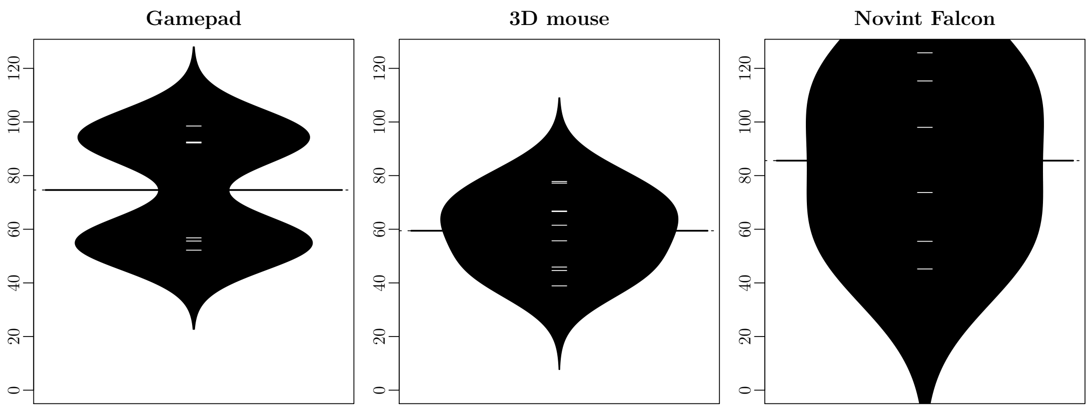


Figure 11.5: Maze: Distribution of flight time per round in seconds

11.3 Maze

Because the completion of this task was voluntary for the pilots, the amount of data obtained differs among the devices. Only 1 pilot decided to use the smartphone and 2 tried the joystick; therefore, we decided not to take those devices into account in this section. Both the gamepad and Novint Falcon were chosen by 6 attendants and the 3D mouse by 9. As there were 9 experienced pilots and only 3 inexperienced, we decided not to divide the results.

The average numbers of collisions and crashes per track were very small: 1.5 collisions and 0.5 crashes. Moreover, it did not vary by much, so it brings no new information to us.

The distributions of the time required to perform the track are shown in Figure 11.5. The best results were obviously achieved with the 3D mouse.

12. Discussion

The purpose of this chapter is to interpret the results of the experiment in order to provide the solution to the problem of the R-UAV remote human control.

Regarding experienced pilots, we learned that they are much less demanding when it comes to the quality of control devices. This is mainly caused by the fact, that they are already accustomed to use the 'two sticks' system. Therefore, they mostly have little motivation to search for other methods of control. Furthermore, special devices like Novint Falcon can seem unnatural for them.

On the other hand, inexperienced pilots have higher requirements for control devices. The 'two sticks' system is not sufficiently intuitive for them and they demand other means of control.

An analysis of all the devices follows, in which we inspect both their advantages and disadvantages. We begin with the *RC transmitter*. Although we did not test it directly in our evaluation, we obtained feedback from the experienced pilots. According to them it is very similar to the gamepad and mostly differs by its capability for use by two pairs of fingers and the greater accuracy of the sticks. The main benefit for the pilots lies in its versatility, as one RC transmitter can be used by several types of vehicles, including airplanes, cars and boats. However, this is probably also its biggest problem, because it is not created specifically for the type of the movement the R-UAV uses.

As we see from the results of the *gamepad*, the inexperienced pilots are not very satisfied with the control mechanism, where four flight parameters are distributed between two hands. Furthermore, it cannot be used separately, but only in combination with another device such as a PC or tablet. Its only advantage is the fact, that it is commonly used by video game players.

Another device with a control mechanism similar to the previous two is the *smartphone*. Pilots appreciate the use of its gyroscope in a manner that the R-UAV copies the tilt of the phone. However, they did not become familiar with the right virtual button, which was the source of numerous accidents. The popularity of the smartphone as an R-UAV control mechanism grows along with the popularity of the Ar.Drone. Regrettably, it does not bring any significant progress to the control mechanism when compared to the RC transmitter.

To video game controllers also belongs the *joystick*; moreover, it is used especially in flying simulators. Surprisingly, it appeared to be very cumbersome for the inexperienced pilots. This was caused by the inconvenience in yaw and gaz control mentioned before. Apparently, this method of combining three axes into one does not prove to be beneficial.

On the contrary, *3D mouse*, which combines all the four axes control in one hand, came out as a winner, at least for the inexperienced pilots. It proved to be the most intuitive method of control, because it feels like the pilot holds the R-UAV in his hand. From the collision and crash rates we can see that the users were able to avoid crashing it to the ground in case of collision with an obstacle.

The results of the *Novint Falcon* have shown that the experience with this device is unique. Not only does it look different, but it also provides the user a higher level of abstraction. When using the other devices, the users specify the pitch and roll in order to move on a horizontal plane, On this device, they simply specify the vector of the movement and let the R-UAV copy it. We must say that not all pilots appreciated this behavior, but it was generally rated as positive. Speaking of negatives, it lacks the fourth axis to control the yaw, as the two buttons proved not to be satisfactory.

In conclusion, we fulfilled the goal of the thesis, as the 3D mouse and the Novint Falcon have brought a new method of R-UAV control, which proved to be intuitive both for beginners and experienced users. We are aware of the fact, that the results from 19 people are statistically insignificant. However, this research may provide a starting point for other projects, either commercial or scientific.

12.1 Future Work

An interesting topic for a project would be to extend our application to also work on smartphones and tablets. If it proves technically feasible, it will allow people to control the Ar.Drone using the 3D mouse without the need of a computer. The structure of the application would enable developers to also add other types of drones, as well as control devices.

The next step would be to incorporate the 3D mouse directly into the RC transmitter, which would further extend the range of supported vehicles. That would be a rather ambitious task, as the developers would probably need to collaborate with a manufacturer.

As the Novint Falcon is dependent on a power supply, its use in practical applications will be limited. Therefore, in order to use its presented control mechanism, the authors would need to create a brand new device following this principle.

Bibliography

[1] Reg Austin. *Unmanned Aircraft Systems : UAV Design, Development and Deployment*. Wiley, 2010.

[2] Yu Takuya et al. Flexbot, 2014.

[3] Gary Fay. Derivation of the aerodynamic forces for the mesicopter simulation, 2001.

[4] Peter Kampstra. Beanplot: A boxplot alternative for visual comparison of distributions. *Journal of Statistical Software, Code Snippets*, 28(1):1–9, 2008.

[5] Tomáš Krajník, Vojtěch Vonásek, Daniel Fišer, and Jan Faigl. AR-Drone as a Platform for Robotic Research and Education. In *Research and Education in Robotics: EUROBOT 2011*, Heidelberg, 2011. Springer.

[6] Junichi Kunieda and Yukinobu Hoshino. Development of rc helicopter control skill study support system in consideration of user interface. In *Proceedings of the 18th International Conference on Fuzzy Systems*, FUZZ-IEEE'09, pages 957–962, Piscataway, NJ, USA, 2009. IEEE Press.

[7] Novint Technologies Incorporated. *Haptic Device Abstraction Layer (HDAL)*, 2008.

[8] Stephane Piskorski, Nicolas Brulez, Pierre Eline, and Frederic D'Haeyer. *Ar.Drone Developer Guide*. Parrot, SDK 2.0 edition, 12 2012.

[9] John Salt. Rc helicopter material - plastic, aluminum, carbon fiber, 2008.

[10] USB Implementers' Forum. *Device Class Definition for Human Interface Devices (HID)*, 2001.

[11] Kimon P. Valavanis, editor. *Advances in Unmanned Aerial Vehicles*. Intelligent Systems, Control and Automation: Science and Engineering. Springer Netherlands, 2007.

[12] Adrian Weber, Bernhard Jenny, Matthias Wanner, Juliane Cron, Philipp Marty, and Lorenz Hurni. Cartography meets gaming: Navigating globes, block diagrams and 2d maps with gamepads and joysticks. *Cartographic Journal*, 47(1):92 – 100, 2010.

Attachments

CD Content

In the table below the top level folder structure of the enclosed CD is summarized.

Folder	Contents
`bin`	Executable file of the application
`cmake`	CMake scripts to create Visual Studio projects from the sources
`doc`	Text of this thesis, user manual and Doxygen-generated technical documentation
`evaluation`	Complete results of the evaluation, along with interesting videos and images
`sources`	Source code of the application