Drone Piloting Study

Christine Graff

Advisor
Alexander Förster

Tutors
Jürgen Leitner

Abstract

A usability study to explore the naturalness of piloting the Parrot AR.Drone, using input devices designed to manipulate 3D graphics.
Contents

1 Introduction 1

2 State of the Art 2

3 Drone 3
   3.1 The SDK ...................................................... 6
   3.2 AR.Freeflight ............................................ 6

4 The Leap Motion Device 7
   4.1 The SDK ..................................................... 8

5 3Dconnexion SpaceNavigator 10
   5.1 The SDK ................................................... 11

6 Interface Development 12
   6.1 Leap Interface ............................................ 12
   6.2 3dConnexion SpaceNavigator Interface ............... 16

7 Usability Study 17

8 Results 20

9 Conclusion 28

A Usability Study Documents 30
   A.1 Consent Form ............................................. 30
   A.2 Questionnaire ........................................... 31
1 Introduction

The term "drone" refers to an aircraft which can be piloted without a human pilot aboard, either remotely or autonomously. Drones can also be referred to as unmanned aerial vehicles (UAVs) or remotely piloted aircraft (RPAs). Although the general public tends to think of drones as a deadly tools used by the US military, drones can benefit society in many other ways. They can be used for search and rescue in dangerous conditions, crop surveillance, aerial footage in films, sports photography, inspecting power lines and pipelines, monitoring wildlife to help conservation efforts, delivering supplies to inaccessible areas, prevention and early detection of forest fires etc. The drone’s capability to perform tasks too "dull, dirty or dangerous" for human-piloted aerial vehicles is what makes it such a popular field of research for contemporary engineers and computer scientists [1]. Current research, which aims to improve drone autonomy, structural capabilities or ease of remote piloting, is very likely to have a significant impact in the years to come.

US military drones are typically controlled via joystick, while the pilot sits at a computer and pilots the drone remotely, using its live video feed, as seen in Figure 1 [2]. The joystick originated as a device for piloting aircraft [18] and was later adapted to manipulate computer graphics in video games. Recreational drones, on the other hand, are more likely to be controlled via tablet, smartphone app, or wireless gamepad. In this way, the recreationals such as the one depicted in 2, are to be able to fly the drone outside without the burden of carrying around a heavy PC and joystick. However, it is possible to pilot drones with a wide variety of other input devices.

Figure 1: Piloting a drone with a joystick [2]
We live in an exciting time, in which computer graphics and input device technologies are rapidly improving. Meanwhile, we are seeing robotics take an ever more prominent role in our society. Devices like drones, which are capable of a certain degree of autonomy are being recognized for their enormous potential to improve the future. Human-computer interaction has never been a more relevant field than it is now. Finding the best interface for a given application, and the right input device to allow humans to control a given electronic system, has become more important than ever before. It seems that devices such as joysticks, gamepads and touch-screens can be used to control both 3D graphics and robots interchangeably. Can we therefore conclude that the optimal interface is no different whether one is moving 3D objects on the screen or in the real world? Input devices such as the Kinect, Leap Motion device, 3dConnexion SpaceNavigator etc. have been designed in recent years with the goal of creating interfaces for video games, which are more natural and enjoyable for humans to use than a traditional gamepad. It would be interesting to see if the benefits of these devices also apply when they are used to control robots rather than computer graphics. The aim of this project is to examine the question more closely. We do this by conducting usability testing on drone-piloting interfaces, which use input devices (namely the 3dConnexion and Leap Motion devices) designed for 3D computer graphics, and examining user feedback.

2 State of the Art

Since the classic mouse emerged in the 1970s [30], we have seen a great deal of innovation with respect to devices for human-computer interaction, particularly for manipulation of 3D graphics in gaming and computer aided design. Several relatively new devices for remote manipulation of 3D graphics, such as the Wii, Kinect, Playstation Move and Leap Motion have achieved widespread popularity. The Wii Remote Controller emerged in 2006 for the Nintendo Wii game console [31]. It uses an ADXL330 accelerometer to sense acceleration along 3 axes, and uses an optical sensor to determine where the Wii controller is pointing [32]. The Kinect was released in 2011 for Xbox360. It tracks movement
through the use of an infrared projector combined with a monochrome CMO sensor and camera, and performs image-based 3D reconstruction [33]. The Leap Motion device entered the market in 2012. It uses LEDs to create a 3D light pattern, which is then compared to the frames generated by two monochromatic IR cameras [22]. The definition of HID specification for USB in the late 1990s has also allowed for increased input device innovation, because a wider category of HID devices are compatible with PCs, without the need to include special drivers [34].

The emergence of these innovative input devices has also sparked a wave of third party development, which often uses the devices for tasks completely different to what they were originally designed for. For example, Ph.D. candidate, Philipp Robbel programmed an interface, using the Kinect, to control an iRobot Create (a small programmable robot, similar to a Rumba). The Kinect is attached to the robot (see Figure 3), and generates a 3-D map of the surrounding room. It can detect humans and respond to gesture and voice commands [35].

![Figure 3: The "KinectBot". [16]](image)

The aim of this project is not merely to control a drone using the Leap Motion and 3dConnexion devices, as this has been done before [36]. It is to explore the level of naturalness which these devices provide the user when they are used for something they have not been designed to do: pilot a drone. More specifically, we want to examine the degree of naturalness by performing usability testing as part of the process of iterative interface development.

3 Drone

The Parrot AR.Drone 2.0, used for this project, is a radio controlled quadrotor constructed nylon and carbon fiber parts, which measures 57cm across [19]. A quadrotor (or quadcopter) is a helicopter with four rotors. Two of the propellers are pitched clockwise and the other two counterclockwise. The relative
speed of these rotors determines the lift and torque of the quadrotor. When
the RPM is equal for all rotors, the quadrotor will hover or change altitude. To
turn (change yaw), the quadcopter increases the speed of the rotors spinning in
the direction it wants to turn. To move forward or sideways (change pitch or
roll), the quadcopter decreases the speed of the rotor located in the position of
the direction it wants to tilt, and increases the speed of the diametrically oppo-
site rotor. When UAV quadrotors like the AR.Parrot Drone achieve stability in
mid-air through the use of an electronic control system and electronic sensors.[4]
Their compact size and speed make quadrotors ideal for flying indoors as well
as outdoors. They can be quite cheap and easy to maintain. This makes them
a popular choice for university research projects [20].

Figure 4: The four rotors of a quadrotor. Rotors 1 and 3 rotate clockwise, while
rotors 2 and 4 rotate counterclockwise [5]

The AR.Drone 2.0 was launched in 2012. It is designed to communicate with
a client device via self-generated wifi hotspot. Like a typical quadrotor, the
AR.Drone has six degrees of freedom. It can move up and down, forwards and
backwards, left and right, tilt forward and backward (pitch), turn left and right
(yaw) and tilt side to side (roll). The onboard computer responsible for rotor
speed control, stabilization, command processing etc. runs a Linux operating
system. The drone is powered with brushless engines with three phases current
controlled by a micro-controller. If any of the propellers are blocked, the drone
can detect it and responds by stopping all engines at once to prevent damage
[17]. Other onboard features include: an ultrasonic altimeter used for vertical
stabilization, air pressure sensor for more stable hovering, an 11.1 volt lithium
polymer battery, a front camera with 90 degree lens which shoots in 720p HD
at 30 fps, vertical camera featuring QVGA sensor recording up to 60fps, 4GB
of storage, GPS tracking and navigation, and a detachable enclosing foam hull which protects the body and rotors in the event of a crash [21].

A client device can control the drone by sending it AT commands to UDP port 5556 at regular intervals (preferably 30x per second for smooth drone movements). If the space between two consecutive commands is more than 2 seconds, the drone will consider the WIFI connection lost. The client can also receive information about the drone’s status, position, speed, engine rotation speed etc. on port 5554. This information is called ”navdata” and is sent about 30x per second. In addition, a video stream from the drone’s camera is sent from drone to client on port 5555. AT commands are text strings which can be sent directly to the drone with UDP packets on port 5556. The drone will simply ignore invalid AT commands. The structure of an AT command consists of three characters ”AT*” followed by the type of command, equals sign, sequence number and finally a list of comma-separated arguments. For example: ”AT*PCMD=2165,1,0,0,0,0”. The length of a command must be less than 1024 characters according to the drone software. The first number after the equals sign is the sequence number. The drone only executes commands whose sequence number is higher than the last valid command sequence number. A client should always send 1 as the first sequence number and always send subsequent commands with increasing sequence numbers, in order for the commands to be correctly processed by the drone [21].
3.1 The SDK

Third party developers must visit Ardrone.org and accept the SDK licence agreement terms and conditions in order to download the SDK. The SDK includes: The AR.Drone developer guide, the AR.Drone Library (ARDroneLIB), the AR.Drone Tool (ARDroneTool) library, the AR.Drone Control Engine library, an open-source iphone game example and several code examples for windows, linux and android. The SDK allows third party developers to write their own applications to remotely control the drone, but it is not possible to overwrite embedded software to directly access the drone’s hardware [21].

3.2 AR.Freeflight

The AR.Drone can be controlled by iOS or Android operating systems with apps on a mobile or tablet. The official controller app released by Parrot for the AR.Drone is the AR.Freeflight app, available for free download for android and iOS. AR.Freeflight connects with the drone via wifi. It includes live video stream from the drone’s camera, and allows the user to record high definition video or take snapshots as the drone flies.
Figure 7 shows the Freeflight app’s control screen. The background shows the live video stream recorded by the drone’s camera. The circle on the left is the accelerometer, which the user can hold down to increase speed. The circle on the right controls the drone’s direction of flight (up and down arrows control vertical direction, and left and right arrows turn the drone horizontally to the left or right). In the app’s default mode, shown in Figure 7, the user moves the drone forwards, backwards, left or right by tilting the phone in the desired direction. However, the app’s piloting mode can be changed to allow the user to move the drone using the touch screen rather than phone tilt.

4 The Leap Motion Device

The Leap Motion device (depicted in Figure 8) was developed by Leap Motion Inc., a company based in San Francisco, and was first released in 2012. It is a USB peripheral sensor device, about 3 inches long, which can be placed on a flat surface such as a table or desk top. Similar to the Kinect, it is designed to track the user’s hand and finger motions without requiring any physical contact. To accomplish this, it uses two monochromatic IR cameras and three infrared LEDs to generate a 3D IR light pattern. The leap motion field of view is an
inverted pyramid centered on the device. Its area of detection spans a distance of about 25 to 600mm above the device (1 inch to 2 feet), and data is sent through the USB cable at a rate of about 300 fps. The Leap Motion controller software then compares the 2D frame data in order to generate 3D positioning information [22].

Figure 8: The Leap Motion device [10]

4.1 The SDK

The Leap Developer Kit can be downloaded for free at developer.leapmotion.com. It includes the Leap SDK, as well as example code for some simple applications in C++ with OpenGL. The Leap Motion library is written in C++, but uses SWIG, an open source tool, to generate language bindings for several other programming languages. The SDK provides APIs for C++, Java, JavaScript, ObjectiveC and Python. Information about the position of hands, fingers, and small "tools" are detected and interpreted by the Leap Motion controller software. It detects not only the discrete positions of these objects, but can also track motion and recognize certain gestures. Distance is measured in millimeters, time in microseconds, speed in millimeters per second, and angle in radians. The device utilizes a right-handed coordinate system. The origin is at the center of the device, the y axis extends vertically, and the x and z axes lie on the horizontal plane [23].
The Frame class is the root class of the Leap Motion API. A Frame object represents a package of data sent by the device, which describes position data it detected and interpreted during a certain time interval: a snapshot of the scene. It contains a list of fingers, hands and tools detected in that particular frame, as well as a list of gestures. The objects in these lists contain more detailed information about the construct they represent. A Hand object contains information about palm position, movement and other characteristics of hands detected in the frame. The Leap Motion device is able to detect more than two hands, however, given the limited scope of detection, it is recommended to keep at most two hands in the frame at a time. Pointable objects represent fingers and tools. Objects such as pencils, pens, and straws are detected as tools rather than fingers because they are thinner and straighter. A finger type Pointable object contains information such as the finger tip position and direction the finger is pointing (see Figure 10) [23].

The gestures recognized by the Leap Motion device and stored in the list of Gesture objects for each frame are: circle, swipe, screen tap and key tap. The gestures are detected by tracking patterns in the motion of a finger over several frames. A circle gesture is detected if there is a finger tracing a clockwise or counterclockwise circle, a swipe detects a long horizontal movement of a finger, key tap is a tapping movement, mimicking the press of an actual keyboard key.
and screen tap is a tapping movement mimicking the user tapping a vertical screen. Gestures are tracked for each finger independently [23].

5 3Dconnexion SpaceNavigator

The 3Dconnexion SpaceNavigator (see Figure 11) is a 3D mouse, designed to facilitate human interaction with 3D computer graphics. It has six degrees of freedom, and allows users to pan, zoom, and rotate. This small, yet unexpectedly heavy device measures 3.1 inches by 3.1 inches horizontally, and 2.1 inches in height, and weighs 1.06 lbs. The SpaceNavigator also includes two buttons and LED lights, which can be turned on or off. It requires Microsoft Windows, OS X or Linux operating systems, and connects with a personal computer via USB port. The pressure sensitive handle allows users to move a virtual object smoothly and fluidly in several directions simultaneously in 3D space (see Figure 12). Alternatively, the SpaceNavigator can emulate the actions of a 2D mouse as well. Its settings can be changed to allow its buttons to act like the left and right click of a 2D mouse and for the movements of the handle to move the mouse cursor [24].

![Figure 11: 3Dconnexion SpaceNavigator](image)

![Figure 12: Navigation](image)
5.1 The SDK

Users must register at 3dconnexion.com in order to be able to download the necessary drivers for the 3Dconnexion SpaceNavigator. Along with the drivers, the user can also download several free applications for the SpaceNavigator, including a training application, settings, and several entertaining games. These applications can be accessed from the 3D Mouse Home window, shown in Figure 13.

![3D Mouse Home](image)

Figure 13: 3dConnexion Home [14]

The API I decided to use to program with the device is Jinput. The necessary jars and plugins for this API can be downloaded at java.net/projects/jinput. Jinput is a library which allows developers to write java code to communicate with input devices such as mouse, keyboard, gamepad, 3D Mouse etc. It is possible to access and distinguish between several devices at once using jinput. The ControllerEnvironment class contains a list of controllers or devices (keyboard, joystick etc.). By continuously iterating through this list and checking the name or ID of each controller, the desired controller can be found. In this case, the name of the desired device is simply "SpaceNavigator". Each controller has a list of components, buttons and axes. Components can also be distinguished by their names. For example, the name of the z axis (describing forwards/backwards movement of the handle) of the SpaceNavigator is predictably, "z". The components have values associated with them which change when the state of the component changes. Components should be continuously polled to check for updates in order to ensure fluid movement of the model [25].
6 Interface Development

This section describes the interface development process to create interfaces for users to control the drone with the SpaceNavigator and Leap device respectively. For each device, the stages of development are as follows: 1) Completion of the necessary installations. 2) Familiarization with the device’s associated API. 3) Brainstorming potential gestures for device-drone communication. 4) The design of a language of gestures, using the Wizard of Oz technique [27]. 5) Debugging with 3D graphics. 6) Integration with the drone.

6.1 Leap Interface

The SDK for the Leap motion device can be downloaded for free at leapmotion.com, and the included Leap API is available for several different programming languages. Both the Leap and SpaceNavigator interfaces for this project are developed using the Java programming language. Its class-based, object-oriented structure makes Java a good fit for an application where devices are receiving input data and communicating with one another via separate threads.

Phase 2) of the development process took about one week. Tutorials can be found at http://developer.leapmotion.documentation.java/devguide.html, which developers can use to familiarize themselves with the Leap Motion API. Phase 3) is a brainstorming phase. The developer should be familiar with the gestures and motions recognized by the Leap device, and can now think of as many gestures as possible which can feasibly be used as catalysts for sending commands to the drone. The takeoff and land commands are each only sent once, so they can be triggered by static gestures. If the gesture is detected, the command is sent once, and it does not matter how long the gesture is held. The possible gestures for takeoff/landing, determined during phase 3) of this project are: fist, screen tap, key tap, peace sign (two fingers), one finger, three fingers and hand enters the frame. The commands for the drone to move up, down, forwards, backward, left, right and turn are more complex. The corresponding AT commands for these actions must be sent continually, every 30 ms, for the drone to continue moving in the desired direction. Thus, it is important for the user to be able to vary the duration of time that the command is held and the speed of the drone in some way.
To control the drone’s vertical motion, phase 3) produced only one option: moving the flat palm up and down along the y-axis in the Leap device’s frame of view. The Leap device is able to detect a hand’s palm position (see Figure 14). The Leap Origin is located in the center of the device’s body, but it can be translated to some point on the y plane above the device, so that the user can control speed by increasing the displacement between palm and origin in either the positive (up) or negative (down) y direction. Keeping the hand at the origin causes the drone to hover. To control the drone’s horizontal motion (forwards, backwards, left and right) phase 3) produced two possible gesture categories. The first category contains gestures which move the drone horizontally based on detection of hand palm-position displacement, while the second category moves the drone according to detected displacement of the angle of the palm normal. Palm position displacement from the origin is used here in the same way as for vertical motion. If the hand moves forwards or backwards along the Leap’s z axis, away from the origin, the drone moves forwards or backwards respectively. Again, speed depends on the displacement between the hand and the origin, with 0 displacement resulting in no horizontal movement at all. For left and right horizontal movement, exactly the same concept applies, except the hand moves along the Leap’s x axis. In this horizontal motion model, the hand emulates the direction it wants the drone to move in. The alternative possibility is to use the hand to emulate the manner in which the drone moves rather than the desired direction. When the drone moves horizontally, it tilts in the direction it wants to move, with speed determined by the angle of tilt. A user can control the drone’s horizontal motion with the Leap device by tilting their palm in direction they want the drone to move, with the angle of palm tilt controlling the speed of motion. The Leap API records the roll, pitch and yaw of a palm normal in radians. Yaw is the tilt around the y axis, roll is the tilt about the z axis and pitch is the tilt about the x axis (see Figure 15). Following this gesture model, palm pitch moves the drone forwards and backwards, while palm roll moves the drone left and right. Palm yaw can be used to turn the drone left or right in the x plane, around the y axis. An alternative for turning for the drone would be to use the circle gesture. The Leap device detects a circle when a finger traces a circle in the Leap’s field of view. It can differentiate between clockwise and counterclockwise circles, and also detect the speed of the finger tracing the circle. Duration can be achieved by continuing to make a circle for
a certain amount of time, and speed of the drone can be determined by tracing the circle faster or slower (see Figure 16).

Figure 15: roll, pitch and yaw [15]

Figure 16: circle gesture [11]

It is important to involve users in the development process (user-based development) from the beginning, in order to avoid the trap of developing a language of gestures based on what the device detects best (system based development) or the developer’s own opinion. This is because the goal of designing a human-computer interaction interface is not only to create an interface which works effectively, but also to maximize the user’s perceived ease and enjoyment of use. In order to decide which gestures to ultimately use in my interface I employed the Wizard Of Oz technique. This is a technique in which between 3-5 users are typically observed individually performing the actions or gestures associated with an input device, while the ”wizard” (in this case me) mimicks the effects of that particular action in some way. The output must be simple enough that the ”wizard” can simulate it in real time. This technique can be useful for obtaining information about the user’s interaction with the interface, testing which actions best serve the interaction, discovering problems etc. [27]. I asked each user to sit at a table with the Leap motion device and perform each of the actions I had brainstormed. While they did this, I mimicked the corresponding intended action of the drone, using a small circular piece of cardboard as a drone model. I then asked the users which of the actions they found most intuitive
for each corresponding drone movement. For vertical motion there was only
one choice: vertical palm movement for the Leap device. For horizontal drone
motion, users preferred the tilting palm over horizontal palm displacement. To
turn the drone they preferred using yaw palm tilt rather than the circle gesture.

After creating a language of gestures communicate with the drone, we can test
its effectiveness by using it to manipulating some 3D computer graphics object,
which simulate the drone’s movements. This is a more effective way to debug
than testing the language on the drone immediately. The AR.Drone automatically stabilizes itself in the air during hovering and other movements to keep
from crashing. This causes it to sometimes make small sporadic movements
which are unrelated to any AT commands sent by the code. This can make
debugging a tricky and imprecise process due to feedback uncertainty. When
testing newly written code on the drone directly, there is also the danger of
losing control of the drone or crashing it if the code contains errors.

In this project, we use a 3D OpenGL cube for phase 5) of the interface develop-
ment. The Java OpenGL (JOGL) library is used to render the cube and display
it in a Swing window. JOGL is a wrapper library for OpenGL C code. It works
by accessing the OpenGL C API, using the Java Native Interface (JNI). A cube
is one of the simplest 3D shapes to create in OpenGL and is also useful to sim-
ulate the drone’s movements. If we color each of the cube’s six sides a different
color, the drone’s six degrees of freedom can be visualized clearly by the cube’s
movements.

The cube is newly rendered in each frame. When the Leap device detects a rele-
vant gesture in our interface language, this causes the coordinates of the central
point of the cube to be updated in such a way that the cube seems to be moving
in the correct direction. This is achieved by running separate threads in the
Leap’s frame listener class and the cube render class respectively. The static
Cube class keeps track of the cube’s current x, y and z coordinates (representing
the point at the very center of the cube) and angle of rotation about the y axis
(the only kind of rotation that the drone is capable of). The Cube object is
passed as a parameter to Leap’s FrameListener class upon creation. When the
listener detects a relevant gesture, it updates the Cube object accordingly. The
Render class then reads the coordinates of the Cube object to redraw the cube in
the correct location for each frame. For example, if the listener reveals a positive
vertical displacement from origin to palm position for the current frame, it will
scale it and add it to the current y position of the cube. While the users palm
position remains above our chosen origin in the leap field of view, the cube will
continue to move upwards. It will move faster or slower depending on the size
of the displacement between the palm and the origin, mimicking the movement
we wish the drone to perform. The cube is a useful tool for debugging our Leap
interface, because it is a good way to visualize whether the interface is able to
detect the correct gestures and influence an object’s movement in (virtual) 3D
space accordingly.

The final phase of development is the integration of our device interface with the AR.Drone. The SDK for the Parrot AR.Drone can be downloaded from projects.ardrone.org. Developers who are programming with the drone for the first time can read the included developer guide to familiarize themselves. There is also an API, but it is not used for this project. Instead, Java socket programming is used to send AT commands directly to the drone.

### 6.2 3dConnexion SpaceNavigator Interface

The phases of development are the same for the SpaceNavigator interface as for the Leap interface. We use Jinput, an open source API for game controller discovery and polled input. This API is not specific to the SpaceNavigator. It detects all input devices connected to your computer at a given time.

After the brainstorming phase, the list of potential SpaceNavigator actions for each drone command was much shorter than the equivalent list for the Leap device. The SpaceNavigator includes two small buttons on the left and right sides of its base. The most logical way to control takeoff and landing is with these buttons. One button could be used for takeoff, and the other for landing, or either button could be used for takeoff. Once the drone has taken off, if one of the buttons is pressed again, it will land. For vertical motion there is also only one intuitive option: pulling the SpaceNavigator handle upwards for the drone to move upwards and pushing it downwards for the drone to move down. To turn the drone left or right it also seemed that the only logical choice is turning the SpaceNavigator handle left or right respectively. For horizontal motion, there is once again a choice between emulating the desired drone movement or the drone itself. The SpaceNavigator handle can be pushed forwards and backwards along the z axis for the drone to move forwards or backwards, and pushed right and left along the x axis for the drone to move right or left. Alternatively the handle can be tilted in direction which the user desires the drone to tilt (which is also the direction the drone will move in). Unlike the Leap device, no padding is needed for the SpaceNavigator to allow users to achieve drone hover. Once the user lets go of the handle it automatically springs back to the origin. Thus, the drone can hover when the SpaceNavigator handle is at the origin i.e. not receiving any input from the user.

We use the *Wizard of Oz* technique to decide which gestures to use in the SpaceNavigator interface. For vertical motion there was again only one viable option: pulling the handle upwards or pushing it downwards. For horizontal movement, the study demonstrated that users prefer tilt over horizontal displacement, finding it more intuitive to tilt the handle forwards, backwards, left
or right in order to move the drone in those directions. To turn the drone, there is only one possibility: turning the handle left or right.

Phases 5) and 6) of our interface development plan (debugging with the 3D graphics cube and moving the drone) were much easier for the SpaceNavigator than for the Leap device, because it is possible to simply reuse the Cube, Render and ARDrone classes from the Leap interface.

7 Usability Study

Usability testing is a tool used in user-centered, iterative interface design which allows developers to create interfaces which better fit the needs of the target user. Typically, a number of test subjects from the pool of potential users are selected to take part in a controlled experiment in which they are asked to complete several tasks using the interface. The developer observes the user’s interaction with the interface and takes notes, collects feedback directly from the user in the form of a questionnaire, or asks qualitative questions. The information gathered from these studies can be used to identify problems, receive suggestions or new ideas, and to discover which parts of the design to maintain as opposed to which parts need to be removed or improved upon [28].

In this project, we use "In-Person" usability testing, a model in which each testing session includes one-on-one moderation from an expert while the user performs tasks with the interface. Given the high likelihood that inexperienced users will crash the drone and become worried or startled, it is better to use in-person usability testing in this case rather than, for example, remote usability testing. It is typical to observe between 5 and 10 test subjects, usually between 1 and 2 days. According to research, 85% of usability problems can be identified after testing with just five users [29].

In this case, the target user are males and females of all ages (excluding, of course, infants and toddlers). Anybody can buy and pilot a recreational drone for fun. Although they are typically more popular amongst tech enthusiasts. There are several factors which could affect the user’s performance using the interfaces described in the previous section. These factors include: the user’s age, experience flying a drone, playing video games, using input devices and controlling toys such as RC cars or planes. Prior experience with drones, video games, motion sensor devices for gaming and RC cars or planes could also make it easier for a user to learn to use my interface effectively since these interfaces involve the control of 3D graphics or objects with input devices. Making these distinction between user groups can help to better determine the reason for certain kinds of user feedback resulting from the study. In order to distinguish between user groups we provide a questionnaire during the study to gather
relevant information about each user, although the user him/herself remains anonymous. To view the questionnaire, visit Appendix A, section A.2.

The questionnaire also contained one open question, asking the user to describe their experience flying the drone with the SpaceNavigator and Leap device, in terms of ease, naturalness, intuitiveness and level of enjoyment. Although it is not possible to create statistics from qualitative feedback such as the user’s feelings of enjoyment when using the interface, this kind of information is also relevant for user-interface improvement. In cases such as this, where the interface is being designed primarily for recreational use, developers should strive to maximize not only naturalness and ease-of use, but also the user’s feeling of “fun” while using the interface. Including an open question also allows users to bring up any issues which were not directly asked about or recorded by the observer.

The location of the usability testing was the Aula Magna of the Università della Svizzera Italiana. It is an optimal location because it is indoors, and thus impervious to variable weather conditions, while also large and spacious enough to accommodate several simple drone piloting tasks. All of the subjects were USI students who happened to pass by the Aula Magna on the day of the testing and agreed to participate. Each user must first sign a consent form (located in Appendix A, section A.1) in which they acknowledge that they are aware of the nature of the study and are participating voluntarily. The user first has 5 minutes to familiarize himself with the drone by piloting it with the AR.Freeflight app. After this short period of freeflying, the controlled experimentations begins. Figure 17 depicts the setup of the experiment space. The user is asked to complete two simple tasks with each interface. Task 1: Take off from location A, fly the drone through an opening suspended off of the ground between two poles, and land in location B. Task 2: Take off from location B, fly the drone through the same opening in the reverse direction, and land in location A.
Figure 17: Setup of the experiment space. Task 1 is depicted with the blue arrow and Task 2 with the red arrow.

The tasks were completed by each user, first using the AR.Freeflight app, then the SpaceNavigator-drone interface, and finally the Leap-drone interface. Figure 18 shows a picture taken during the study of a user performing Task 1, using the Leap interface. When a user completed a task, the completion time (timed by an observer) was recorded on paper, and if the user was not able to complete the task, it was recorded as "not completed". The time information can be used to compare relative ease of piloting the different interfaces. They can also show whether the user-distinguishing factors mentioned in the questionnaire are relevant or not. In addition, recording the times is an effective way to track progress in iterative interface development. In the next round of usability testing, completion time data can be compared to that of the previous round of testing in order to determine if the interface has indeed improved. Before beginning the tasks using the SpaceNavigator and the Leap device, each user was given a brief (5 minute) "training" period to familiarize himself with the device by using it to move around the OpenGL cube graphic, created for debugging during the development phase. At the end of the experiment, the user is asked to fill out the previously mentioned questionnaire.
Ideally, when using an iterative approach to design, developers should perform several rounds of usability testing and improve the interface based on the results after each round. In the context of this project, we perform only two rounds of usability testing due to time constraints. 10 participants took part in the first round, and 5 in the second round. The basic structure of the second round is the same as the first. The only variables are the Leap and SpaceNavigator interfaces which have been improved based on the results of the first round of testing.

8 Results

The quantitative data results for round 1 of the usability testing are displayed in Table 1. The table includes the average completion times for tasks 1 and 2 performed with the AR.Freeflight app, SpaceNavigator and Leap devices respectively. The incomplete column shows the percentage of tasks which were not completed with a given device. Figure 19 displays the differences in completion time across tasks and devices. The SpaceNavigator and Leap times are too close to draw any conclusions about which device is a more natural option for drone piloting. However the AR.Freeflight times are significantly longer than those of the other devices for both Task 1 and Task 2. The percentage of incomplete tasks is also higher for the AR.Freeflight app. These results can perhaps be attributed to the fact that the AR.Freeflight app is the first device which users use to complete the tasks in the experiment. It is possible that the times for the SpaceNavigator and Leap device are faster because users have a higher level of familiarity with the tasks, and more practice piloting the drone once they begin using these devices, since they have already finished completing
the tasks with the AR.Freeflight app. However, if experience is the determining factor then the times for the Leap Motion device should be significantly faster than for the SpaceNavigator as well, and this is clearly not the case.

ROUND 1

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task 1 avg. completion time (s)</th>
<th>Task 2 avg. completion time (s)</th>
<th>incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR.Freeflight</td>
<td>49.50</td>
<td>56.63</td>
<td>30%</td>
</tr>
<tr>
<td>SpaceNavigator</td>
<td>34.33</td>
<td>31.50</td>
<td>15%</td>
</tr>
<tr>
<td>Leap Motion</td>
<td>33.92</td>
<td>37.50</td>
<td>25%</td>
</tr>
</tbody>
</table>

Table 1: Quantitative data, round 1.

Figure 19: Task completion times (in seconds) for each device, in round 1 of usability testing.
Figure 20: Task completion times (in seconds) using the Leap interface, comparing users with and without prior leap experience.

Figure 21: Task 1 completion times (in seconds) for each device, comparing users with and without prior drone experience.
Distinguishing between user groups based on age, video game experience, and familiarity with different gaming devices and RC toys did not reveal very little relevant information in conjunction with task completion times. Several users had some experience with the Leap motion device before the study began, but this did not make them more likely to complete the tasks faster than other users with the Leap interface. Figure 20 displays the average times in seconds for users with prior leap experience (blue) and users with no prior leap experience (red) to complete both tasks using the Leap interface. Prior experience with Leap motion also did not increase the likelihood that a user would prefer the Leap interface over the SpaceNavigator interface, or have faster completion times with the Leap interface than the SpaceNavigator interface. Familiarity with remote control toys also did not affect the data in any noticeable way. However, users who had piloted a drone before had faster-than-average completion times with all the interfaces. Figure 21 shows the Task 1 average completion times (in seconds) for users with prior drone experience versus the average completion times for all users. It is clear from the chart that users with prior drone experience were able to complete the task much faster. Figure 22 shows the Task 2 average completion times (in seconds) for users with prior drone experience versus the average completion times for all users. Task 2 exhibits the same trend, with the users who have prior experience completing it is clear from the chart that users with prior drone experience performed better in Task 2 as well.

All users who took part in the study, except two, reported that they habitually
play video games. The high percentage of gamers in the study could be due to the fact that the people most likely to agree to participate in such a study are tech enthusiasts. It is highly likely that most people who are interested in technology such as drones, also find video games interesting. The data did not reveal that people with video game experience are able to complete the tasks faster than those without video game experience, or that they would prefer any particular device over the others. The users who participated in the study were all between the ages of 18 and 40. Most of them were between age 25 and 30. Since the study took place at a university, it was more difficult to find potential test subjects over 40. Younger people are perhaps also more likely to be interested in trying new technology. According to the data, there was no distinguishable correlation between the user’s age and task completion time with any of the interfaces.

Users also had the opportunity to answer an open question in the questionnaire, in which they could write their thoughts about the experience using the AR.Freeflight app, Leap Motion device and SpaceNavigator to pilot the drone. Almost all users found the AR.Freeflight app less intuitive and less enjoyable than the other interfaces. Most users preferred the Leap interface as the most natural and "fun" of the three interfaces. Despite their preference for the Leap interface, however, users tended to feel that more precision was possible with the SpaceNavigator interface. The majority of users complained about the lack of clear feedback provided by the Leap interface. Since there was no GUI, they found it difficult to orient their hands in the leap field of view or to be certain whether they were executing the correct hand gesture, simply by looking at the drone’s response. All of the users also found it difficult to orient themselves with respect to the drone with all three of the interfaces, but particularly with the Leap interface. Once the drone has turned, and its front is no longer facing in the same direction as the user, the interfaces move the drone relative to the direction in which the drone is facing, not the direction the user is facing. Since the forward facing direction of the drone is always changing, it can be difficult for users to adjust quickly enough to the correct orientation of gesture and direction. The drone is round in shape and its front side looks almost identical to the left, right and back sides. This also makes the user’s task of determining where the drone will move in response to a gesture more challenging. Figure 19 reflects the user’s difficulty adjusting to changing drone orientation, in that the completion times for the AR.Freeflight app and Leap interface are higher for Task 2 than they are for Task 1. In Task 1, the user is expected to fly the drone from the takeoff position near him, to the landing position farther away from him. The drone begins in the takeoff square, facing the same direction as the user (which is also the direction the user wants to fly in). In Task 2, however, the user is expected to fly the drone towards himself, from the far square to the near square. The drone begins by facing towards the user. This shift in orientation could explain why users took longer to complete Task 2 with the AR.Freeflight and Leap interfaces. We do not, however, see the same trend
exhibited in relation to the SpaceNavigator times. This could indicate that the user finds it easier to adapt to changing orientation using SpaceNavigator interface, due to the higher precision and handle-position feedback provided by the SpaceNavigator.

To solve the problem of user orientation, we would have to change the interfaces in such a way that the drone always moves in a direction relative to the user, rather than the drone's front. However, this is quite a hard problem, as attitude determination is non-trivial without external sensing, and due to time constraints as well as lack of technology, we did not consider it feasible to add this improvement to the interface. Instead, to help the user orient himself better, brightly colored tape markers were added to the drone's front side to make it easier to distinguish for round 2 of usability testing. To solve the Leap interface feedback problem, which many users commented on after the first round of usability testing, the Swing window, displaying the OpenGL cube, was added as a GUI to the SpaceNavigator and Leap interfaces. In round 1, the cube interfaces for Leap and SpaceNavigator were used only in a quick training session for the users to familiarize themselves with the new devices by moving the cube around. In the improved interfaces, the cube simultaneously moves in the same direction as the drone flies. The quantitative results of Round 2, shown in the table below, demonstrate the results of these changes.

### ROUND 2

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task 1 avg. completion time (s)</th>
<th>Task 2 avg. completion time (s)</th>
<th>incomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpaceNavigator</td>
<td>29.11</td>
<td>30.50</td>
<td>20%</td>
</tr>
<tr>
<td>Leap Motion</td>
<td>30.75</td>
<td>26.90</td>
<td>10%</td>
</tr>
</tbody>
</table>

Table 2: Quantitative data, round 2.
Figure 23: Task completion times (in seconds) for each device, in round 2 of usability testing.

Figure 24: Round 1 vs Round 2 comparison for task 1.
Table 2 contains the average completion times, in seconds, and the percentage of incomplete tasks for round 2 of usability testing. Figure 23 depicts the differences in completion time (measured in seconds) between tasks 1 and 2, as well as between the Leap and SpaceNavigator interfaces. In round 2, we do not include the AR.Freeflight app since that interface did not change since the first round of testing. Figure 24 shows that there is significant reduction in Task 1 completion time for both the Leap Motion and SpaceNavigator interfaces in Round 2, compared with Round 1. Figure 25 shows the same trend for task 2. These results indicate that the modifications to the interface based on the feedback from round 1 improved the effectiveness of the interface to some extent. Once again, age, experience with RC toys, video games, and familiarity with different input devices does not seem to have affected the quantitative data significantly. None of the users in round 2 of the usability studies had ever flown a drone before.

Overall, the qualitative user feedback for the second round was more positive. No one complained that there was not enough feedback with the Leap interface, although orientation was still a concern for both devices. Despite the fact that the completion times are quite close for the Leap interface and SpaceNavigator interface in round 2, most users still preferred the Leap interface over that of the SpaceNavigator. Several users commented that they found the SpaceNavigator interface to be more precise, but that it was still unintuitive in comparison with the Leap interface, which was both more natural and more enjoyable to use.
9 Conclusion

The fact that completion times using for the AR.Freeflight app. in our usability study, were significantly lower than those of the Leap and SpaceNavigator interfaces is interesting to note. The AR.Freeflight app was designed specifically for the purpose of piloting the Parrot AR.Drone, whereas the SpaceNavigator and Leap Motion device were designed to manipulate 3D graphics on a computer screen. Users in this study considered devices, designed for 3D graphics manipulation, to be more natural and intuitive than the AR.Freeflight app. This could be evidence to suggest that an optimal interface for humans to manipulate 3D computer graphics naturally can also be employed as a natural interface for to manipulate real objects in our 3D world, such as drones and other robots.

There are, however, two important differences between 3D graphics manipulation on a computer screen and the manipulation of real 3D objects, which this project also brings to light. One is the problem of accurate feedback. In the real world there are many factors which can change the position of an object independently of user input. For example, the position and direction of the drone can be affected by wind, obstacles, stabilizing adjustments etc. Often it is impossible for users to know whether one of the drone’s movements was caused by input to the interface or some external factor. This may cause the user to interpret the drone’s feedback incorrectly in certain cases, which can lead to confusion and reduced effectiveness of the interface.

Another difficulty is the orientation problem, mentioned by many users in the qualitative section of the questionnaire. The interfaces used in this project move the drone based on the drone’s orientation, not the users’. Users had difficulty orienting themselves when the drone was not facing the same direction as them, because it forced them to be aware of two different axes of orientation at once in order to pilot the drone effectively. This typically does not pose a problem in 3D graphics manipulation, because it is easy to translate the graphics ”world” such that it corresponds with the user’s perspective. One could argue that video feed from the drone’s camera can be used to acheive the same effect, however this is not the case, because it is not possible for the user to see the drone itself in its own camera feed, and thus adapting to orientation is still difficult once the drone turns to face in a different direction. The results of the usability studies performed in this project indicate that perhaps this discrepancy of orientation can be ameliorated through the use of a GUI. After Round 1 of the usability testing, a GUI was added to the Leap and SpaceNavigator interfaces, displaying a 3D OpenGL cube, which provided feedback by moving in the virtual 3D graphics world in response to user input at the same time as the drone was being controlled. The results of Round 2 of usability testing showed that the addition of this interface significantly improved task completion times, and qualitative user feedback about the interfaces were also more positive. These results suggest that in order for interfaces designed to manipulate 3D graphics to be adapted
for robot manipulation, it is perhaps necessary to include a corresponding GUI displaying 3D graphics whose movements mimic the robot’s, in order to provide adequate feedback and aid the user in visualizing the changing orientation of the robot with respect to the device.

The results of the usability testing in this project also suggest that the user’s experience with the device is perhaps less important than the user’s familiarity with the 3D object being manipulated. Users who had prior experience with the Leap Motion device did not complete tasks with the Leap interface faster than those with no prior experience. However, users who had prior experience flying recreational drones had completion times which were significantly better-than-average across all interfaces. Perhaps the more determining factor is not the input device, but how well the user understands the movement of the object he wants to manipulate. This could indicate that the best way to improve an interface for 3D objects (both real and graphical) is not to create a more intuitive input device, but instead to improve the GUI in such a way that the it provides the user with more information and feedback about the 3D object.
A Usability Study Documents

A.1 Consent Form

I hereby volunteer to participate in a research project conducted by the IDSIA Robotics Lab and USI during the course of the Bachelor projects in 2014. I understand that the project is designed to gather information about how users control aerial drones with a variety of devices. The project’s results will be used in the Bachelor thesis by Christine Graff.

My participation in this project is voluntary. I may withdraw and discontinue participation at any time without penalty and without the need to state a reason.

I understand that the researcher will not identify me by name in any reports using information obtained from this usability study, and that my confidentiality as a participant in this study will remain secure. Subsequent uses of records and data will be subject to standard data use policies which protect the anonymity of individuals.

I understand that there may be pictures and videos taken of the study. I understand that I might appear in those pictures and videos. These might further be used to present the research (e.g. in the thesis or on our webpage) to the general public. I may request the removal of any such pictures and videos at any time in the future by contacting the study organisers.

I want to be notified when the final results are published and presented. (Please provide your email address!)

Study organisers

Christine Graff
Bachelor Student
Università della Svizzera Italiana (USI)

Jürgen Leitner, Alexander Förster
IDSIA Robotics Lab
Università della Svizzera Italiana (USI)
A.2 Questionnaire

User Questionnaire Aerial Robotics User Studies  
BSc Project Christine Graff  
Lugano, May 2014

Age:  □ 18 - 24  □ 25 - 30  □ 31 - 40  □ 41+  
Gender:  □ m □ f

Have you flown a drone before?  □ no  □ yes  ..................................................

Do you play video games?  □ no  □ yes, how often ...........................................

Which of the following devices have you used before?

□ Nintendo Wii Controller  □ Microsoft Kinect  □ Sony Playstation Move
□ Leap Motion Device  □ 3D Mouse  □ Joystick  □ Tablet/Smartphone

Do you have experience with remote controlled toys, such as RC cars or planes?

□ no  □ yes  .................................................................

Describe your experience flying the drone with the 3D Mouse and the Leap Device?
(How comfortable/enjoyable/natural/intuitive was the control experience with each device?)
References


REFERENCES


[19] "Parrot AR.Drone Quadricopter Controlled by iPod touch, iPhone, iPad, and Android Devices (Orange/Blue)". Amazon.com.


[28] Monica Landoni, senior research fellow at the Faculty of Informatics.


