Next Generation of User Interface Design

Speech & Gesture recognition

By Willi Wolff & Tunahan Pece

Submission Date: June 9, 2014

LEAP MOTION

Abstract

With massive influx of computers in society, human-computer interaction (HCI) has created an increasingly high demand for new user interfaces (UI). Into the bargain, usability of user interfaces was quite long laborious and ponderous to use. Consequently, these interfaces have grown to be familiar but inherently limited the speed and naturalness with which humans could interact with computers. This limitation represents one main aspect this document is going to deal with.

"User Interface Design" is actually not a completely new topic but indeed in reference to new technology. The way of interacting with computers is changing constantly. After the development from text-based UIs controlled by keyboard to graphical user interfaces (GUI) usable via mouse and keyboard, a first step in the direction of natural operation was done by introducing touch-sensitive devices like todays smart phones and tablet devices. Another technic, already well-known from videogame consoles but not recently well developed for computer use is represented by gesture-based controls. Gesture recognition encapsulates a topic in computer science referring to human-machine interaction with the goal of interpreting human gestures by the help of mathematical algorithms.[1] Along gesture-based controls another great way of interaction is rising up – speech recognition. A long-term attempt in HCI has been to migrate the natural means that humans employ to communicate with each other into HCI.

Nevertheless, the following paper shall serve both an introduction to future-oriented accessible use of computers in terms of gesture and speech recognition as well as presenting the next generation of user interfaces. Thereby the study work covers the precise way of functioning of the newly released *LEAP Motion Controller* with specially going into detail on enhanced gesture based theoretical background. Unfortunately the original used algorithm of the Leap Motion has not been published yet due to the fact of patent pending. Hence, this document will take a closer look to already published gesture recognition algorithms and cover the Leap Motion algorithm briefly. In addition one chapter is going to deal with basic speech recognition technics and new concepts for designing graphical user interfaces (GUI) will be particularly introduced. Subsequently, a newly developed way of controlling an *AR Drone* as visible above with help of *The Leap* will be presented. The thereafter following section deals with implementation of extended gesture-based controls and advanced interfaces such as speech recognition. To sum it up NUI remarkable functionality will be generally the focus of this document.

[1] comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review*

Table of Contents

List of abbreviations

Acronym	Term
CLI	Command Line Interface
GUI	Graphical User Interface
HCI	Human-Computer Interaction
NUI	Natural User Interface
UI	User Interface
W3C	World Wide Web Consortium

List of tables

List of figures

1 Introduction

In May 21st, 2012 the American company *Leap Motion, Inc.* firstly introduced its new product *The Leap* which is a computer hardware for gestured-based control with extremely high accuracy. It is indeed not completely new, since Microsoft has led the charge toward a future of gestured-based controls with its Kinect. Nonetheless, the Leap represents one of the first impressive examples for the next generation of interaction with computers. Instead of using mice and keyboards, one can interact highly precisely by simple gestures. Leap Motion even claims its new product is 200 times more accurate than existing technology.[2]

This product offers developers a variety of different application options which leads us to the topic of this document. As already addressed before, the Leap provides a great opportunity for granting accessibility for miscellaneous people and devices. With regard thereto, this study work expands the fact of accessibility by use of speech recognition. Therefore we would like to introduce the LEAP's algorithms superficially and give an insight how speech recognition algorithms are used to understand spoken commands. This work targets for new areas of utilization base on an existing elementary LEAP controls for an AR Drone 2.0. This will be covered in detail by the end of the documentation in combination with concepts for new designs of next generation user interfaces.

[2] comp. Engadget (May 21st 2012): *Leap Motion reveals super-accurate motion control […]*

2 Theoretical backgrounds

This chapter covers five different sections. Each of them approaches miscellaneous theoretical basics. In this process, this section shall serve as brief introduction for each issue that will be practically dealt with in subsequent chapters. Therefore these sections deliver a fundamental base understanding following sections.

2.1 Natural User Interfaces (NUI)

As new input methods are developed, the third generation of user interface types – Natural User Interface (NUI) is significantly growing its importance for end users, whereby on contrary the Command-line-Interface (CLI) and the Graphical User Interface (GUI) are losing their attractiveness and appropriation. While CLIs are commonly strict and GUIs are using metaphors like the “Home”-button, NUIs are created to be just direct and intuitive. This means new guide lines, methodologies and standards have to be carefully elaborated for developing such interfaces.

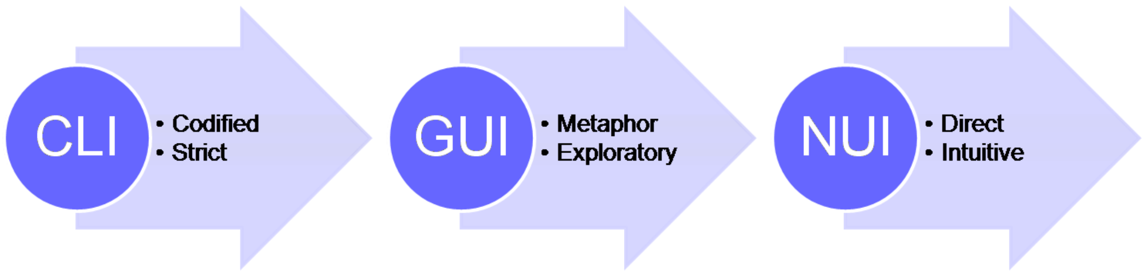


Figure 1 Evolution of the user interface

2.1.1 Definition

A general definition of natural user interfaces:

> *“An NUI is a type of* user interface *that is designed to feel as natural as possible to the user. The goal of an NUI is to create seamless interaction between the human and machine, making the* interface *itself seem to disappear.”*[3]

[3] TechTerms (July 26, 2012): *Definition of NUI*

This definition of NUIs is confirmed by Bill Buxton, a principal researcher at Microsoft Research, and his video[4], in which he points out the aim of NUIs to grant a natural and an intuitive way of connection and interaction between user and computer with the example of tilting and zooming first on a sheet of paper and afterwards on a document with a smartphone or pager.

Further, natural user interfaces inherit a combination of following three components:

- *"Invisible Computing: Invisible computing is when hardware virtually disappears, as computing technology unobtrusively integrates with everyday, natural human function.*
- *Supportive Computing: Supportive computing is computing technology that supports natural human function, rather than requires humans to adapt to computing functions.*
- *Adaptive Computing: Adaptive computing and machine learning intelligently recognize and interpret human patterns to produce output based on relative context."*[5]

2.1.2 Natural input methods

To reach the aim of NUIs, as mentioned above, computers need new input methods on the basis of e.g. human to human interaction.

The most important input methods are pointed out by the *Bresslergroup*[5]:

"Behaviors that could be associated with natural user interfaces include:

- *Natural physical gestures and motion*
- *Verbal and non-verbal speaking*
- *Facial features or natural movements*
- *Eye tracking"*[6]

For Bresslergroup smartphones providing gesture based interactions are only a subcategory of natural user interfaces. In general, this subcategory is the most important one to be looked at for defining developer guide lines. This will be covered in [REFERENZ ZU "Developer guide lines]

2.1.3 Software Ergonomics

In general, software ergonomics aims the improvement of working conditions with the adaptation of software to the abilities and working behavior of its user.[7] Conspicuously, the aims of software ergonomics and natural user interfaces have an intersection, which is also revealed by partially common guide lines for ergonomic GUIs and NUIs.

[4] Youtube (March 21, 2010): *Microsoft Research's Bill Buxton on Natural User Interfaces*
[5] Provita Labs Alex Chong (August 14, 2013): The Future of UI/UX: The Natural User Interface
[6] Bresslergroup: *Interaction Design, Natural User Interfaces*
[7] Ergo-Online (October 09, 2008): *Software-Ergonomie und Benutzerfreundlichkeit*

2.1.4 Developer Guidelines

First of all it is important to point out that input technologies such as the Leap Motion or speech recognition technologies themselves are NUIs. Applications with various features however need a GUI to provide clarity, help the user to understand the system and assist user inputs. A GUI e.g. showing the recognized gesture will avoid misunderstandings and smoothly assist the user.

In case the application allows to control the common cursor by gesture, the most important guidelines are the following:

1) Fitts's-law[8]	– Considering the distance and size of input fields and buttons – Fast access and control provide usability
2) Grouping related objects	– Related objects are placed close to each other – Raising usability and intuitiveness
3) Considering the logic of the user	– Commonly top-down – Improving intuitiveness
4) Considering precision loss of the cursor with gestures	– Great size of interaction fields – Avoiding muscle exhaustion – One finger scrolling
5) Using already spread gestures	– Two finger zoom-in and zoom-out – Three finger rotating – Four finger swipe – Better acceptance and intuitiveness
6) Avoiding permanent repetitions	– Avoiding muscle exhaustion

[8] Interaction Design Foundation (August 26, 2013): *Fitts's Law by Mehmet Göktürk*

2.2 Ways of Gesture recognition

As partly introduced before, while computing moves increasingly away from desktops, there is a rising need for new ways to interact with computer interfaces. The focus of new interfaces lays stress on supporting human ways of receiving and perceiving information from their environment, such called Natural User Interfaces. This section introduces different the definition of a gesture as well as various types of gestures and additionally presents a classification of gesture recognition algorithms, since those represent one main interest which can be used to create user interfaces that offer natural ways of interaction between machines and humans. Contemporaneously, these concepts will be useful for the development of natural user interface using the LEAP for gesture recognition.

2.2.1 Definition

Outside the HCI framework, gestures are differently defined. They often refer to communicational aspects of the human hand and body movements. However, in the domain of HCI the gesture's definition differs, because inside of a computer controlled environment one wants to use the human hand to perform tasks that mimic bot the natural use of the hand as a manipulator, and its use in human-machine communication.[9]

Therefore there were made differences between several types of gestures, which will be explained in detail in the next subsection.

[9] comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* | p. 679

2.2.2 Types of gesture recognition

During the introduction of various multi-touch Frameworks one differentiates between two types of gesture recognition.[10] They thereby consider online gestures as direct manipulations for instance scaling and rotating of objects. In contrast, offline gestures are processed after the interaction is finished, like a circle is drawn to activate a context menu. In this context, gestures that are recognized by the AR.Drone are represented as online gestures, since the user's movements are directly effecting the AR-Drone's flight behavior without being processed.[9]

Figure 1 shows various models of interaction that should be dealt with to offer NUI's. Thereby the focus is on creating user interfaces which may have different possibilities to function, depending on the purpose of the application. Referring to the purpose, the programmer may choose to implement different ways to operate on or with the user interface. But there always needs to be considered a fallback concept that offers adaption of the interaction model to the terminal device/ user.

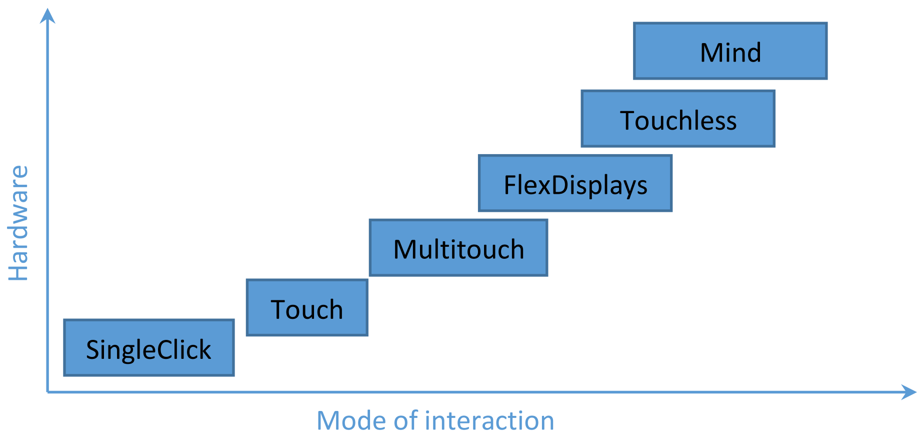


Figure 2 Different Models of Interaction

Even though, the upcoming user interface supports natural ways of interacting for the user, the user will still be able to participate with the user interface in a usual way by means of GUI's – fallback concept.

[10] D. Kammer, M. Keck, G. Freitag, M. Wacker (unkown): *Taxonomy and Overview of Multi-touch Frameworks: Architecture, Scope and Features*

2.2.2 Gestural Taxonomy

This section shall introduce a classification of gestures. Since the LEAP algorithm is not released yet, we like to present a brief overview of the gesture classes that have been defined.

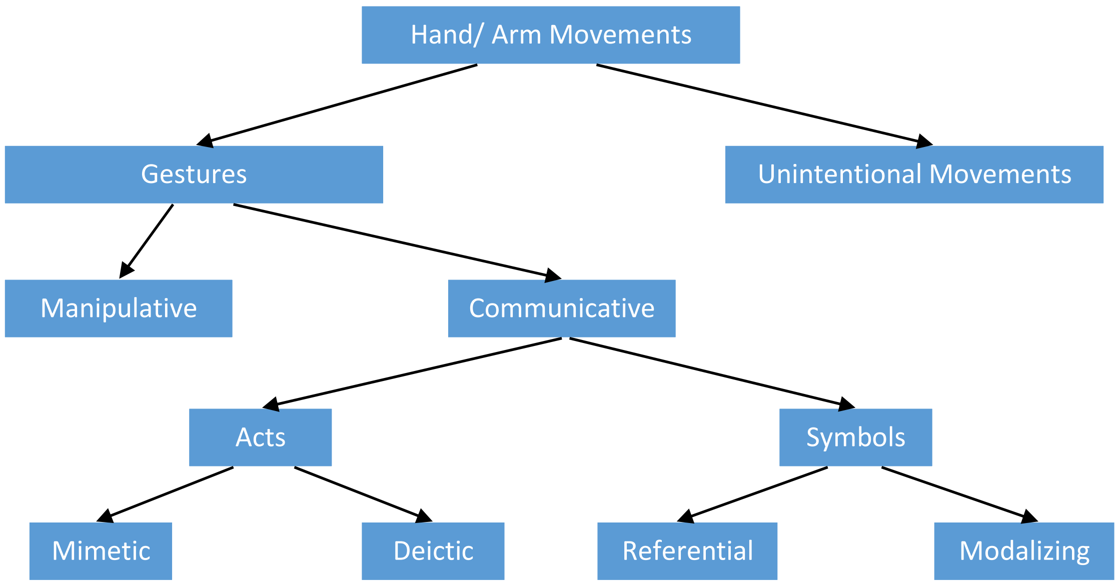


Figure 3 Classification of Gestures

In general, all gestures can be first classified into two major classes, as can be seen in *Figure 2* from above. Thereby unintentional movements do not convey any meaningful information. Unlike gestures, which can have two modalities: *communicative* and *manipulative* (comp. *Figure 2*).

In this context, manipulative gestures are obviously the ones used to act on objects in an environment[11], for instance object rotation or movement. On the other hand, communicative gestures have an inherent communicational purpose, which in a natural environment is usually accompanied by speech. Thereby *Communicative* can either be *Acts* or *Symbols*, in which Symbols are identified by its linguistic role, whereby they symbolize referential action, for example circular motion of index finger. In contrary, acts are gestures that are directly related to the interpretation of the movement itself. Such movements can be classified again as either mimetic (imitate some actions) or deictic (pointing acts).[10]

[11] comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* | p. 680

2.2.3 Spatial Modelling of Gestures

Gestures are generally observed as hand and arm movements, action in 3D space. The description of gestures, hence, also involves the characterization of their spatial properties and changes. In a HCI domain, one can basically differentiate between two major models. On the one side, there are *3D Hand Model-Based* models of gestures that use articulated models of the human hand and arm to estimate the hand and arm movement parameters. Such movements are later recognized as gestures. On the other side, *Appearance-based models* directly link the appearance of the hand and arm movements in visual images to specific gestures.[12]

There are further subclasses for each model, but this is beyond the scope of this document. Therefore detailed information on modelling of gestures can be found in the paper listed in the sources section.

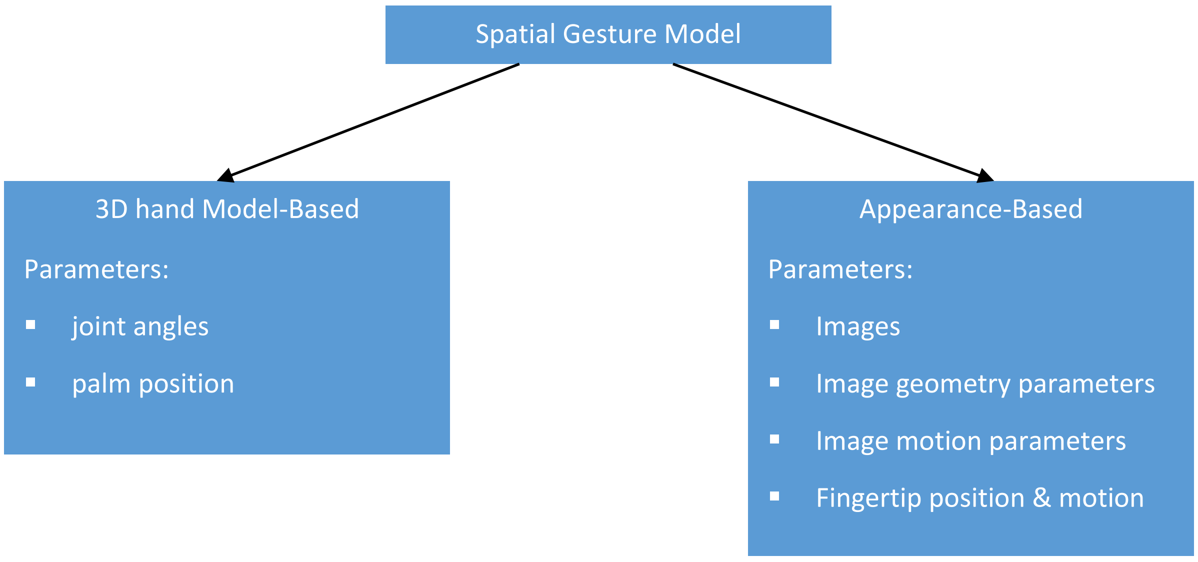


Figure 4 Spatial Gesture Model

[12] comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* | p. 681

2.2.4 Gesture Recognition using Wireless Signals - WiSee

Even if this topic is not directly related to what this document and the upcoming pilot is going to focus on, we like to show different and new methods for intuitive ways enhancing possibilities for NUIs.

WiSee is the name of the first wireless system which can identify gestures in line-of-sight, non-line-of-sight and through-the-wall scenarios.[13] Unlike other gesture recognition systems as Kinect, Leap Motion or MYO, WiSee requires neither an infrastructure of cameras nor user instrumentation of devices.

WiSee leverages the property of Doppler shift, which is the frequency change of a wave as its source moves relative to the observer. It thereby considers the multi-path reflections from the human body as waves from a source, then a human performing a gesture, results in a pattern of Doppler shits at the wireless receiver. Thus, a user moving her hand away from the receiver results in a negative Doppler shift, while moving the hand towards the receiver results in a positive Doppler shift. One main problem that occurs using Wi-Fi signals is that human hand gestures result in very small Doppler shits than are difficult to detect from typical wireless transmissions.

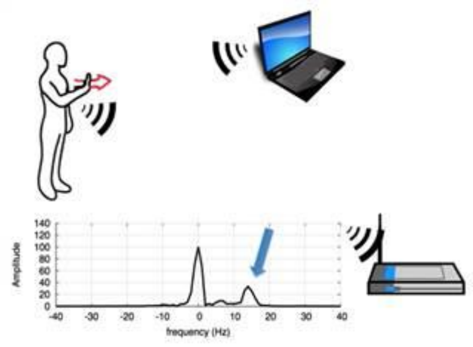


Figure 5 Wireless Gesture Recognition – WiSee

Figure 2 above demonstrates the way of functioning of WiSee, whereby the laptop represents the source and the human influences the Wi-Fi transmission by moving its hand forward. This small change of frequency in form of a Doppler shift can be captured as seen above. Afterwards it can be interpreted by software for initializing an event.

[13] WiSee (unknown): *What is WiSee?*

2.3 Libraries for Speech recognition

Similar to the enormous quantity of gesture recognition algorithms, there are also plenty different frameworks, tools as well as libraries for speech and voice recognition. Some of the most popular ones are covered in the sections beneath. Thereby the focus is on analyzing and matching each one of them against various checkpoints, which are most valuable for this document's aim.

In addition to the checkpoints, an evaluation column judges each library according to different checkpoints. By the end, one library is recommended for the following Leap pilot project.

Checkpoints	Speech recognition library	Specification	Points
		Required Functionality	
Scope of operation	Google Web Speech API	fully satisfied	+5
	Julius	fully satisfied	+5
	Pocketsphinx.js	satisfied	+4
	Annyang	fully satisfied	+5
Complexity, Ease of Integration	Google Web Speech API	low/easy	+5
	Julius	complex/ medium	+2
	Pocketsphinx.js	low/ easy	+5
	Annyang	low/ easy	+5
License	Google Web Speech API	Open-Source (Apache 2.0 License)	+5
	Julius	Open-Source[14]	+5
	Pocketsphinx.js	Open-Source (BSD-License)[15]	+5
	Annyang	Open-Source (MIT)[16]	+5
Browser Support	Google Web Speech API	Chrome 25+[14]	+1
	Julius	Desktop Application[15]	0
	Pocketsphinx.js	Chrome 25+, Firefox[15]	+2
	Annyang	Chrome 25+, FF, Opera, Safari, IE 11[16]	+5
Community, Version & Release Date	Google Web Speech API	Large/ January 2014	+5
	Julius	Small/ 4.3.1/ January 2014	+4
	Pocketsphinx.js	Small but increasing/ March 2014	+2
	Annyang	Small but increasing/ 1.1.0/ January 2014	+2

[14] StiltSoft (May 2, 2013): *Google Chrome: How to Use the Web Speech API*
[15] Julius (2014): *Open-Source Large Vocabulary CSR Engine Julius*

Quality & Accuracy	Google Web Speech API	Excellent	+5
	Julius	Satisfiable	+4
	Pocketsphinx.js	Sufficient	+2
	Annyang	well	+4
Total Score	Google Web Speech API	27	
	Julius	24	
	Pocketsphinx.js	24	
	Annyang	30	

Table 1 Speech Recognition Library Comparison

This subsection shall underline and explain the evaluation from above. In reference to the ***Scope of Operation*** all chosen libraries scored similarly. The only reason why *PocketSphinx.js* did not get all of the available points is that it always requires a grammar to function properly. This way, the recognition is very inflexible and static.[16] In contrast to the others which offer different possibilities to ensure dynamic speech recognition with help of wildcards or named variables.[17]

Referring to ***Complexity & Ease of Integration***, all web-based libraries scored the full amount of available points, unlike *Julius* which actually is only available for operating systems as external software. Therefore its integration is quite difficult, because it needs to recognize and interpret audio data in real-time and simultaneously forward those in a completely different channel to the AR Drone via Wi-Fi. In contrast to the other web based libraries which can easily be implemented inside the natural web user interface using the same communication channel as the interface itself. In addition, *Julius* supports various very detailed options to configure different recognition situations. It comes with a great variety of functionality but for this project's purpose it is too complex.

The ***License*** checkpoint was fulfilled highly satisfiable by all chosen libraries, because of the fact that this was an initial criteria for choosing usable libraries – since this is an open-source project.

In accordance to the ***Browser Support***, there are quite a lot of recognizable differences. As the *Table 1* already displays, there was given one point for each supporting browser (Google Chrome, Internet Explorer, Firefox, Opera, Safari).

[16] Pocketsphinx.js Speech Recognition in JavaScript (unknown): *Welcome to PocketSphinx.js*
[17] Annyang (unknown): *SpeechRecognition that just works*

The checkpoint ***Community, Version & Release Date*** lay stress on the community size as well as topicality of the software. Besides the fact that all of the chosen libraries are up to date and are still being improved, the Google's community is tremendously higher than those of the others. Moreover Google tries to set up standards which may be soon officially accepted by the World Wide Web Consortium (W3C). The other three have an increasing community since the interest in NUI's rises. Indeed Julius community is very extensive but not that popular as Googles way of trying to set new standards.

Last but not least, ***Quality & Accuracy*** plays the most interesting aspect in this whole evaluation part. As addressed before, *Julius* comes with a great spectrum of different recognition functionality, but lays stress on too intense detail which is not required in this use case. Therefore it is overqualified and does not get the full amount of points. In contrary, *Google's Web Speech API* comes with its overwhelming integration and mainly perfect fit into already webpage development technics.

In this context, *PocketSphinx.js* always tries to match recorded data to its currently given grammar. If the term which was said is not defined in the grammar, *PocketSphinx.js* actually tries to enforce finding a match – which seems to be guessing. This behavior often ends up getting awkward results, which is according to this study work a major disadvantage, since there is not much failure space for those commands which shall be later recognized by the NUI controlling the AR Drone. Nevertheless, *PocketSphinx.js* still offers great recognition for the terms defined in its grammar, but nothing beyond that. Therefore two points are justifying those above mentioned aspects. Lastly Annyang completely fulfills this documents particularly desired speech recognition requirements by its great developer opportunities of using wildcards, named variables or optional word phrases.

After evaluation of those four different libraries according to the six most relevant checkpoints, Annyang finally suits all requirements best and is therefore recommended for the upcoming development.

2.4 Annyang Functionality

As briefly introduced before, Annyang offers great and simple support for web-based speech recognition. This section shortly presents Anyang's opportunities for dynamic and flexible recognition.

To sum it up, annyang is a small javascript library that supports multiple languages.[18] It is even that simple, that developers with basic programming skills can easily understand and use it. That's possible, because of Anyang's simple way of functioning by matching rules containing strings against its recognized words, which is shown in the following demonstration:

(1) Importing Annyang library

```
<script
src="//cdnjs.cloudflare.com/ajax/libs/annyang/1.1.0/annyang
.min.js">
</script>
```

(2) Define Command JSON object

```
var commands = {
       'show me *tag: showFlickr,
       'calculate :month stats': calculateStats,
       'say hello (to my little) friend': greeting
};
```

- the wildcard operator "*" passes any term after is as parameter to the function
- the colon operator ":", also called *named variable* is a one word variable that can be placed anywhere in the command
- the parenthesis can be used to define optional phrases, annyang will respond to both – no matter if the term was recognized or not

(3) Define functions

```
var showFlickr = function(tag) {
  var url =
  'http://api.flickr.com/services/rest/?tags='+tag;
  $.getJSON(url);
}
```

[18] annyang! (unkown): *annyang! SpeechRecognition that just works*

```
var calculateStats = function(month) {

  $('#stats').text('Statistics for '+month);

}

var greeting = function() {

  $('#greeting').text('Hello!');
}
```

- each function is saved in its corresponding variable, being accessible via the variable name based on the one in the commands array[18]

2.5 Touchless for Windows

As one of the aims of this project is to navigate contact-free on the UI the Leap Motion Application "Touchless for Windows"[19] is an important software requirement, since with help of this application the user is able to point with his or her finger on the screen and threw the gesture recognition of Leap Motion a cursor comparable to the mouse cursor appears on the screen. As the description of "Touchless for Windows" in the Leap Motion Airspace Store tells:

"Touchless for Windows creates a virtual 'touch' screen in the air."[19]

This two dimensional virtual touch screen divides the recognition sphere of Leap Motion in two functional areas comparable to Figure 7. The virtual screen is in the middle of the sphere and is the area where on touch a click event is triggered on the correspondent point on the real screen. Pointing in front of the virtual screen is similar to moving the cursor around without clicking, which also triggers hover effects of UI elements.

Since this application not only allows navigation on the Leap Pilot UI, but on the whole operation system, depending on usage it should be switched on and off over its icon in the windows toolbar.

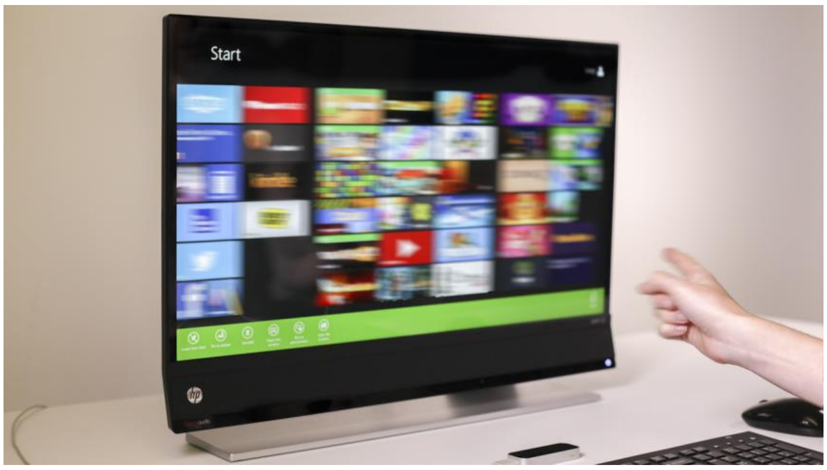


Figure 6 Touchless For Windows

[19] comp. Leap Motion (unknown): Touchless For Windows

2.6 Required Hardware

2.6.1 Leap Motion technical specifications

In general the Leap is a small USB peripheral which creates a 3D interaction space of eight sphere feet to precisely interact with and control software. Thereby it senses individual hand and finger movements independently, as well as items like a pen. As stated before the Leap is 200 times more sensitive than existing touch-free products and technologies. Unfortunately due to the patent pending, there are not much detailed information about how the Leap actually works. The fundamental creates a new mathematical algorithm. It uses two CCD cameras and three infrared LEDs to capture 2 frames and combine them to calculate a 3 dimensional one - including depth information. This way an enormously high accuracy can be guaranteed for the user which comes with its difficulties while trying to navigation through small buttons or generally non-gesture-supporting user interfaces. The basic concept profits of the reflection of each finger and calculates the difference of each position which is delivered by the frames. In this process the sensor streams frames of data with a maximum of 290 frames per seconds via USB. Besides a wireless variant is planned for the upcoming generation.[20]

Below *Figure 1* demonstrates what volume the Leap basically is able to recognize while users interact with their desktop monitors.

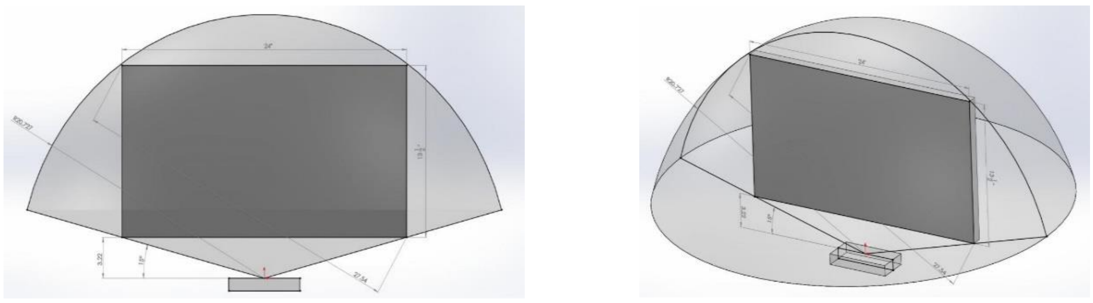


Figure 7 Spherical volume of space captured by the Leap

[20] comp. Forum Leap Motion (May 8, 2012): *The unofficial Leap FAQ*

2.6.2 Construction and technical specification of AR.Drone 2.0

The Parrot AR.Drone 2.0 is a remote controlled civil quadrotor helicopter with an onboard HD-video camera developed and produced by the French company Parrot SA[21]. Supporting Android and iOS, the drone is supposed to be controlled by mobile or tablet devices.

2.6.2.1 Introduction to Quadrotor

"The mechanical structure comprises four rotors attached to the four ends of a crossing to which the battery and the RF hardware are attached. Each pair of opposite rotors is turning the same way. One pair is turning clockwise and the other anti-clockwise."[22]

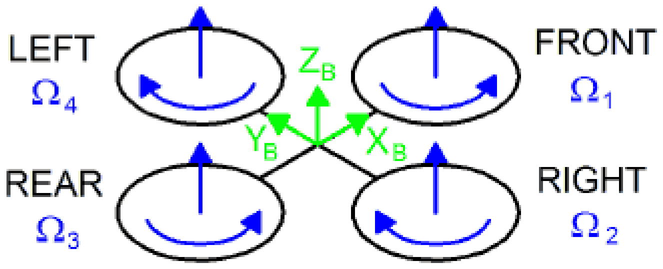
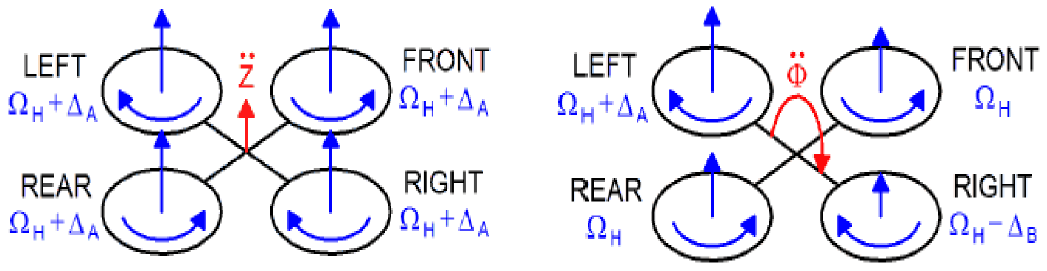
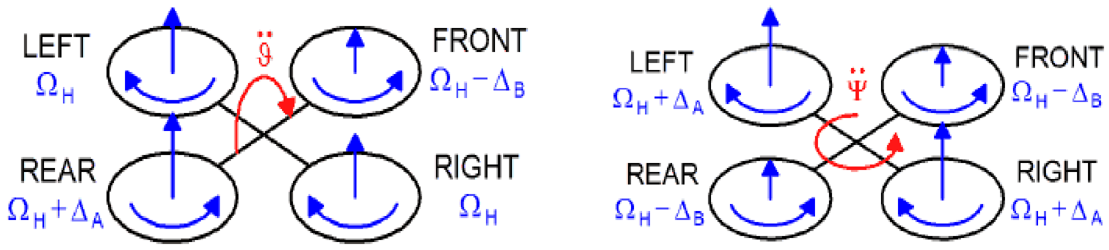


Figure 8 Mechanical structure of the AR.Drone's four rotors



(a) Throttle (b) Roll



(c) Pitch (d) Yaw

Figure 9 Drone movements

[21] comp. Parrot (2008): Legalmentions
[22] (i.a.) Piskorski, S. (2012), ll. 5 - 7

"Maneuvers are obtained by changing pitch, roll and yaw angles of the AR.Drone 2.0."[22]

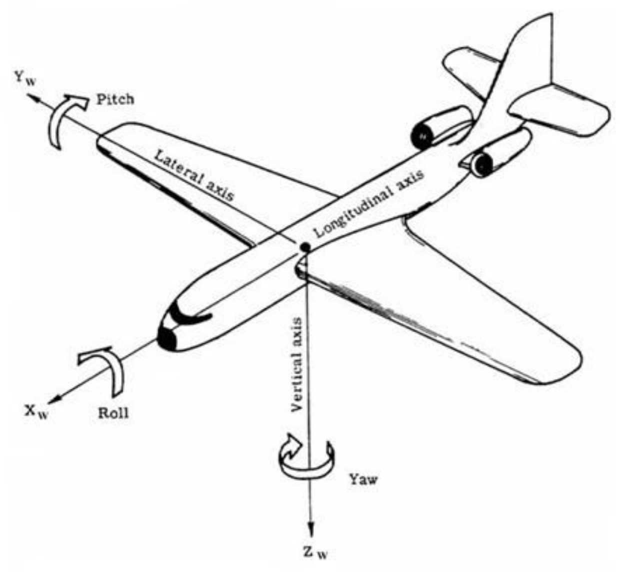


Figure 10 AR.Drone Maneuvers

"Varying left and right rotors speeds the opposite way yields roll movement. This allows to go forth and back. Varying front and rear rotors speeds the opposite way yields pitch movement. Varying each rotor pair speed the opposite way yields yaw movement. This allows turning left and right. "[22]

2.6.2.2 Motion sensors

Below the central hull the drone owns a 6 Degree of Freedom, Micro-Electro-Mechanical Systems based, miniaturized inertial measurement unit, providing the software with pitch, roll and yaw measurements. With these information flying assistance like automatic pitch, roll and yaw stabilization and assisted tilting control is done. To provide altitude information there is also an ultrasound telemeter, providing assisted vertical speed control. The AR.Drone 2.0 also implements a three axis magnetometer, which is mandatory for Absolute Control mode allowing the pilot to send absolute direction commands in contrast to always piloting from the view of the drone.[23]

[23] comp. (i.a.) Piskorski, S. (2012), l. 8

3 Leap Pilot

As partly introduced before, this document's focus lays stress on the development of a natural user interface which supports controls for the Leap Motion use as well as for the AR.Drone 2.0. Therefore we would like to begin this practice-oriented chapter with a brief introduction to the concept and theoretical thoughts according to the structure and functionality of the interface. Thereby the main interest refers to a user interface whose configuration opportunities and ways of interaction are completely natural treatable. Contemporaneously, there will be additional supports for backwards compatible ways of interaction, which means the user interface will still be able to accept mouse and keyboard input.

3.1 Natural User Interface Concept

After first tests with the AR.Drone 2.0, one quickly recognizes the extensive variety of configuration possibilities that are available to perfectly suit the AR.Drone's abilities referring to its environment. Therefore the NUI – introduced in this section, will have four different views to offer. All four are briefly introduced in the following part:

I. Landing Page:

- which will consist of a basic overview to navigate between the other three

II. Configuration View:

- which will contain all adjustment possibilities according to the flight configuration of the AR.Drone

III. Piloting View:

- is actually the summarizing interface or cockpit for flights, that contains read-only information referring to the current status of the drone as well as presenting its recognizing gestures

IV. Media Library View:

- displays all videos of maneuvers and snapshots that have been made during excursions in one media library

3.1.1 Landing Page

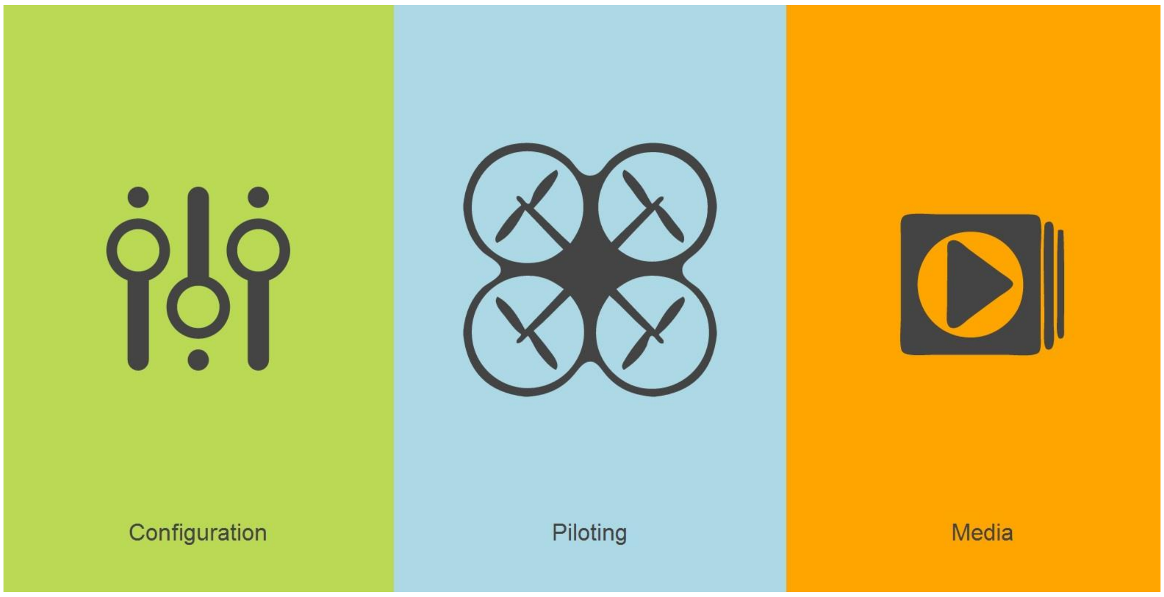


Figure 11 Landing Page Concept

As mentioned before, this conceptional view will only represent a sort of navigation for the others. Each tile can be gesturally chosen by pointing on it with the help of the Leap. Additionally, the user can say the tile's name, which will trigger the interface's speech recognition which then navigates to the called view. How exactly the implementation looks like will be covered in the implementation section.

The ulterior motive is to offer intuitively understandable interfaces which do not require any explanation to be understood. In this context, simplicity plays a significant key element. Thereby the focus of some page should only lay stress on its main concerns – and should not contain distracting information that may affect user's focus, which may end up confusing him/ her.

In connection therewith, and along with *Figure 7* from above, the *Landing Page* only highlights the main interest of any visitor by offering simple comprehensible interface elements. The application icons used, fully underline the straightforwardness.

3.1.2 Configuration View

This subsection deals with the adjustment possibilities and its view for the NUI user. Besides taking off, flying and landing maneuvers, the user will also be able to regulate multiple technical properties completely natural. Therefore one can choose the *Configuration View* to customize the drone's flight behavior depending on which environment the drone is going to fly in or one can chose different piloting modes depending on the user's skills.

Since, there are many different characteristics available, the *Configuration View* is separated into different sections, which basically classify similar adjustment options. With regard thereto, most options shall be accessible for the LEAP Motion and therefore need more space than on usual webpages. The LEAP Motion control is very accurate, which makes it difficult to point onto small elements. Therefore most elements seem to be oversized, which is one significant advantage for gestural control.

But before the different properties will be discussed, this concept first focuses on the classification of them. Altogether the *Configuration View* comprises four sections from which only two directly deal with the customization of the drone. The other two summarize the software and hardware versions as well as some additional sources and contact details.

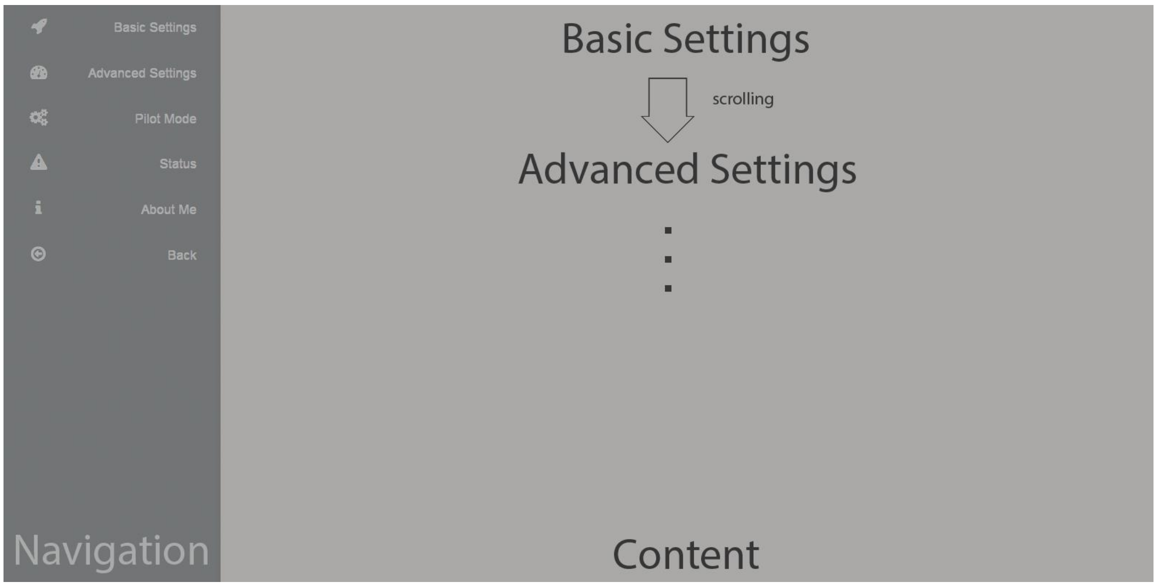


Figure 12 Configuration Concept View

The *Configuration View* is basically a one-pager on which all subpages can be reached by scrolling.

The first class is called *Basic Settings*. The *Basic Settings* include the *Altitude Limit*, *Vertical Speed*, *Rotation Speed* and *Tilt Angle* configuration. All four of them will be adjustable by different sliders which offer the most accessible way for gestural control.

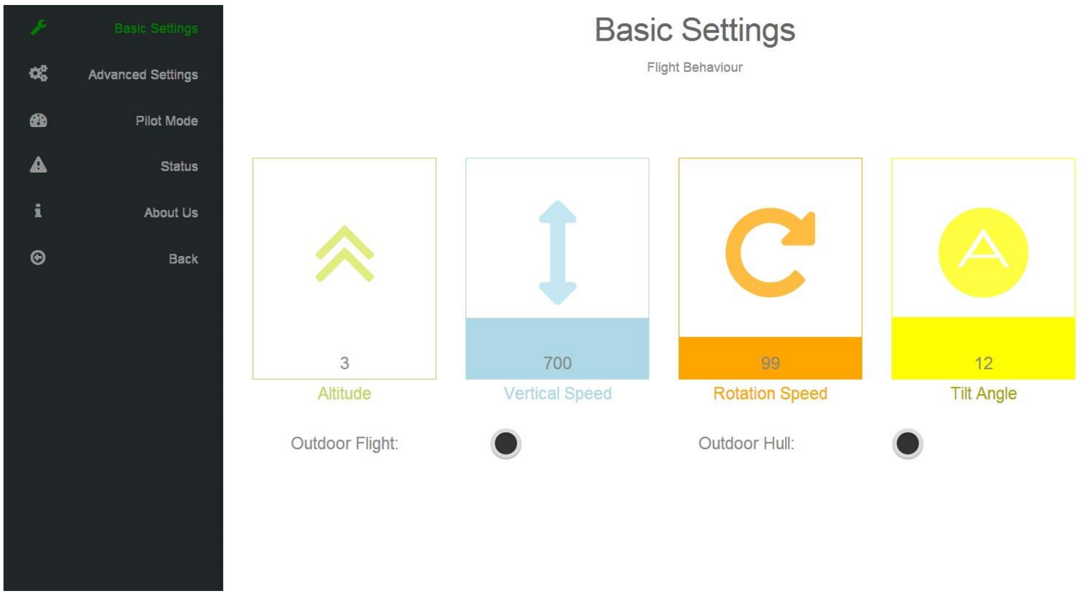


Figure 13 Basic Settings Preview

Moreover, the user can choose a specific profile for either flying in- or outside, because both require the drone to be controlled more or less sensitive depending on the drone's environment. This adjustment is offered by the checkbox, which can be seen in *Figure 9* on the bottom of the picture.

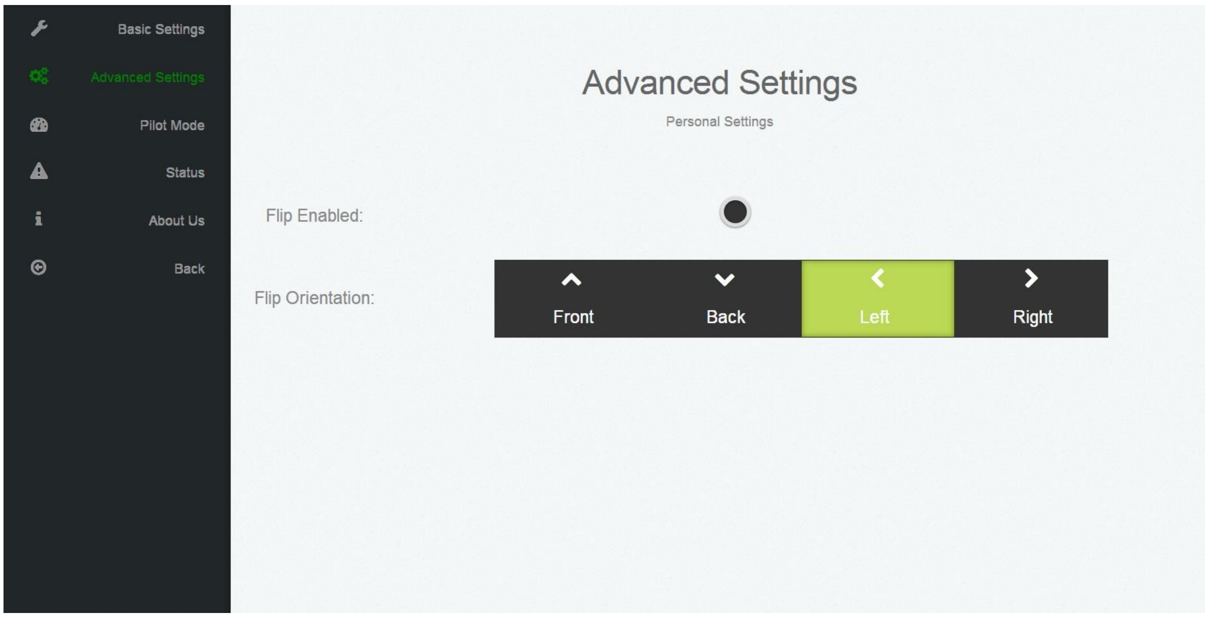
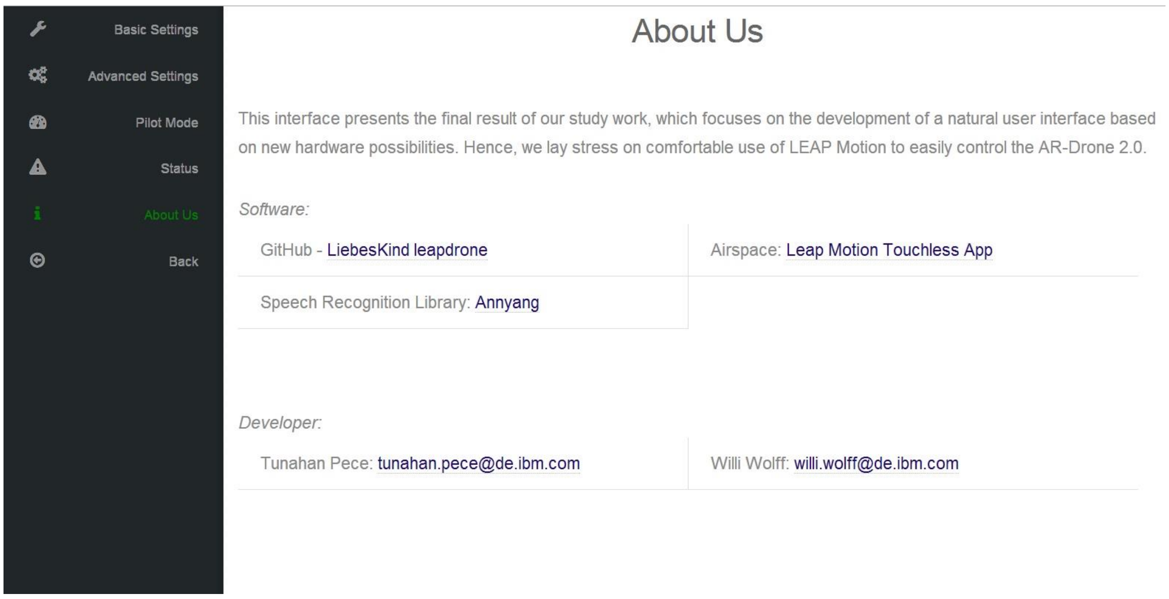


Figure 14 Advanced Settings Preview

According to the *Advanced Settings* the user can define if he wants to enable the option for a flipping maneuver or not. In addition, the user can choose the flip orientation.

The *Piloting Mode* differentiates between *Absolute Mode* and *Normal Mode*. In this process, the *Absolute Mode* offers a more comprehensible way of piloting, because the drone always flies relatively to the user's hand gestures, no matter if it was twisted before or not. This includes for instance: in case the user flies forward and makes a turn of 180 degrees so that the front of the drone is directed to himself, the user does not need to rethink about the reversed control to fly. He can easily move his hand backwards to him to let the drone fly towards him.

Last but not least, the *About Us* section summarizes all relevant sources which have been used referring to the most comfortable way of using the Leap in cooperation with this natural user interface and what speech recognition library and *AR.Drone* control software have been used in this project. Moreover contact details of the developers are provided for further improvement advices.



About Us

This interface presents the final result of our study work, which focuses on the development of a natural user interface based on new hardware possibilities. Hence, we lay stress on comfortable use of LEAP Motion to easily control the AR-Drone 2.0.

Software:

GitHub - LiebesKind leapdrone
Airspace: Leap Motion Touchless App
Speech Recognition Library: Annyang

Developer:

Tunahan Pece: tunahan.pece@de.ibm.com
Willi Wolff: willi.wolff@de.ibm.com

Figure 15 About Us Preview

3.1.3 Piloting View

This view provides Leap Pilot's main functionality, piloting the AR.Drone. First of all, the captured video from the AR.Drone during the flight is streamed and viewed in the Piloting-View. Information like the battery status or flight height of the drone are also displayed in real-time. As an important aspect of piloting assistance, the recognized gestures for flight commands are also displayed and shall help new users to improve their piloting gestures for a better piloting experience. The concept consists of a large video stream area and a minimalistic command console in the bottom center of the screen.

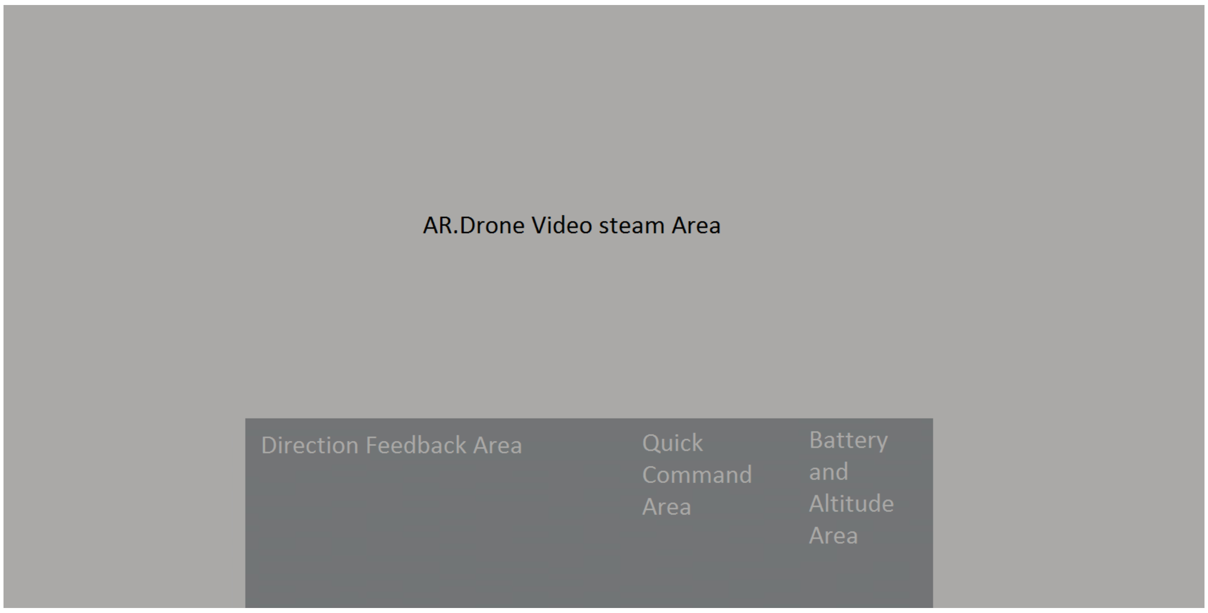


Figure 16 Leap Pilot Piloting view Concept

As Figure 16 the design concept of Leap Pilot's Piloting view reveals, besides the video stream, battery and altitude widgets and the directions feedback area there are also UI components for quick commanding. The quick command area first of all contains a design changer, which will transform the piloting view design from the basic minimalistic design of Leap Pilot to a "Metal"-design. The different designs are displayed in *Figure 17* and *Figure 18*.

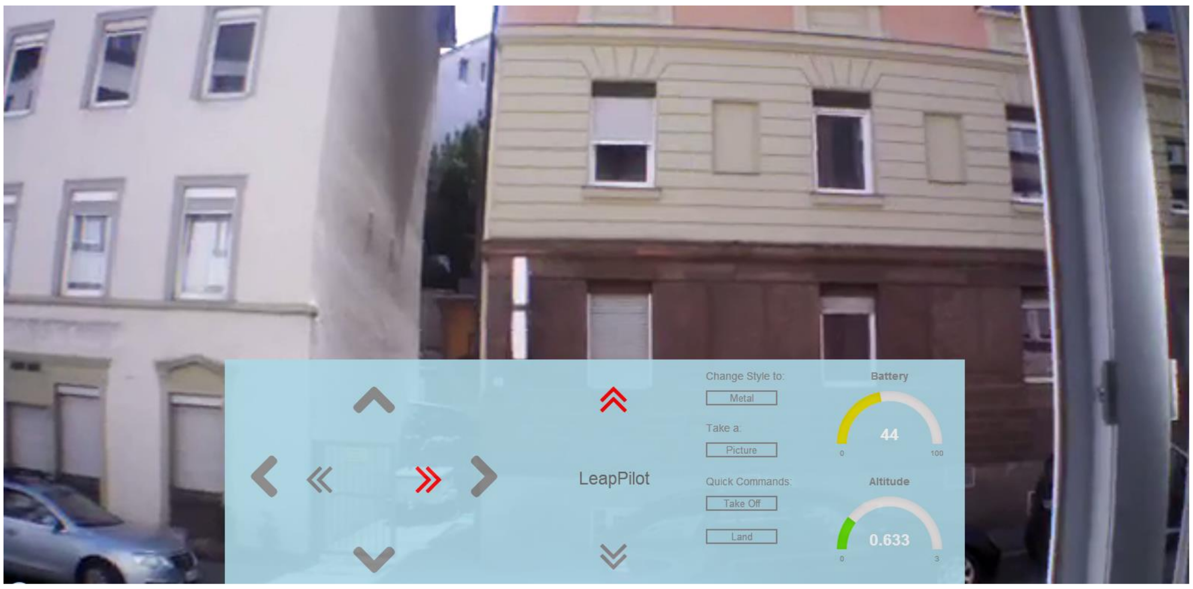


Figure 17 Piloting View Standard Design

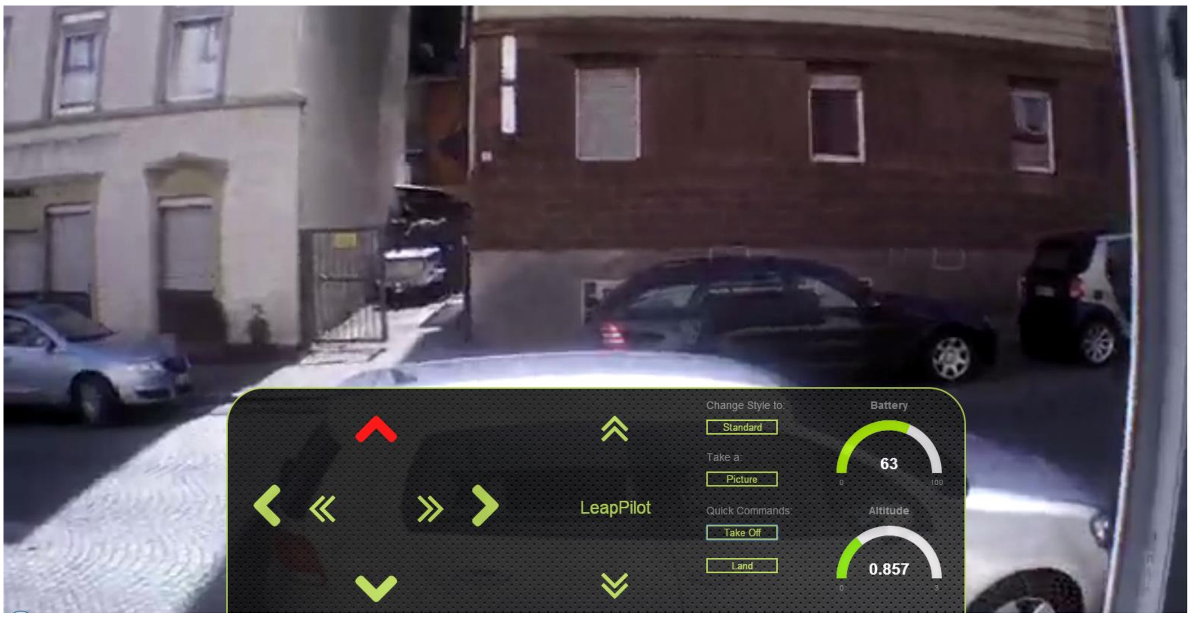


Figure 18 Piloting View Metal Design

Also non gesture “Take Off” and “Land” Buttons are implemented in the command console for usage in emergency situations. The “Picture” Button shall capture the last image received from the drone and save on the local disc. This function is in planning mode and not implemented yet.

The left 65 percent of the command console is planned to reveal the recognized commands sent to the drone, this happens primarily by gestures the Leap Motion recognizes and secondarily by key commands, which are introduced completely in 3.2.3 Piloting View.

All icons, including the arrows and the spinning icon, to be seen in the center of *Figure 19*, are included from Font Awesome[24]. The spinning icons means that no video data is streamed from the drone. The missing widgets for battery and altitude on the very right of the command console mean that no data is received from the drone at all. These widgets are provided by JustGage[25], a JavaScript plugin for generating gauges. More information on implementation and updating are to be found in *Piloting View*.

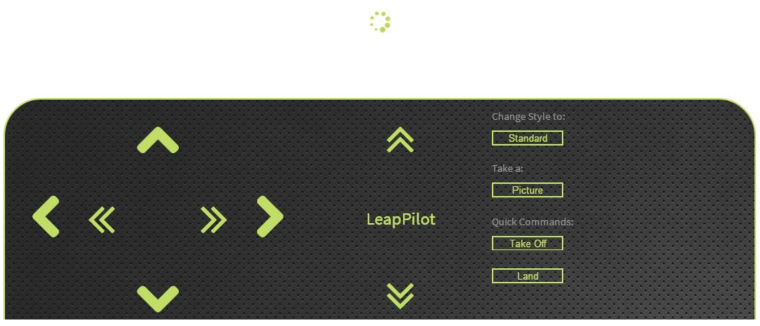


Figure 19 Piloting View without connection to AR.Drone

3.1.4 Media Library View

With the help of the AR.Drone's onboard HD-camera videos and pictures can be captured and will be viewed in this view. The user shall be able to watch and view those immediately.

The individual media elements are presented as large tiles supporting usability with the Leap Motion. Videos and Pictures can be distinguished by the transparent icon on each tile.

In future versions of the Leap pilot the Media Library view could be extended by enabling to grant names to the displayed media elements or a variety of sorting possibilities for the user.

[24] Font Awesome (unknown): The Iconic Font And CSS Framework
[25] JustGage (unknown)

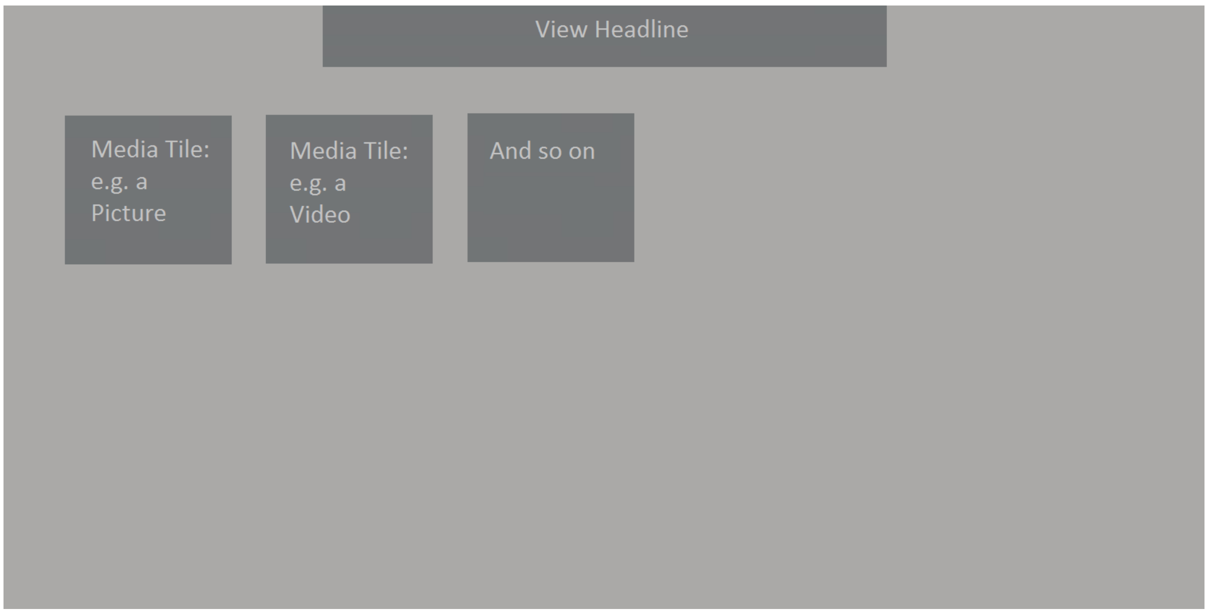


Figure 20 Media Library View Concept

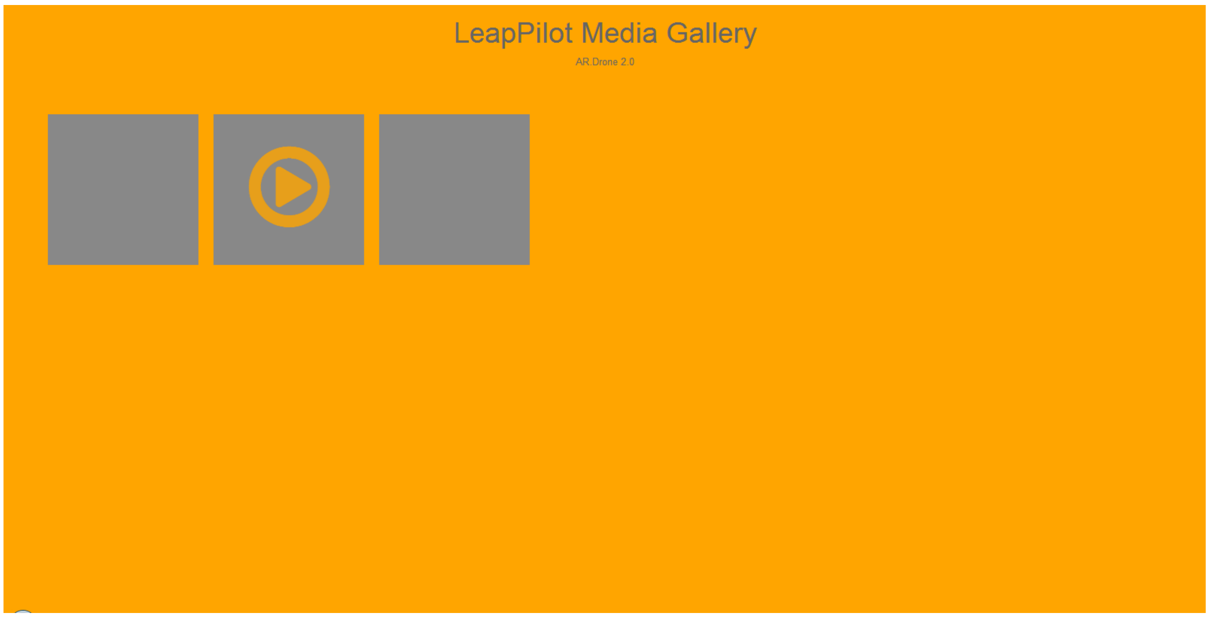


Figure 21 Media Library View

As Figure 21 reveals, the media library is currently in planning and development mode. The design is basically complete but it is missing the JavaScript functionalities it need, to view images and video data saved in a specified path on the local or online storage.

3.2 Implementation & Solution

3.2.1 Landing Page

As partly mentioned before, the large area interface elements offer perfect support for gestural navigation with the help of the LEAP. The HTML-Template is kept simple and will be shortly covered at first.

```
<div class="speechRecognition"></div>

<div class="wrapper">
    <a href="#" class="tile config">
        <h1>Configuration</h1>
    </a>

    <a href="#" class="tile pilot">
        <h1>Piloting</h1>
    </a>

    <a href="#" class="tile media">
        <h1>Media</h1>
    </a>
</div>
```

The first div – Container that holds the attribute *class="speechRecognition"* is used as carrier for text responses according to what has been understood by the Annyang library. It enables an improved interaction with the user interface, because it actually displays what it is trying to do after what has been communicated. This way recognition mistakes can be easily discovered.

Subsequently, three link container represent those clickable areas for navigating to different views. Besides all three are wrapped with another div – Container for responsive design reasons. Since, the gestural navigation is independent from the implementation, because of the fact, that all HTML elements can be treated similarly to the ways of interaction with mouse and keyboard. Therefore click-events are triggered by gestural movements which are analyzed by the *Touchless* Leap App.

Before continuing with the speech recognition implementation, one should ensure that chrome is allowed to use the computer's microphone, which is done by clicking *Allow*. Otherwise you will not be able to control the interface with your voice.

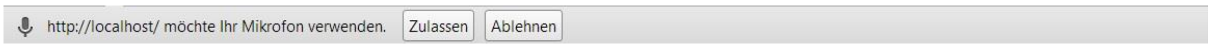


Figure 22 Enables Chrome to use the Computer's microphone

The speech recognition part is separately implemented into our interface by the use of the Annyang library. In reference to the various methods for voice commands, we defined a *commands* array variable *[see line 29, Figure 13]*, which holds all speech commands currently available on the page with their corresponding function name that is called if the command string is recognized. A short introduction according to the functionality can be found in *Annyang Functionality*.

```
window.onload = function(){
    if (annyang) {
      var openView = function (view) {
        var notFound = 0;

        if (view.search("config") != -1) {
          $('div.wrapper').load("config.html");
        } else if (view.search("pilot") != -1) {
          $('div.wrapper').load("views/pilot/pilot.html");
        } else if (view.search("media") != -1) {
          $('div.wrapper').load("views/media/media.html");
        } else {
          notFound = 1;
          $('div.speechRecognition').text("Could not find a match for: " + view);
          $('div.speechRecognition').css('backgroundColor', '#A0000F');

        }

        if (notFound != 1) {
          $('div.speechRecognition').text("Opened " + view);
          $('div.speechRecognition').css('backgroundColor', '#008500');
        }

        $('div.speechRecognition').animate({'marginTop': '0px'}, 200, "swing", function() {
            $(this).delay(3000).animate({'marginTop' : '-70px'});
        });
      }

      var commands = {
          'Open *view': openView
      };

      annyang.addCommands(commands);
      annyang.debug();
      annyang.start();
    }
}
```

Figure 23 Landing Page Speech Recognition Implementation

To open a view with the help of one's voice, the following syntax is crucial for Annyang's comprehension:

"Open <View Name>" *<View Name>:= {"config", "pilot", "media"}*

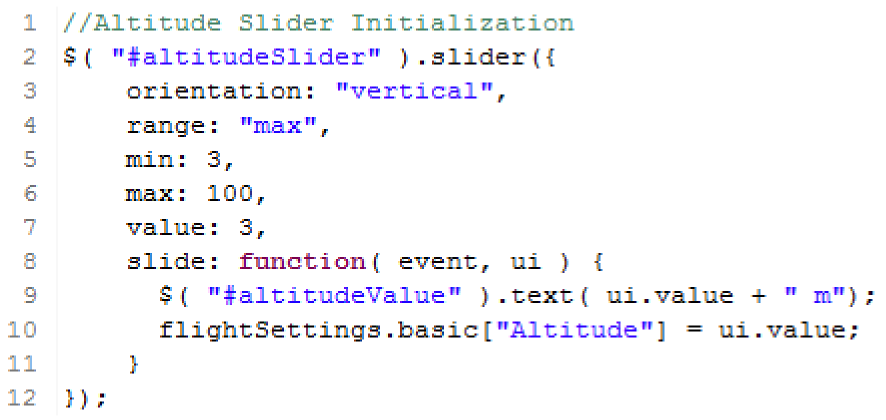
The *View Name* can be any string that contains one of the set's substring. Thus, optimal support for flexible recognition is guaranteed, since any further extensional words are distracting Annyang's speech recognition. Annyang's set of rules should always be defined as generally as possible for the most effective speech recognition. Nevertheless, there should not be any rules which contradict. Annyang is only able to match the most specific rule for a string. There is no way of triggering multiple rules that may apply for the same string. Therefore the generality of a rule should be limited to its circumstances that eventually apply if there are more than one rule.

After the correct function name is called, the string flexibility of the function's parameter is guaranteed by the *search()* JavaScript function which looks for a specified substring inside a string. This method is used in *Figure 12 [see line 6 – 11]*, to ensure if one of the set's substrings is recognized, the corresponding page is loaded into the div – container that is identified by its class called *wrapper*.

In case there was a string recognized by Annyang that did actually match the *Open ** - rule but did not contain any of the defined substrings, the statement in line 19 applies, which states an error messages that is inserted into the already mentioned div – container for response messages.

3.2.2 Configuration View

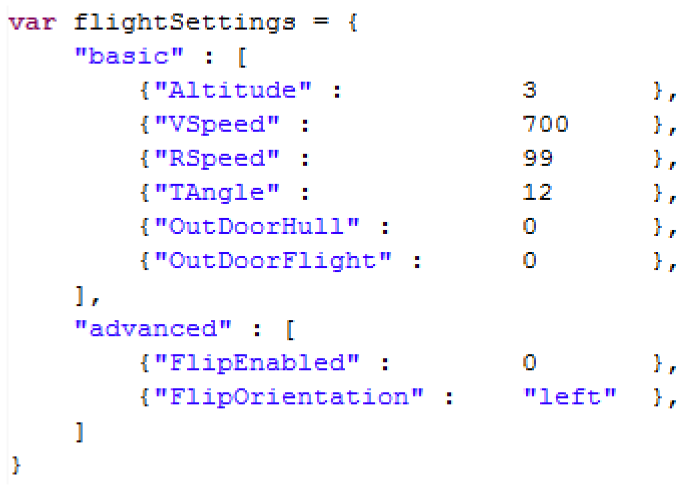
The main focus of this solution deals with the JavaScript speech recognition implementation. Besides, this section will briefly cover the different sections of the configuration view. According to the four sections of the configuration view different UI elements were used to enable gesture support. Hence, vertical sliders and large area checkboxes offer easy accessibility for the use of the Leap motion controller. Each vertical slider is based on the jQuery UI slider plugin, which offers simple JavaScript syntax:



```
//Altitude Slider Initialization
$( "#altitudeSlider" ).slider({
    orientation: "vertical",
    range: "max",
    min: 3,
    max: 100,
    value: 3,
    slide: function( event, ui ) {
      $( "#altitudeValue" ).text( ui.value + " m");
      flightSettings.basic["Altitude"] = ui.value;
    }
});
```

Figure 24 jQuery Vertical slider Plugin

Figure 14 perfectly demonstrates the *Altitude* slider initialization, which holds several important attributes. Thereby, *min* and *max* define the range in which values can be chosen. In addition, the *slide* variable triggers the callback function that displays the value of the slider itself in the label as well as saves its value into a JSON object – named *flightSettings* which holds all configuration values. The JSON object can be found in the file named *config_json_object.js* and looks as follows:



```
var flightSettings = {
    "basic" : [
        {"Altitude" :           3       },
        {"VSpeed" :             700     },
        {"RSpeed" :             99      },
        {"TAngle" :             12      },
        {"OutDoorHull" :        0       },
        {"OutDoorFlight" :      0       },
    ],
    "advanced" : [
        {"FlipEnabled" :        0       },
        {"FlipOrientation" :    "left"  },
    ]
}
```

Figure 25 JSON object containing configuration details [not complete]

Since the configuration view offers various accessibility options for different speech commands, the JavaScript implementation is more complex compared to the variety of the landing page. Regardless which view is currently open, annyang's *commands* array contains all available speech commands from the beginning on, which enables the user to adjust single configuration parameters without even opening the configuration view. Furthermore every speech command can be executed from any view, which shall support high accessibility and flexible navigation.

In addition to the already before introduced *Open* – command to switch between different views, there are further commands available which will be discussed in this following part:

"*Select <Configuration Section>"*

<Configuration Section>:= {"basic", "advance", "pilot/ mode", "status", "page/ landing"}

Similarly to the example of the *Open* – command, which was already explained above, the *Select* speech command functions the same way. Thereby, each of the set's substrings is iteratively matched against the recognized string. If a match is found, the corresponding procedure is executed, whereby in this case a click event is triggered that also fires a *scroll* down event. In contrast to the *setProperty* function whose *call* - command looks as follows:

"(Set) <Property Name> to <Value>"

<Property Name>:= {"altitude", "vertical", "rotation", "tilt/ angle", "flip/ orientation"}

Nonetheless, if properties are set then four steps need to be executed. First, the new property value needs to be saved and written in the JSON Object. Second, the UI slider object needs to be updated as well as third the UI slider label. The second and third step actually only present the new value to the user. But these are not necessary for the AR.Drone configuration update. Thus, the fourth step covers the transaction of the configuration to the drone. But since we had some problems in setting these configurations, we are going to deal with that in the following chapter.

The local storage of the client configuration will be available by saving the JSON object locally. Each time the client connects to the AR.Drone the current JSON object is transferred as configuration for the drone. Since the node.js server does not save any configuration details, but transfers all client-side incoming configuration commands directly to the drone, this is the only way for easily saving the client's drone configuration permanently.

Lastly, another speech command to turn on or off checkboxes by the help of the following call command:

"Enable (outdoor) <Property Name>"

<Property Name>:= {"flight", "protection", "flip"}

This command is preferred used for the options like switching *Outdoor Hull*, *Outdoor Flight*, *Flip Enabled* and *Absolute Mode*.

3.2.2.1 Problem of Configuration Transmission

As slightly mentioned before, there is a problem with transferring the configuration commands from the server to the AR.Drone. But before dealing directly with the problem, this section gives a hierarchic overview from the beginning to the point where the problem occurs.

(1) Call client-side configuration command

```
18  slide: function( event, ui ) {
19    $( "#altitudeValue" ).text( ui.value + " m");
20    flightSettings.basic["Altitude"] = ui.value;
21
22    faye.publish("/drone/config", { // sends a message to /drone/ with config for max. altitude
23      key: "control:altitude_max",
24      value: "\"" + (ui.value * 1000) + "\""  //100.000 are about 100m from ground
25    });
26  }
```

Figure 26 Client-Side Configuration Submission

Every time the value of the jQuery UI Slider of the *Altitude* is changed, the above function is executed. The most important part states the line 22 to line 25, which publishes a *key – value – pair* to the server. The key *control:altitude_max* names the representative variable for setting the maximum altitude for the AR.Drone. As described in the developer guide, the accepted values are in the range of 3.000 to 100.000 and represent Millimeter values.[26] *Figure 16* basically shows a general example on how the process of initiation of setting a configuration parameter starts.

(2) Server Side Subscription

On the other side (server-side), these parameter are fetched by the following *subscribe* – command:

```
/*
 *    Subscribe to configuration settings published by the client to set on the drone
 */
client.subscribe("/drone/config", function (d) {
  console.log(d);
    return drone.config(d.key, d.value);
});
```

Figure 27 Server-Side Configuration fetching

With the help of *subscribe* – command, seen in *Figure 17*, the server can get the submitted values of both transferred parameters, which also appear correctly on the server console when the transmission is triggered. This underlines the successful parameter transmission between client and server.

[26] P. Stephane, B. Nicolas, E. Pierre, D. Frederic (May 12, 2012), *AR. Drone Developer Guide*

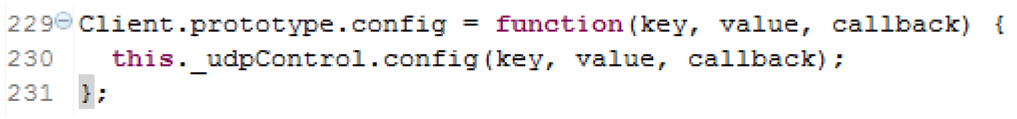
(3) *Config* Function Prototype Call

Nonetheless, the process is not finished here, since the values are not set on the drone. After receiving the values, the server then fires the so called *config* function of the client instance, which is represented as *drone* in *Figure 17*, line 46.

The client variable *drone* calls its *config* function, which can be found in the following class:

\node_modules\ar-drone\lib\Client.js

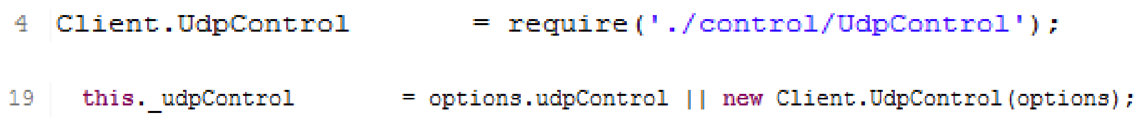
The *config* function looks as follows:



```
Client.prototype.config = function(key, value, callback) {
  this._udpControl.config(key, value, callback);
};
```

Figure 28 config function prototype

The function displays another call of the required *UdpControl* class, which was previously assigned to *_udpControl* and can be seen the figure below:



```
Client.UdpControl        = require('./control/UdpControl');

  this._udpControl        = options.udpControl || new Client.UdpControl(options);
```

Figure 29 Declaration of _udpControl Variable

(4) *UdpControl* Class function reflection

The *config* function is later overwritten by a more specific one. But before that, the *UdpControl* class, found in

\node_modules\ar-drone\lib\control\UdpControl.js ,

dynamically creates *AtCommandCreator objects* for each method name, also called function reflection. This process can seen in *Figure 21* and will be discussed later on in detail.

(5) *AtCommandCreator* class

At this time the reflection process is not that important. The aspect we care about most is overwriting the method that is called from the *ATCommandCreator* and can be found under the following Path:

\node_modules\ar-drone\lib\control\AtCommandCreator.js

The *AtCommandCreator* class implements the more specific *config* function, which basically creates a raw AtCommand that is send to the AR.Drone and sets the desired configuration. The implementation can be seen in *Figure 20* below:

```
66 // Can be called either as config(key, value, callback) or
67 // config(options, callback) where options has at least keys "key" and
68 // "value" (but may also have other keys).
69 AtCommandCreator.prototype.config = function(key, value, callback) {
70   var options;
71   if (key && typeof(key) === 'object') {
72     // Called with signature (options, callback).
73     options = key;
74     callback = value;
75     key = options.key;
76     value = options.value;
77   } else {
78     options = {};
79   }
80   return this.raw('CONFIG', ['"' + key + '"', '"' + value + '"'], true, options, callback);
81 };
```

Figure 30 Config - Function Implementation

This function checks and assigns the parameters (comp. line 73 and following) which are then inserted as parameter to the *raw* function (comp. line 80) that creates an AtCommand which is understandable by the AR.Drone firmware. This AtCommand is returned to the *UdpControl* class, which was already introduced above. Next to initializing raw AtCommands the *UdpControl* class is also responsible for pushing these commands to the AR.Drone. At this point the problem occurs, because these commands are blocked.

(6) Blocking Configuration Commands

```
meta.methods(AtCommandCreator).forEach(function(methodName) {
  UdpControl.prototype[methodName] = function() {
    var cmd = this._cmdCreator[methodName].apply(this._cmdCreator, arguments);
    if (cmd.blocks) {
      //console.log("BLOCKED: " + cmd.args);
      this._blockingCmds.push(cmd);
    } else {
      //console.log("PUSHED: " + cmd.args);
      this._cmds.push(cmd);
    }
    return cmd;
  };
});
```

Figure 31 UdpControl Function Reflection and Command Blocking

As mentioned above, the function from *Figure 21* is responsible for initializing *AtCommandCreator* objects of each incoming method name. The respective *AtCommandCreator* object then creates the corresponding AtCommand for the AR.Drone firmware, which is returned afterward to this function, seen in *Figure 21*. After creating the AtCommand and saving it in the *cmd* variable (comp. line 43, *Figure 21*), the *cmd* variable is checked if it blocks and is then either be added to the *_blockingCmds* array or pushed to the drone. In any case, all of our commands were classified as blocking commands and therefore they were not sent to the drone to be set.

After looking in different GitHub projects and email conversations with the developers of the *Ar-Drone* module, we still could not find a solution for sending unblocked commands to the drone.

3.2.3 Piloting View

This section is subdivided in sections for each functional area which are analog to the design and concept areas described in 3.1.3 Piloting View. Beside the client side implementation of each functionality the corresponding server side implementation is also documented here.

3.2.3.1 Video Stream

Supporting a resolution up to 720p (1280*720) the AR.Drone uses the standard video codec H264[27] which is described in detail in the developer guide attached to this document for further information. The Video stream transmission runs over a TCP websocket on port 5555 and starts immediately after a client is connected.[27] Rendering the H264 video frames in the browser is completely handled by the drone module "dronestream", which is enabling to stream the video into an HTML element using WebGL. First of all this module has to be included to the server:

```
// for video rendering
require("dronestream").listen(3001);

```

Figure 32 Dronestream Module Integration on Node Server

After including the "nodecopter-client.js", to be found in the "dronestream" module path, it just has to be initialized on the correct HTML element as shown in *Figure 33* below.

```

    <script src="../../../node_modules/dronestream/dist/nodecopter-client.js"></script>
    <script>
        $(function () {
            function startArDRoneStream() {
                new NodecopterStream(document.getElementById("dronestream"), {port: 3001});
            }
            startArDRoneStream();
        })
    </script>
    <div class="piloting_wrapper">
        <div id="dronestream"><i class="fa fa-spinner fa-spin flat"></i></div>
        <div id="command_console" class="flat">
            <section class="controls_center">
            <section class="controls_right">
                <div class="quick_commands_bar">
                <div class="analogs">
            </section>
        </div>
    </div>

```

Figure 33 Piloting View HTML Fragment

[27] comp. (i.a.) Piskorski, S. (2012), ll. 59

3.2.3.2 Command Console

Basically the command console is an absolute to the bottom positioned HTML "div" element with a height of 300 pixels. Its width is responsive to the screens width and has just the CSS rules to be at least 830 pixels wide, which is the minimum width for the console to look regular, and a maximum distance of 300 pixels to the left and right corners of the screen. The inner structure consists of two HTML section elements, one for the icons explained in the next subsection and one for the buttons in the quick command bar and the battery and altitude widgets. It also can be seen in *Figure 33.*

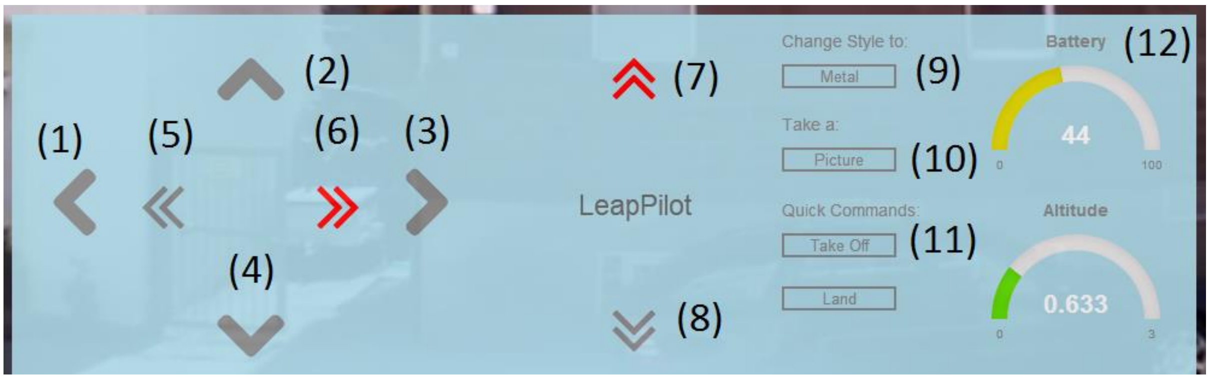
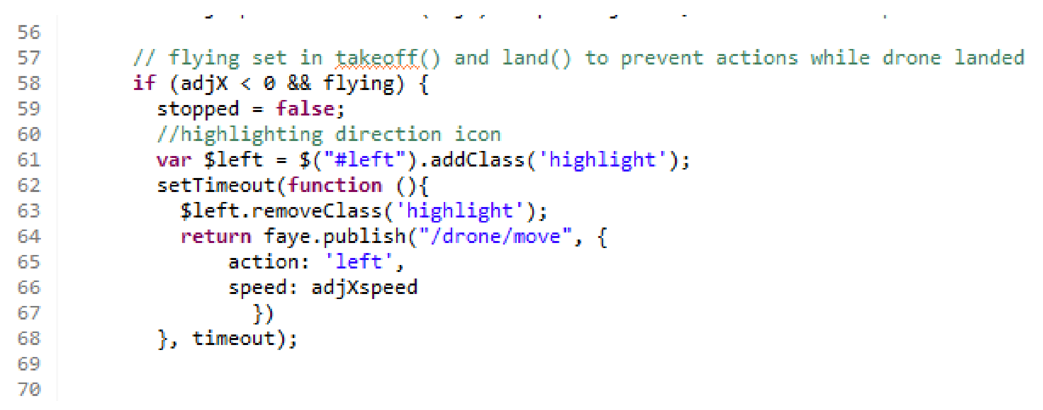


Figure 34 Command Console

3.2.3.3 Direction Icons

As already mentioned, the icons in *Figure 34* symbolize flying directions. While icons number 1 to 4 visualize relative or absolute flying directions, icons number 5 and 6 symbolize clockwise and counterclockwise rotation and icons 7 and 9 increase or decrease of the flight height. Since, flight commands can be sent by LEAP Motion via gestures and by the keyboard, both ways trigger the accentuation of corresponding icons by highlighting them via JavaScript and CSS.

The LEAP Motion gesture handling is done in the "leap.js" JavaScript file which is provided by the public git repository "leapdrone" by Daniel Liebeskind. In this file the highlighting of direction icons was modified.

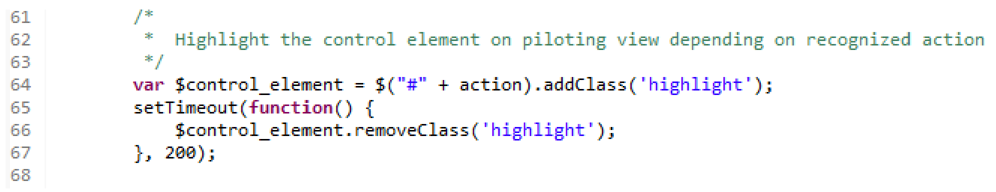


```
56
57        // flying set in takeoff() and land() to prevent actions while drone landed
58        if (adjX < 0 && flying) {
59          stopped = false;
60          //highlighting direction icon
61          var $left = $("#left").addClass('highlight');
62          setTimeout(function (){
63            $left.removeClass('highlight');
64            return faye.publish("/drone/move", {
65                action: 'left',
66                speed: adjXspeed
67                  })
68          }, timeout);
69
70
```

Figure 35 "leap.js" leftf-flight command handling

In *Figure 35, a snippet of "leap.js", the handling of a left-flight command is displayed. After checking for the condition of a left-command in line 58, the highlighting has to be set. This happens in line 61 by first selecting the icon from the HTML DOM tree with jQuery and adding the "highlight" class to that element. Afterwards, in the timeout function for sending the move command to the server for further processing, the CSS class "highlight" is removed from corresponding icon.*

The second file where highlighting happens is the "client.js" which handles keyboard inputs on the piloting view.



```
61          /*
62           *  Highlight the control element on piloting view depending on recognized action
63           */
64          var $control_element = $("#" + action).addClass('highlight');
65          setTimeout(function() {
66              $control_element.removeClass('highlight');
67          }, 200);
68
```

Figure 36 Generic highlighting

In this file, the keys are mapped to flight commands, like the key "w" to "front" etc. The difference in highlighting to "leap.js" is that here the "action" variable in line 64 in *Figure 36 is generic to what key is pressed. After being checked whether the key is mapped to a command, the icon gets highlighted and immediately the flight command sent.*

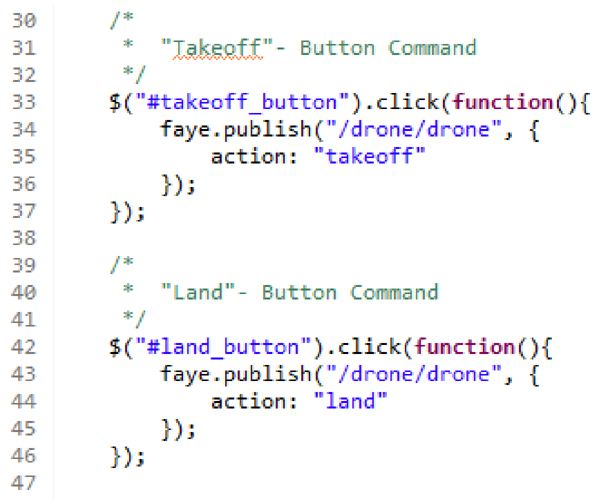
3.2.3.4 Quick Command Bar

First of all a style changer is to be found in the quick command bar, which is already described in 3.1.3 Piloting View. The functionality is implemented in "custom_piloting.js" which on the one hand handles the button functionalities in the quick command bar, on the other hand initialized and updates the battery and altitude widgets in the command console.

A JavaScript click event is attached to the buttons and in the case of the style changer button the event causes the change of several CSS classes in HTML elements like the command console to change its design.

The save picture button currently gets a base64 encoded image source, which seems not to be easy to view. The image gets saved but can't get displayed properly. Regarding this issue, the Parrot Video Encapsulation (PaVE) could be interesting to debug regarding encapsulation of frames for further de-encapsulation.

Below the picture button, there are "Take Off" and "Land", direct flight commands to the drone. As the correspondent gestures were mostly very inconsistently recognized, the main idea of the quick command bar was to enable a chance of direct commanding basics like taking off and landing.



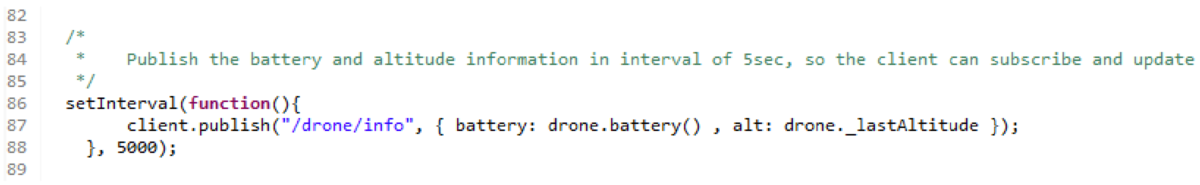
```
/*
 *  "Takeoff"- Button Command
 */
$("#takeoff_button").click(function(){
    faye.publish("/drone/drone", {
        action: "takeoff"
    });
});

/*
 *  "Land"- Button Command
 */
$("#land_button").click(function(){
    faye.publish("/drone/drone", {
        action: "land"
    });
});
```

Figure 37 "Take Off" and "Land" Button implementations

The basic communication between server and client is running threw a publish – subscribe mechanism implemented by the AR.Drone module "faye". It enables publishing data with an identifier string and subscribing it from client to server or the other way around. In this context, the client is publishing an action, namely "land" or "takeoff". The server subscribing to the string "/drone/drone" handles this action in sending it to the drone.

3.2.3.5 Battery and Altitude Gauges

As mentioned in the subsection before, the server and client are communication over the module “faye”. The server has direct access to the drone and is able to get its current battery percentage and altitude. These information are pulled in an interval of 5 seconds and published, as it can be seen in ***Error! Reference source not found.***. With these information the gauges are initialized and afterwards updated.



```

/*
 *    Publish the battery and altitude information in interval of 5sec, so the client can subscribe and update
 */
setInterval(function(){
      client.publish("/drone/info", { battery: drone.battery() , alt: drone._lastAltitude });
  }, 5000);

```

Figure 38 Server publishing battery and altitude information

3.2.3.6 Video capturing

Since, the video capturing blocks video streaming, this functionality currently is in development and for delivery disabled. The H264 encoded video stream of AR.Drone can be extracted by parsing it and removing the Parrot Video Encapsulation (PaVE). This already works fine and a “.h264” video file is generated, viewable in a video player providing this file format. For capturing the video a TCP connection has to be made to the video port of the drone. As the “dronestream” module also tries to connect to the same socket a problem occurs. The socket seems to enable only one connection so either the video stream in the piloting view is enabled, or video capturing. This technical bug could be fixed in further releases of this project.

3.2.3.7 Media Library View

Since, media capturing problems described in *3.2.3.4 Quick Command Bar* and *3.2.3.6 Video capturing*, the media library view is still in planning. The idea of implementation is to create a path where all media created is stored and display them as tiles in this view.

4 Outlook/ Reflection

Within eight month, we reviewed various different areas of application of the LEAP Motion Controller until we finally decided to create a natural user interface for piloting an AR.Drone. Since, there is a wide range variety for the use of gesture recognizing hardware we spent a reasonable amount of time looking for an appropriate study work project. Next to other project ideas, we shortly conceptualized an approach for a gestural based operating system controller. After consultation with our project representative, we got the opportunity to integrate an AR.Drone in our project ideas.

Subsequently, we quickly decided to develop a future-oriented user interface that implements natural interfaces like gestural and speech recognition for piloting and configuring the AR.Drone. Nonetheless, the interface is still limited to its hard- and software recognition barriers as for instance the gestural limits of the LEAP or Annyang's weaknesses according to naturally speech misunderstandings. Further improvements referring to the recognition can therefore be achieved by improving the recognition algorithms or changing the way of interaction.

During our more practical focused time for implementation, we had inconveniently one pair of LEAP Motion Controller and AR.Drone which made it difficult to work together, since we could work apart as virtual team. Because of that implementation work often ended up in one testing code for the other, which is a highly ornate way of collaboration.

In this connection, problems occurred lately during our development process, which affected our final results tremendously. Technical issues, as described in *Problem of Configuration Transmission*, led us to reconsider our time management. Although, we spent a large amount of time in finding potential solutions and consulting the "*ar-drone*" – module developer we still did not find a way for solving the *Problem of* Configuration Transmission.

Last but not least, we could show easy implementable ways for the next generation of user interfaces that offer great variety in recognizing human-oriented ways of interaction. Besides, we could create a base for further projects in the area of Leap Motion Controller and AR.Drone combination. In this context, next versions could solve current problems. Moreover upcoming versions may include functionality for a social network integration in means of sharing media data.

5 List of Sources

Index	**Source**
1	comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* URL: *http://www.cs.rutgers.edu/~vladimir/pub/pavlovic97pami.pdf* (last visited: February 28, 2014)
2	comp. Engadget (May 21st 2012): *Leap Motion reveals super-accurate motion control […]* URL: *http://www.engadget.com/2012/05/21/leap-motion-3d-motion-and-gesture-control/* (last visited: February 27, 2014)
3	TechTerms (July 26, 2012): *Definition of NUI* URL: *http://www.techterms.com/definition/nui* (last visited: March 6, 2014)
4	Youtube (March 21, 2010): *Microsoft Research's Bill Buxton on Natural User Interfaces*, URL: *http://www.youtube.com/watch?v=NcdrfacG_y4* (last visited: March 6, 2014)
5	Provita Labs Alex Chong (August 14, 2013): *The Future of UI/UX: The Natural User Interface* URL: *http://pivotallabs.com/the-future-of-uiux-the-natural-user-interface/* (last visited: April 22, 2014)
6	Bresslergroup (unknown): *Interaction Design, Natural User Interfaces*, URL: *http://www.bresslergroup.com/expertise/interaction-design/natural-user-interface* (last visited: March 6, 2014)
7	Ergo-Online (October 09, 2008): *Software-Ergonomie und Benutzerfreundlichkeit* URL: http://www.ergo-online.de/site.aspx?url=html/software/grundlagen_der_software_ergon/software_ergonomie.htm (last visited: April 10, 2014)
8	Interaction Design Foundation (August 26, 2013): *Fitts's Law by Mehmet Göktürk* *URL:* http://www.interaction-design.org/encyclopedia/fitts_law.html (last visited: April 10, 2014)
9	comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* p. 679 URL: *http://www.cs.rutgers.edu/~vladimir/pub/pavlovic97pami.pdf* (last visited: April 15, 2014)
10	D. Kammer, M. Keck, G. Freitag, M. Wacker (unkown): *Taxonomy and Overview of Multi-touch Frameworks: Architecture, Scope and Features* URL: *http://vi-c.de/vic/sites/default/files/Taxonomy_and_Overview_of_Multi-touch_Frameworks_Revised.pdf* (last visited: March 30, 2014)
11	comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* p. 680 URL: *http://www.cs.rutgers.edu/~vladimir/pub/pavlovic97pami.pdf* (last visited: April 15, 2014)
12	comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* p. 681 URL: *http://www.cs.rutgers.edu/~vladimir/pub/pavlovic97pami.pdf* (last visited: April 15, 2014)
13	wisee (unknown): *What is WiSee?* URL: *http://wisee.cs.washington.edu/* (last visited: March 30, 2014)

14	StiltSoft (May 2, 2013): *Google Chrome: How to Use the Web Speech API* URL: *http://stiltsoft.com/blog/2013/05/google-chrome-how-to-use-the-web-speech-api/* (last visited: April 15, 2014)
15	comp. Julius (2014): *Open-Source Large Vocabulary CSR Engine Julius* URL: *http://julius.sourceforge.jp/en_index.php* (last visited: April 15, 2014)
16	comp. Pocketsphinx.js Speech Recognition in JavaScript (unknown): *Welcome to PocketSphinx.js* URL: *http://syl22-00.github.io/pocketsphinx.js/* (last visited: April 15, 2014)
17	comp. annyang! (unknown): *SpeechRecognition that just works* URL: *https://www.talater.com/annyang/* (last visited: April 15, 2014)
18	comp. annyang! (unkown): *annyang! SpeechRecognition that just works* URL: *https://www.talater.com/annyang/* (last visited: May 15, 2014)
19	comp. Leap Motion (unknown): Touchless For Windows URL: https://airspace.leapmotion.com/apps/touchless-for-windows/windows (last visited: June 3, 2014)
20	comp. Forum Leap Motion (May 8, 2012): *The unofficial Leap FAQ* URL: *https://forums.leapmotion.com/forum/general-discussion/general-discussion-forum/434-the-unofficial-leap-faq?420-The-unofficial-Leap-FAQ* (last visited: February 27, 2014)
21	comp. Parrot (2008): Legalmentions URL: http://www.parrot.com/de/legalmentions/ (last visited: June 1, 2014)
22	(i.a.) Piskorski, S. (2012): AR.Drone Developer Guide URL: https://abstract.cs.washington.edu/~shwetak/classes/ee472/notes/ARDrone_SDK_1_6_Developer_Guide.pdf (last visited: June 7, 2014)
23	P. Stephane, B. Nicolas, E. Pierre, D. Frederic (May 12, 2012), *AR.Drone Developer Guide* URL: *http://www.msh-tools.com/ardrone/ARDrone_Developer_Guide.pdf* (last visited: June 7, 2014)
24	Font Awesome (unknown): The Iconic Font And CSS Framework URL: http://fortawesome.github.io/Font-Awesome/ (last visited: June 8, 2014)
25	JustGage (unknwown) URL: http://justgage.com/ (last visited: June 8, 2014)
Cover Page Image 1	Melelectronics (unknown): *Leap Motion LM-010* URL: *http://www.melectronics.ch/medias/sys_master/hc0/hae/8937145368606/Leap-Motion-Sensor-02-700x350.jpg* (last visited: March 4, 2014)
Cover Page Image 2	Official Leap Motion (unknown): *Leap Motion Logo* URL: *https://www.leapmotion.com/* (last visited: March 4, 2014)
Cover Page Image 3	Amazon (unknown): *Parrot AR.Drone 2.0* URL: *http://ecx.images-amazon.com/images/I/81RNYV29HCL._SL1500_.jpg* (last visited: March 4, 2014)
Figure 1	comp. http://instructionaldesignfusions.wordpress.com/ (November 15, 2010): *Evolution of the user interface*

	URL: http://instructionaldesignfusions.files.wordpress.com/2010/11/nui-evolution2.png (last visited: April 10, 2014)
Figure 2	comp. S. Taranko & M. Keck (May 13, 2011): *Next Generation Interfaces: Das Meta-Visualisierungstool DelViz* URL: *http://vi-c.de/vic/sites/default/files/2011-05-13_nextgenerationinterfaces.pdf* (last visited: April 15, 2014)
Figure 3	comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* p. 680 URL: http://www.cs.rutgers.edu/~vladimir/pub/pavlovic97pami.pdf (last visited: April 15, 2014)
Figure 4	comp. IEEE Transactions on pattern analysis and machine intelligence (July 7, 1997): *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review* p. 681 URL: http://www.cs.rutgers.edu/~vladimir/pub/pavlovic97pami.pdf (last visited: April 15, 2014)
Figure 5	thevoltreport.com (September 22, 2013): *Wi-Fi network utilized to recognition human gestures to operate the gadgets* URL: *http://www.thevoltreport.com/wp-content/uploads/2013/11/gesture-control-technology.jpg* (last visited: March 30, 2014)
Figure 6	Leap Motion (unknown): Touchless For Windows URL: https://airspace.leapmotion.com/apps/touchless-for-windows/windows (last visited: June 3, 2014)
Figure 7	Photobucket (unknown): *joe_limon's Bucket* URL: *http://smg.photobucket.com/user/joe_limon/media/fov_zps9bccb280.jpg.html* (last visited: March 4, 2014)
Figure 8-10	(i.a.) Piskorski, S. (2012): AR.Drone Developer Guide URL: https://abstract.cs.washington.edu/~shwetak/classes/ee472/notes/ARDrone_SDK_1_6_Developer_Guide.pdf (last visited: June 7, 2014)