Pós-Graduação em Ciência da Computação

LUCAS SILVA FIGUEIREDO

A DESIGN METHOD FOR BUILDING IN-AIR GESTURAL INTERACTIONS

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

Lucas Silva Figueiredo

A Design Method for Building In-air Gestural Interactions

A Ph.D. Thesis presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science.

ADVISOR: **Veronica Teichrieb**

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

F475d Figueiredo, Lucas Silva
A design method for building in-air gestural interactions / Lucas Silva Figueiredo. – 2017.
139 f.:il., fig., tab.

Orientadora: Veronica Teichrieb.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências.

1. Ciência da computação. 2. Computação gráfica. I. Teichrieb, Veronica (orientadora). II. Título.

004 CDD (23. ed.) UFPE- MEI 2017-168

Lucas Silva Figueiredo

A Design Method for Building in-air Gestural Interactions

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação

Aprovado em: 23/02/2017.

_____

Orientadora: Profa. Dra. Veronica Teichrieb

BANCA EXAMINADORA

_____

Prof. Dr. André Luis de Medeiros Santos
Centro de Informática / UFPE

_____

Prof. Dr. Geber Lisboa Ramalho
Centro de Informática / UFPE

_____

Prof. Dr. João Marcelo Xavier Natario Teixeira
Departamento de Eletrônica e Sistemas/ UFPE

_____

Prof. Dr. Thales Miranda de Almeida Vieira
Instituto de Matemática / UFAL

_____

Prof. Dr. Pedro Martins Alessio
Departamento de Design / UFPE

Acknowledgements

Agradeço especialmente a Yvonne pela atenção, carinho, paciência e a força durante esses anos de companheirismo.

Ao meu pai, sempre me acompanhando mesmo que à distância. À minha mãe, que de formas que não sou capaz de descrever, moldou o que sou. A Vera, Caio, Ana e Clara que trazem energia tão boa e essencial à minha vida. À toda minha família, avós e avôs, tias e tios, primas e primos.

Aos meus amigos sempre mostrando que diversão é uma parte essencial da vida.

À minha orientadora, professora e amiga Veronica Teichrieb, sempre mostrando um caminho, uma opção sobre como resolver os desafios com os quais nos deparamos no dia a dia.

Ao Voxar Labs. É um prazer ajudar a construir esse time, aprendo todos os dias e confesso que pra mim o conceito de "trabalho" está longe do senso comum por conta de vocês. Em especial, agradeço a Chico, Joma, Jonga e Rafael pelo trabalho em conjunto que desenvolvemos ao longo dos anos, boa parte da responsabilidade por me identificar como pesquisador hoje é de vocês. Também agradeço a Mari, Edvar, Thiago e Alana pelas contribuições diretas neste trabalho.

Por fim agradeço a David Molnar, Ben Livshits e Margus Veanes por me mentorarem de forma extramamente objetiva e com alta produtividade.

Abstract

In-air gestures are part of our everyday communications. Giving a “thumbs up,” pointing to an object of interest or raising a hand to call for attention are just a few examples. Current vision-based technologies such as the Microsoft Kinect, the Leap Motion have shown real-time tracking capabilities that enable a large range of developers and companies to explore these gestures on human-computer interactive solutions. We approach the process of building these interactions. Our efforts are to understand and aid designers, developers, and researchers in the field of Human-Computer Interaction (HCI) to build in-air gestural interfaces. With that goal in mind, we provide a set of techniques and tools to explore and prototype concepts of possible gestural interactions for a given target task. We divide the toolset into two main phases: the conception of the gestures to be used; and the prototype of solutions using these gestures. For the conception phase, we propose the use of a set of creative techniques. Moreover, we introduce a pair of web catalogs (as tools) to be used for analysis and inspiration while suggesting and creating new gestural interactions. By reviewing the literature regarding how researchers define the used gestures, we cataloged several examples according to a developed taxonomy. We also performed a similar study and built a catalog of in-air gestures present on Science Fiction (Sci-Fi) content. Sci-Fi contents, although not representing real interfaces, show potential while exploring innovative concepts that can influence the creation of new interfaces. For the prototype phase, we focused on the steps of producing and testing low-fidelity and high-fidelity prototypes for the recognition of the conceptualized gestures. For low-fidelity prototyping, we propose and validate the use of the Wizard of Oz technique, which enables fast testing of different concepts. For high-fidelity prototyping, we introduce a recognition tool called Prepose, which aims the easiness of use for creating and editing the gesture recognizers. Prepose allows a gesture to be written in natural language making it easy for developers and non-developers to read, write and edit the target gestures. At the same time, Prepose allows a gesture to be automatically written with one sample of its execution, speeding up the time to build recognizers for complex gestures. At last, we also conducted a pilot study to demonstrate the use of the toolset. In this study, we generated a set of 32 interaction concepts that was incrementally reduced while using the proposed techniques for selection and prototyping. In the end, the application of the toolset resulted in a high-fidelity prototype for the best-evaluated interaction concept.

Keywords: Gestural input. Systems and tools for interaction design. Science-Fiction. Interface design prototyping. Domain specific languages.

Resumo

O uso de gestos ao ar (*in-air gestures*) é parte do cotidiano da comunicação entre seres humanos. Levantar o polegar para expressar concordância, apontar para um objeto de interesse ou levantar a mão para atrair a atenção são somente alguns exemplos. Tecnologias atuais baseadas em visão computacional como o Microsoft Kinect e o Leap Motion demonstram a capacidade de restrear o usuário em tempo real, habilitando desenvolvedores e empresas a explorar gestos como forma de interação entre seres humanos e máquinas. Neste trabalho nós abordamos o processo de construir tais interações. Nossos esforços são direcionados a entender e facilitar o processo de construção de interfaces gestuais para designers, desenvolvedores e pesquisadores do campo de Interação Humano-Computador. Com este objetivo em mente, nós introduzimos um conjunto de técnicas e ferramentas, para explorar e prototipar conceitos de possíveis interações gestuais para determinada tarefa. Este conjunto é dividido em duas fases principais: a concepção dos gestos a serem utilizados; e a construção de protótipos destes gestos. Para a fase de concepção nós propomos o uso de um conjunto de técnicas criativas. Em adição, nós introduzimos dois catálogos web para serem usados para análise e inspiração durante a criação de novas interações gestuais. Ao revisar a literatura relativa ao uso de gestos por pesquisadores, nós catalogamos diversos exemplos do uso de gestos. Realizamos uma revisão similar (também apresentada em um catálogo web) do uso de gestos em conteúdos de Ficção Científica. Estes conteúdos, apesar de não representarem interfaces reais, apresentam potencial ao explorar conceitos inovadores que podem influenciar a criação de novas interfaces. Para a fase de construção do protótipo nós focamos nas etapas relativas à produção e testes de protótipos de baixa e alta fidelidade. Para os protótipos de baixa fidelidade nós propomos o uso da técnica do Mágico de Oz, a qual permite testar rapidamente diversos conceitos. Para a prototipação de alta fidelidade nós apresentamos uma ferramenta para reconhecimento de gestos chamada Prepose. Prepose permite que cada gesto seja escrito em linguagem natural (usando o Inglês como idioma) tornando assim a atividade de construir e editar reconhecedores de gesto acessível para desenvolvedores e não-desenvolvedores. Ao mesmo tempo, Prepose permite que um gesto seja transcrito automaticamente através de um treinamento que requer uma única execução, acelerando o processo de construir reconhecedores de gestos complexos. Por fim, conduzimos um estudo piloto para demonstrar o uso das técnicas e ferramentas propostas. Neste estudo, geramos um conjunto de 32 conceitos de interações gestuais que foi incrementalmente reduzido através das técnicas de seleção e prototipação propostas. Ao fim, obtivemos um protótipo de alta-fidelidade com o conceito de interação mais bem avaliado.

Palavras-chave: Controle gestual. Sistemas e ferramentas para design de interação. Ficção-Científica. Prototipação de interfaces. Linguagens específicas de domínio.

List of Figures

List of Tables

List of Acronyms

AAOS	American Academy of Orthopedic Surgeons	86
AR	Augmented Reality	38
BNF	Backus-Naur Form	88
CAVE	Cave Automatic Virtual Environment	21
CDF	Cumulative Distribution Function	96
DF	Decision Forest	100
DIY	Do It Yourself	17
DSL	Domain-Specific Language	26
DT	Decision Tree	100
DTW	Dynamic Time Warping	100
FUI	Future User Interface	62
HCI	Human-Computer Interaction	15
HMM	Hidden Markov Models	100
IDE	Integrated Development Environment	88
NB	Naïve Bayes	100
NN	Nearest Neighbor	100
NUI	Natural User Interface	15
OS	Operating System	16
RF	Random Forest	100
Sci-Fi	Science Fiction	25
SDK	Support Development Kit	18
SMT	Satisfiable Modulo Theories	26
SVM	Support Vector Machines	100
UX	User eXperience	19
VR	Virtual Reality	76
XML	eXtensible Markup Language	83

Contents

1 Introduction

Gestures are an ancient form of communication among humans. As quoted by Schebesch found “figurines from the Upper Paleolithic display certain postures: they have body language” (SCHEBESCH, 2014). They are used on interactions as an expression resource, communicating explicit messages (e.g. while pointing to a particular subject in the space), or implicit messages (e.g. using beat gestures, rhythmically move hands synced to speech to emphasize a particular topic) (MCNEILL, 1992). Humans can use gestures as an alternative to spoken languages, mapping signs to existent words, or in a more abstract way on artistic performances such as mimicry or dance. Different parts of the body can be explored, such as hands, arms, the whole body or the face for example. The particular case of *in-air* (or mid-air) gestures refers to motions in space, not related to physical interactions with the environment such as touching or grasping objects.

A technology capable of recognizing these motions empowers itself with a similar understanding humans have about each other’s body language and body activities. That the human inner-knowledge, inherited from cultural and physiological background goes through the technological barrier. Several research fields tackle this goal such as Natural User Interfaces (NUIs), Eyes, Face, Hand and Body Tracking, Gesture and Activity Recognition, and so on. Each one of these areas investigates a particular challenge within this larger goal.

1.1 Context

In 1980, after companies such as Intel, Apple, IBM and others launched the Personal Computers (PCs) a significant change concerning the use of computers occurred. Figure 1.1 part A illustrates the popularization of PCs over the years, which started to be present in work and home environments. This context favored the consolidation of the research field of Human-Computer Interaction (HCI). The increased use of PCs boosted the need for new research focused on the users, and on how they perceived and acted towards the machines. Studies arose regarding topics like intuitive graphical interfaces, and precise input devices with faster response times. The concept of “human factors” was then used to give the name of one of the most important

HCI existing conferences, the Conference on Human Factors in Computer Systems, also known as CHI.

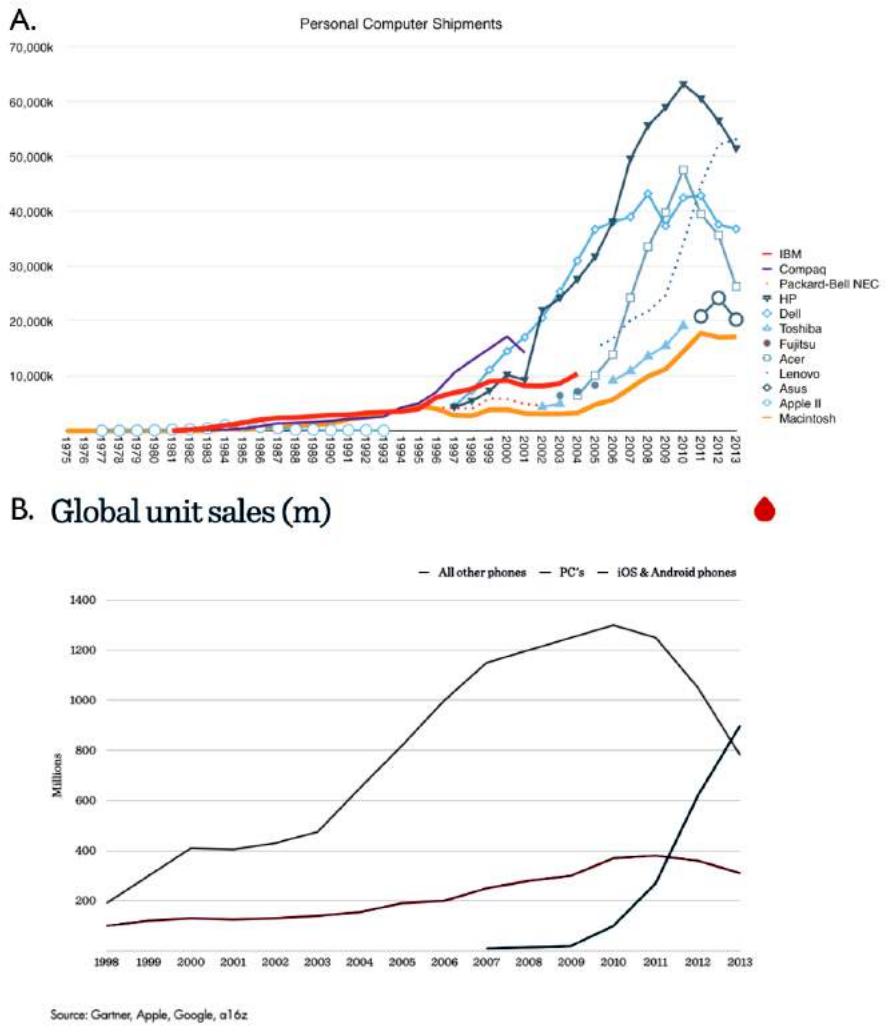

Figure 1.1: On part A, the number of sales per year of each PC model (in millions of units) between 1975 and 2013 (ASYMCO, 2015). On part B, the sales of smartphones (in millions of units) compared to the sales of PCs and other types of phones (PERCOLATE, 2014).

The current context presents another boost of popularized systems, as shown by the examples of increasing sales of mobile devices and laptops, illustrated on Figure 1.1 part B. Additionally a new range of devices is more present on the market. Tablets and detachable laptops, such as Microsoft Surface, fill the gap of medium screen sizes between laptops and smartphones. Nowadays TVs can be considered as fully capable systems with embedded Operating System (OS). Wearables include smartwatches and bracelets. Computers are increasingly surrounding users as time passes, and the range of users broadens.

Given this context, the topic of Natural Interaction emerges as a direction towards more intuitive and efficient interactions. Therefore, while assuming *users can not afford to learn from the scratch how to handle each one of the amounts of different devices and systems*, this context

demands such direction. Base interaction strategies, metaphors, methods, and tools can be useful while developing new interfaces.

1.1.1 Natural Interaction

The definition of Natural Interaction can be broad; in this work we consider the description given by Alessandro Valli (2005) which states "People naturally communicate through gestures, expressions, movements. Research work in natural interaction is to invent and create systems that understand these actions and engage people in a dialogue while allowing them to interact naturally with each other and the environment. People do not need to wear any device or learn any instruction, interaction is intuitive. Natural interfaces follow new paradigms in order to respect human perception. Interaction with such systems is easy and seductive for everyone". Valli evidences the change of roles; *systems should understand users, not the opposite*. Particularly, he points gestures as an evidence of human behavior to be understood by systems.

1.1.2 In-air gestures technological background

The context also favors the development of science and technologies over gestural interaction. Nowadays devices and algorithms are reaching the point to allow precise real-time tracking of body motions on consumer level hardware, allowing the exploration of gestural applications in a broad way.

Areas such as machine learning, parallel programming, computer vision, hardware miniaturization, cloud computing, pattern recognition, and augmented reality have shown outstanding development in the last few years towards real-time interactive solutions. The effort has come not only from science fields, science labs or researchers in general but also from the high-end industry. Companies such as Google and Microsoft are leading academic research related to gestural interaction (GOOGLE, 2015; SHARP et al., 2015).

The maker community also plays a significant role in the rise and popularization of new solutions. Forums like the NUI Group (2012) expanded the knowledge and interests of developer community towards the discovery and experimentation of Do It Yourself (DIY) versions of multi-touch systems (GROUP, 2008). Since the launch of the first Microsoft Kinect device (MICROSOFT, 2010) *real-time depth sensors* have been broadly explored and similar solutions have been shown by Asus Xtion (ASUS, 2012) and Intel RealSense F200 (INTEL, 2015a). In this context, small companies are born, based on differentiated products such as the Leap Motion (MOTION, 2013).

Figure 1.2 shows some examples of depth sensors available at low retail prices. The range of device niche is broad, examples of categories are: standalone (Xtion Pro Live) and embedded depth sensors (Project Tango); short range (Leap Motion) and mid-range (Microsoft Kinect V1 and V2); attachable design for the front (Intel SR300) or else for the rear (Intel R200) of the processor device.

Figure 1.2: Samples of depth sensors available on market.

Depth sensors like the ones previously mentioned are capable of retrieving real-time bi-dimensional discrete data, assigning for each unit (or pixel) the depth information of that part of the scene. For interactive purposes, this information is highly valuable because it empowers tracking systems with 3D info about the environment, which is harder to acquire with standard color (RGB) cameras.

Firstly, the depth information allows an easier segmentation of foreground and background. This way, tracking systems can quickly discard environment data and focus on the user body (or body parts). The depth information can also be used to understand the shape of the foreground elements, aiding the task of inferring whether or not there is a human, and if there is, how she poses her body. For example, Shotton and colleagues (2013) demonstrated the use of the depth information for training to later identifying the user body posture by detecting some of the skeleton joints such as elbows, hips, and knees.

Usually, each depth sensor comes coupled with a tracking solution in its Support Development Kit (SDK). Microsoft Kinect SDK 2.0 (MICROSOFT, 2015) offers a full body tracking solution, retrieving a skeleton containing 25 joints representing the user body, as well as a face tracking solution. Leap Motion is, in its turn, a short-range depth sensor and its SDK (LEAP, 2015) focuses on tracking hands, retrieving a skeleton of the hand. The Intel RealSense SDK (INTEL, 2015b) provides skeleton tracking of both face and hands.

These solutions provide real-time skeleton data of the tracked body parts. Also, they require no attached devices, once they are vision based solutions. Therefore, these vision-based solutions present an advantage from the viewpoint of the user freedom of motion, as discussed by Valli (2005).

1.1.3 Inspiration and concepts development

Once these new devices are available worldwide at low-cost, new interaction possibilities arise for several developers, researchers, interaction designers, and companies.

Alongside the technological challenges, there are also ideation challenges. The creation of new interfaces based on in-air gestures brings a considerable amount of decisions on why and how these interfaces should use the user postures and motions. The interaction patterns established to WIMP systems (abbreviation to windows, icons, menus and pointers), such as the ability to navigate through windows, were polished through the years to a particular interaction environment. These patterns may not apply to emerging technologies in the given context.

Multi-touch and stylus-based interfaces are going through a similar maturation process. In part, the interactions on these interfaces inherit from previous patterns, mapping touch actions on mouse events. In part, new concepts arise, e.g. swiping in or out the edge of a device screen to change or dismiss applications, or even touching the back of the screen as shown on the Playstation Vita (LLC, 2016). By its turn, in-air gestural interactions are newer to both the HCI research community and to the market, which brings more challenges on how to properly approach and explore the related technologies from a User eXperience (UX) viewpoint.

1.2 Motivation and research goals

Given the context, the motivation arises towards the conception, prototyping[1], and development of in-air gestural interfaces. More specifically, we direct our thoughts to the challenges imposed to developers and interaction designers that are assigned or self-driven to build these interfaces.

Once depth sensors, and therefore real-time skeletal trackers, reach the market at affordable pricing, we observe an increasing action of developers and researchers to endeavor towards using these devices and algorithms on their projects and research. As an evidence of this growth, Figure 1.3 shows results of the search for the depth sensor “Microsoft Kinect” on GitHub and Google Scholar, for each year from the device launch in 2010 until the year of 2015.

1.2.1 Motivational examples

As a starting point for the following discussion, we would like to present two motivational examples. The first example is a gestural command used in a video game called *Fighters Uncaged*

[1] According to Martin and Hanington (2012), prototyping is the “creation of artifacts at various levels of resolution, for development and testing of ideas within design teams and with clients and users.” Low-fidelity prototypes represent roughly built artifacts, which are not required to fully work as expected by the final solution. For instance, a low-fidelity prototype of a computational system may not include a single line of code. On this case, the low level of fidelity is intentional, meant to allow rapid tests of different concepts for the desired solution. By its turn, high-fidelity prototypes are working versions of the defined concept for the system. A high-fidelity prototype for in-air gestures interfaces should be capable of automatically recognize the input gestures and provide the planed output.

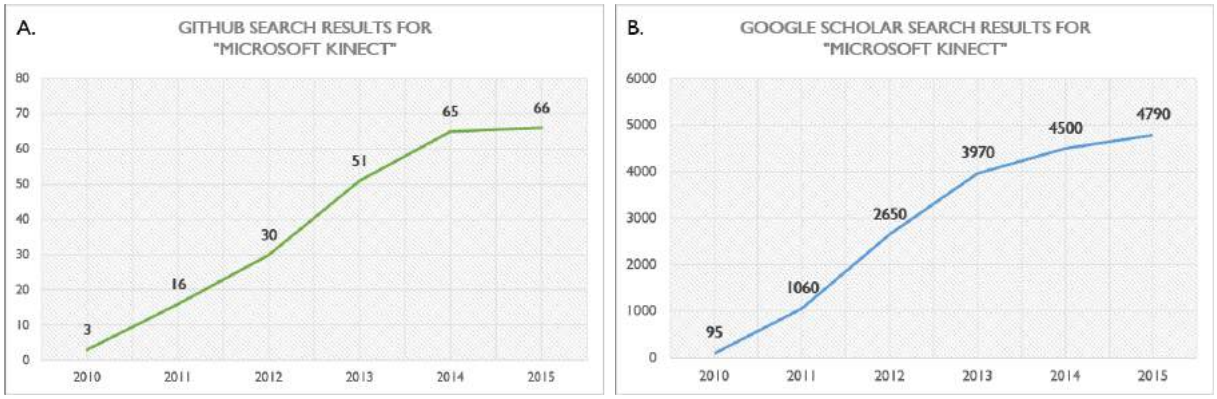

Figure 1.3: On part A, the number of active GitHub projects per year as result of a search for "Microsoft Kinect". On part B, the same search applied to published academic papers found by Google Scholar.

(UBISOFT, 2010). Published by Ubisoft in 2010, the game targets the Microsoft Xbox 360 console and entirely bases its interface on the first version of the Microsoft Kinect depth sensor (Microsoft Kinect V1). The game allows the user to control an avatar by performing specific body movements mapped on fighting moves. While testing the game on the tutorial mode, we noticed, as users/players, issues on the interaction such as delays, plus unmatched (false-negatives) and mismatched (false-positives) movements. In a review performed by IGN the game scored 3 out of 10, and among the pointed issues, the reviewer mentioned its "Unresponsive motion controls" (IGN, 2010).

The issues mentioned above are related to the game built-in gesture recognition. Also, we found some issues related to counter-intuitive choices of mapping between user and avatar movements, as shown in Figure 1.4. In this particular case, the user is asked to perform an upwards punch gesture (green arrow) while the avatar response is a downwards circular punch move (red arrow). For us, while playing the game, this mismatch between execution and feedback proved to be counter-intuitive, requiring additional mental effort to remember which gesture would trigger that movement, and thus increasing the cognitive load of the interaction.

Figure 1.4: Interaction snapshot of *Fighters Uncaged* showing the mismatch between the user movement and the avatar feedback.

The second example, shown in Figure 1.5, is a gestural interface built to allow users to navigate in 3D environments using a Cave Automatic Virtual Environment (CAVE) as visual output platform, and the Microsoft Kinect V1 as the input sensor. To navigate, the user performs specific body poses to activate commands such as *move forward*. The interface is a result of a project from the University of Michigan 3D Lab (UM3DLab) called Kinect - Navigation in a Virtual Reality CAVE (MICHIGAN 3D LAB, 2013).

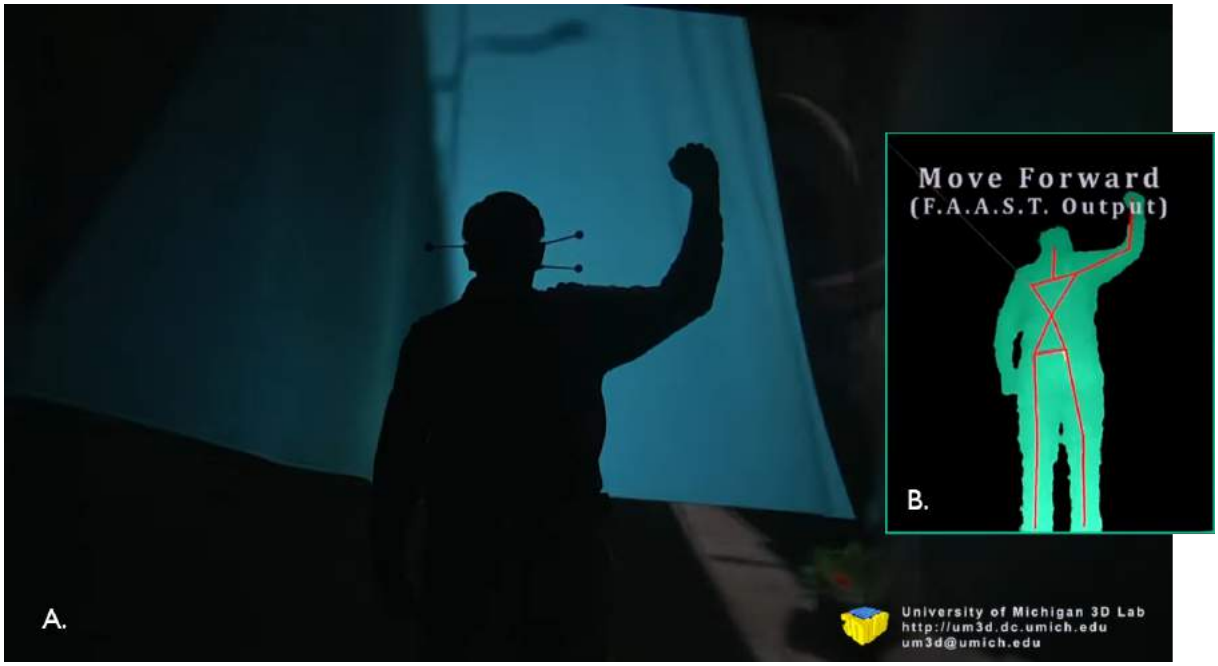

Figure 1.5: On part A snapshot showing the user on a CAVE, navigating in a 3D environment using gestures. On part B the recognition result of the current gesture is shown.

In this case, the command *move forward* requires the user to point up her right forearm, with her right arm aligned with her shoulders as shown in Figure 1.5. This body posture is easily recognizable by the Kinect because it is free from occlusions, being all body parts easily visible from the sensor viewpoint. On the counterpart, this gesture presents issues for the user for being counterintuitive by associating different directions (point up to move forward). Another point is that this gesture is similar to another commonly depicted as a gesture of tactical operations meaning *Hold* or *Freeze*, in opposition to the desired concept of *Move*. The last pointed issue is the fatigue. On pilot-tests, while prototyping a similar interface, we found this suggested gesture to be uncomfortable, and to move forward in a 3D environment is a highly frequent and sustained task, which would produce fatigue within a few minutes of interaction.

1.2.2 Research problem and directions

On our initial investigation regarding existing gestural interfaces, on both industry and academy, the examples mentioned above appeared as warnings related to the challenges ahead of developers and researchers. There are challenges on the gesture recognition side, as pointed

by the false-negatives experienced on *Fighters Uncaged*. Also, there are challenges related to the definition of which gestures are used to control the target tasks, as discussed on the *CAVE Navigation System*.

In addition to the cases shown on the previous section, we also observed a negative reaction on the online media regarding Kinect-based games. On an article from Business Insider (https://goo.gl/XQ1bcq), Matt Weinberg discusses about "The downfall of Kinect" and comments about both the adoption of the challenge of building gestural interactions by the developers as well as the "lack of solid titles" on these cases:

> *"Developers started to line up to make games for the device, too, with 17 available at launch, including 'Kinect Adventures,' a Microsoft-made game that came packaged with the Kinect sensor. Most of those games were panned by reviewers: 'Critics are complaining about a lack of solid launch titles for the new control system; only 'Dance Central' seems to have anything to recommend it,' said a Metacritic roundup of launch titles at the time."*

In addition, on an article from Techradar (https://goo.gl/hF0v6w), Matt Swider talks about the Kinect-based games for Xbox One and points that "Its one good game took forever to come out." On another article from GAMERANT (https://goo.gl/MIhD9W), Christian Spicer states that on "Microsoft E3 (2011) press conference featured sequel after sequel after sequel. The only real new or original games shown were Kinect titles – and they all looked pretty awful ..." Although these articles do not state research evidences, they point the impression of the media regarding failures on Kinect-based games that are linked to the use of in-air gestural interaction.

In fact, in-air gestural interfaces present additional challenges yet not dealt on interactions based on mouse, keyboards or multi-touch devices. The use of the space around the user now appears as an additional concern. Given that the tracking sensor may be positioned on different parts of the environment, and be even hidden, the binding between user and sensor, that was previously well defined for touch, mouse and keyboard, now can be diverse. For these interfaces the user can provide input even without acknowledging the existence of a sensor or a system. At last, in-air gestures also present a significant difference regarding the interaction engagement. On mouse and keyboard the engagement requires the user to touch and then press a button. For multi-touch screens the user only needs to touch the screen to engage and start the interaction. For in-air gestures the user is always being tracked and interpreted, the engagement occurs once the user reaches the field of view of the device and this difference challenges the interface regarding the exclusion of user movements that were not intended to mean any kind of interaction with the system. These and other challenges and pitfalls while building in-air gestural interfaces can be found on the book "Brave NUI World," by Daniel Wigdor and Dennis Wixon (2011).

It is natural to expect a knowledge gap for breakthroughs on HCI, such as the popularization of multi-touch screens, or depth-sensors and body skeleton trackers in our case. Given the

captivating aspect of these new tracking technologies (e.g.: Microsoft Kinect), and the absence of efficient and well-established interaction paradigms, there is a lack of guidance about how to proceed while building gestural interfaces. By relating this concern with the previously discussed examples on Section 1.2.1 we questioned ourselves about *how to avoid such cases?* Therefore, we direct our efforts on the research problem of *building successful in-air gesture interfaces*.

While asking this question, we attribute the meaning of success to the quality of the interface usability among additional criteria for success regarding the interface concept. Here the concerns towards the interface usability represent goals such as high recognition rate and small response time, intuitiveness of the gestures and so on. The additional criteria may be used to cover additional ground towards the meaning of success for each interface. These criteria are diverse and may vary from one concept of interface to another. Examples of additional criteria are the social acceptance of the used gestures and the required physical effort to perform them.

It is important to notice that the measurement for success is specific for each interface. For instance, a particular interface may define success as the opposite concept of another. For example, on a gesture controlled application for a living room, one of the defined criteria for success can be to allow the user to fully control the TV functions without any physical effort. On the other hand, an exergame fails if the user can complete the game objectives without any physical effort. This way, the definition of the target interface must present, as pre-defined requisite, what are the success criteria to be considered.

In this thesis, we direct our efforts towards the application of the design process for building such interfaces. The design process allows the division of the target problem on phases to understand, conceive and develop solutions for the given problem. While applying a design process the development team can explore more options for solutions, identify failures faster, gather insight during the process and develop a final solution with increased chance for success. Therefore, by using such process, we aim to reveal the issues mentioned in the motivational examples in advance. We discuss the design process in detail on Chapter 2.

In order to apply the design process for building in-air gestural interfaces, it is required to provide tools and techniques for each of the design phases. According to our knowledge, no other work has proposed a set of techniques and tools to be used within a design process with the goal of producing in-air gestural interfaces. Therefore, we narrow our research problem in order to *assemble a set of techniques and tools for building in-air gestural interfaces through a design process*. The design process is well established, however, on the other hand in-air gestural interaction is a new topic and this way, there is room for investigation on how to apply the design process on this specific context. For the sake of simplicity, we will call our assembled set of techniques and tools as a toolset.

1.2.3 Scope

Considering that a solution development process can have its phases summarized as follows:

- *Understand* e.g.: observing the user, the environment, and the activity rules.
- *Create* e.g.: ideation, brainstorming, planning the solution.
- *Deliver* e.g.: prototyping, developing, testing.

Our toolset has a delimited scope regarding the design process; its coverage over these design phases includes the *Create* (which later we define as *Conception*), and *Deliver* (which later we define as *Prototype*) phases. This way, we use as input the decision (made on the *Understand* phase) that, whichever is the case, an in-air gesture is potentially desired to control a particular task. Once a team makes this decision and defines a target task as the input, our toolset becomes applicable.

On the other end, we do not focus on the ultimate solution (on the *Deliver* phase). Rather, we endeavor our efforts to decrease the time required to create low and high-fidelity prototypes for the gestural interface. Once made the decisions over the prototypes, it is possible that other works, from state of the art on gesture recognition, can offer more accurate solutions. That said, it is also possible that the resulting high-fidelity prototypes produced while using our toolset are adequate as final solutions.

The here exposed techniques and tools for prototyping are focused on the system input. On our toolset, while the input is defined to come from in-air gestures, the output can be diverse. The desired output can be a game control, navigation commands on a 3D virtual environment, movements of a robot or the illumination control of a room. Each interface may have a particular type of output once the desired gesture input is recognized. We focus on handling the input half, and on providing means to connect it to the needed output.

Regarding the covered body parts, this thesis scope is focused on, but not limited to, hand and arm based gestures. Hands are one of the most resourceful gesture communicators, and also used for an infinity of tasks connecting our intent to the environment around us, through tangible and intangible interactions. This way, although some of the following developments and results may be applicable on broadening scenarios, cases related to facial expressions, for example, fall out of the scope.

We take other scope decisions along the thesis development. We present each of these decisions in its particular context on the following Chapters.

1.2.4 Goals

The main goal of this thesis is to deliver a practical toolset for building in-air gestural interfaces. The specific goals are related to each of the discussed phases:

- Conception: explore and develop methods and tools to support the ideation of gestural interactions.
- Prototyping: explore and develop (low and high-fidelity) prototyping solutions to support the creation of gesture-driven applications.

1.3 Contributions

As contributions of this thesis, the first is the toolset itself. According to our knowledge, this is the first assembled set of techniques and tools to tackle the phases of *Conception* and *Prototyping* in-air gestural interfaces in a practical manner. It receives a target task as input and potentially delivers a high-fidelity prototype. The introduced toolset can be used to non-programmers and provide a high-fidelity prototype within an hour or even less. We provide the summary of the resulting toolset on the following link `http://cin.ufpe.br/~lsf/thesis/deliverables/summary.pdf`.

Regarding the conception of gestural interactions, we present two catalogs (tools) as contributions for future inspiration and analysis of gestural interfaces. The first catalog concerns about published research papers that mention the use of some gesture to control any task. It comprises the years between 2005 and 2014, and includes a full description of each gesture used on each paper and its corresponding target task. The resulting catalog of gestural interactions found in literature can be accessed by the following link `http://cin.ufpe.br/~gesturescatalog/classification`.

We also cataloged Science Fiction (Sci-Fi) movies and series, focusing on scenes in which characters use in-air gestures as controls. The Sci-Fi content presents fictional interfaces that can inspire and be a target of analysis while developing real interfaces. In total, we cataloged 229 scenes of titles produced from 1995 to 2014. We present the catalog of Sci-Fi gestural interactions on the following link `http://cin.ufpe.br/~gesturescatalog`. There is also additional information about this catalog on the research paper entitled "Sci-Fi Gestures Catalog" (FIGUEIREDO et al., 2015).

Both catalogs can be used as an inspirational source. The catalogs are also incremental, allowing the community to suggest and insert new content. At last, both count with a feature for producing comparison charts on real-time using the cataloged data, which can be used for further analysis. As an additional contribution, a result of creating and reviewing the catalogs was the creation of a revised taxonomy for in-air gestural interactions introduced Wobbrock and colleagues (2009). We based the revision and extension of that taxonomy on the literature and updated to support the analyzed content on each catalog. We also extended a taxonomy of tasks for interactive systems introduced by Piumsomboon and colleagues (2013).

On the *Prototype* phase, we focused on providing both low-fidelity and high-fidelity prototyping options. For low-fidelity, our main contribution is the validation of a prototyping

technique called Wizard of Oz (detailed by Martin and Hanington (2012)) for real-time sustained (continuous *Flow*) interactions using in-air gestures. The Wizard of Oz technique is usually applied to turn-based and delayed interactions (e.g.: voice commands). After training our wizards, we were able to achieve a response time compatible (within 100 milliseconds of difference) with the Microsoft Kinect V1 response time. We demonstrated the use of the technique on the scenario of navigation in 3D virtual environments. We present additional information about this experiment on the paper entitled "In-place natural and effortless navigation for large industrial scenarios" (FIGUEIREDO et al., 2014).

As contributions on high-fidelity prototyping, we developed two tools that were later combined. The first tool is capable of learning gestures from a one-shot training. The tool records the user state (or states on dynamic gestures) during the training and, from that point, it is capable of recognizing the same gestures on real-time (within a few milliseconds). We call each recorded state a checkpoint, and the tool Checkpoints (CHAVES et al., 2012; GAMA et al., 2012).

The second tool is called Prepose and is a rule-based gesture recognizer. Rules have some advantages over training, such as, the possibility of broader definitions of gestures. For example, a gesture that asks the user to clap her hands can be performed by several different ways, like clapping hands with the hands pointing up (side by side), or one on the top of the other, or both hands over the head, and so on. A training-based (machine learning) approach would require the recording of all those cases, while a simple rule checking if the user's hands were briefly touching each other would suffice in the rule-based case. Given that, Prepose first appeared as a complementary tool to the Checkpoints, allowing gestures to be written as *rules*.

We write the *rules* of Prepose through an extensible Domain-Specific Language (DSL), compatible with natural written language (English), this way being a Natural Programming Language, which allows the description of static and dynamic gestures. According to our knowledge, this is the first DSL for in-air gestures description created as a subset of natural language. This way, Prepose supports phrases like "put your hands above your head", or "touch your left elbow with your right hand". One advantage of describing gestures in natural language is that non-programmers can easily understand the content of the DSL, edit it and create their gestures. For example, a physiotherapist can describe an exercise on Prepose, or a game designer can prototype and tune some gestures for her gesture-controlled game.

We also created a C# library that recognizes gestures written in the same scripting language in real time allowing interaction. The same library can output common key and mouse events allowing non-programmers to prototype gestural interactions. Otherwise, developers can use the library on any C# application. This way, a second advantage of using natural language to recognize gestures is that the semantics of the library inference is the same as our understanding of the gesture itself. It is possible to give real-time feedback to the user about the following steps required by the gesture, or of what the user is missing on her attempt to perform the gesture.

Prepose is also the first rule-based recognition tool created on top of a Satisfiable

Modulo Theories (SMT) solver. SMT solvers are capable of reasoning and analysis over logical expressions. We used this capability to answer questions among three issues related to rule-based gesture descriptions and recognition: *validity, conflicts, and safety*. Once a gesture is written, it can be *invalid* due to impossible combinations of expressions, such as "put your hands above your head" and "put your hands below your neck". It is also possible to detect *conflicts* between two gestures by asking the solver if it is possible to *satisfy* both sets of expressions simultaneously. At last, it is possible to check if the gesture description asks the user to perform any *unsafe* motions or postures according to predefined *safety* restrictions. The solver used on Prepose is called Z3, available in different programming languages including C#, and it is pointed as a fast solver (DE MOURA; BJØRNER, 2008). We translate all written code on Prepose to gesture *rules* as Z3 expressions.

Additionally, we combined the Checkpoints solution with Prepose by providing an automatic Prepose writer based on the same one-shot training algorithm used in Checkpoints. This way, it writes in real-time the user body behavior using natural language. Both Checkpoints and Prepose consider the skeletal tracking result as input. The adopted sensor is the Microsoft Kinect V2, but both tools are extensible to any sensor (and its target body part) once provided a skeleton data. Prepose stable version can be found in the following link `https://github.com/Microsoft/prepose`, and its latest version in `https://github.com/lsfcin/prepose`.

Prepose concept also points contributions to the Security and Privacy field. By using Prepose, any third-party applications can ask for the recognition of specific gestures by sending the corresponding code (written in Prepose DSL). This way, these applications have no longer need to access the sensor data directly. For instance, the Microsoft Kinect V2 retrieves a color image (using a RGB camera and providing a clear vision of the entire environment), a depth image (containing information about the scene structure, e.g. living room furniture), a player detection based on the depth frame and the skeletal tracking data. The Kinect V2 is a sensor meant to be always-on and placed in the living room. A malicious application can take advantage of this acquired information to inspect user's privacy or even to find security flaws on the environment (using the depth and color images). This way, the Prepose concept proposes a trust boundary between the sensor and third-party applications (FIGUEIREDO et al., 2016, 2014, 2017).

1.4 Structure

This thesis is structured as follows. On the next Chapter 2 we present the proposed set of tools and techniques, which can be used to build a gestural interface capable of accomplishing the target task. On Chapter 3 we detail the *Conception* phase, showing selected or developed methods and tools to support the ideation of desirable gestural interaction. On Chapter 4 we explore different prototyping approaches for low and high fidelity purposes. Chapter 5 shows a

first case we used to demonstrate the application of the design process using the provided toolset. Lastly, on Chapter 6 we draw the main conclusions and present the future work.

2

Toolset

This Chapter introduces proposed toolset for building in-air gestural interfaces. The Chapter includes its background context, goals, definition, flow and an introduction of each technique and tool used for conception and prototyping of such interfaces. Later, we detail the proposed tools and techniques the following Chapters.

2.1 Related works

As related work, we found few cases which researchers pursued the goal of providing solutions comprising both phases of Conception and Prototyping regarding the construction of interactive solutions. While reviewing the literature the most common case was to find works presenting authors efforts on conceiving and testing a particular set of gestures for a specific application domain or activity. On Chapter 3 we detail a range of ten years (2005 to 2014) of research publications found on four different bases regarding the use of in-air gestures. On that Chapter our focus is to present base material for designers to perform an easier Secondary Research.

Nevertheless, in addition to that, while reviewing those works we also searched for related works which aimed to deliver a toolset likewise we intended to at the time. No works were found on that study treating the problem of building in-air gestural interfaces on a macro level considering both the Conception and Prototyping challenges. On a second and later search for related work we found two publications which we relate to on the following paragraphs.

In the first related work, Calegario and colleagues (2017) proposed a method and a toolkit for the creation of Digital Musical Instruments (DMIs). In this work the authors propose a morphological chart to support the creation of concepts for DMIs and a customizable artifact which allows the prototyping of such concepts. The provided concepts are linked to buttons and tangible gestures, which are a common ground for players of music instruments, instead of in-air gestures.

In the second related work, Othman and colleagues (2016) reinforce the concerns regarding the Conception phase stating that “problems arise when choosing the best set of gestures for

a given application." Therefore, the authors analyze previous work on creating in-air gestures and synthesize the used methods for that goal. The authors also point the Wizard of Oz as a resourceful technique for collecting and prototyping concepts of gestural interactions. Moreover, the authors also point techniques such as questionnaires, semi-structured interviews and the think-aloud protocol for data collection regarding the user experience on the tests.

2.2 Background

We built the proposed toolset considering the design phases of conception and prototyping searching for tools and techniques to deliver options for each step within the design process given our scope delimitation (explained on the scope section of Introduction).

At first, we performed a review of design processes in order to set up the base of phases and steps for which we would provide tools and techniques. Design processes are well known and used worldwide. On "How do you design?" Hugh Dubberly (2005) exposes a compendium of processes as common ground for various application domains, including architecture, mechanical engineering, and software development. The compilation presents a non-linear additive definition of design and development processes.

It starts from the simplest archetype of $input \rightarrow process \rightarrow output$, to the two-step definition of $input \xrightarrow[analysis \rightarrow synthesis]{process} output$, later expanding these two steps to three, five and seven as shown in Figure 2.1.

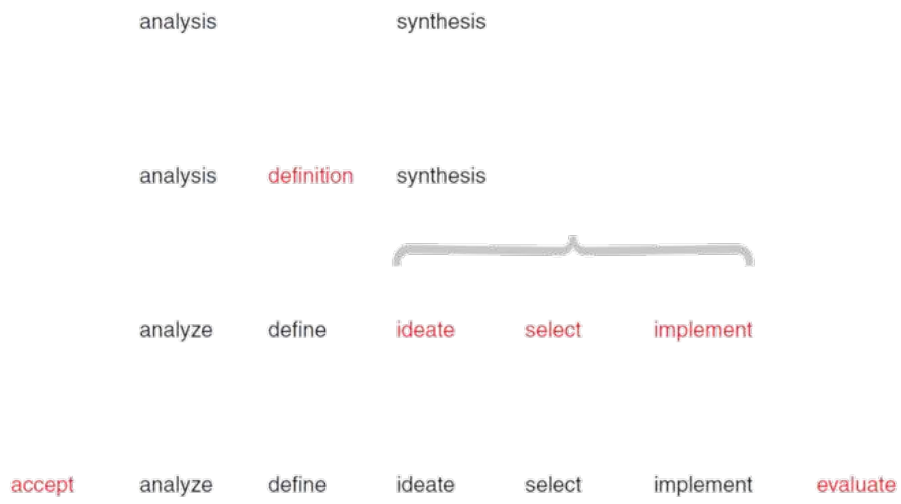

Figure 2.1: Expasion of the two-step *analysis, and synthesis* (DUBBERLY, 2005).

These seven steps are core for a series of succeeding proposals for design processes. While discussing Design Thinking on their course, Matthew Ellingsen, Emilie Fetscher and Emma Saunders (2012) presented a variation with three phases (Understand, Create and Deliver) comprising five steps (Empathy, Define, Ideate, Prototype, Test) as shown in Figure 2.2.

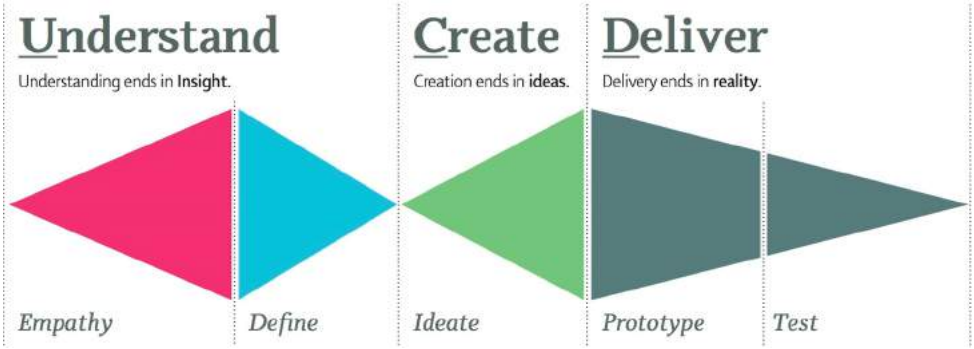

Figure 2.2: Design process in three phases (MATTHEW ELLINGSEN EMILIE FETSCHER, 2012).

Figure 2.2 also explores the concepts of convergence and divergence within each phase and step. The process flow usually implies in periodic movements of divergence and convergence. On the search for understanding, designers first diverge with an open mind, looking for issues and rooms for improvement in the target context. They later converge to a definition of which are the issues to tackle, or which are the challenges to pursue. Then divergence takes place once again while creating (ideate) and expanding the possibilities, later focusing on reducing these options to the final solution through prototyping and testing. This way, these concepts guide the definition of expected results for each step. We used the definition of design process shown in Figure 2.2 and tackled the Create and Deliver phases for our toolset. Given the scope delimitation explained on the Scope Section on the Introduction Chapter, we consider that the Understand phase was previously performed by the team of designers and as output they defined the target tasks that should be controlled through in-air gestures.

Dubberly also points the *fractal quality* of processes, allowing the partitioning in minor (and more particular) processes as shown in 2.3. Each process fits the black box archetype ($input \rightarrow process \rightarrow output$) which allows their concatenation, using the output from a process as input for the following one. This fractal quality can be explored indefinitely, breaking processes as long as it shows to be positive to achieve the expected results.

These minor processes can be defined as *techniques*, aiming to meet specific goals in each phase or step of a broader process. In their book *Universal methods of design*, Bella Martin and Bruce Hanington (2012) expose and detail over one hundred techniques; examples include well-known techniques such as literature reviews, questionnaires, and experiments. Martin and Hanington present each technique coupled with suggestions about which phases of the entire process the it should be used. They divide the applicability of the presented techniques within five phases: 1 Planning, Scoping, and Definition; 2 Exploration, Synthesis, and Design Implications; 3 Concept Generation and Early Prototype Iteration; 4 Evaluation, Refinement, and Production; 5 Launch and Monitor.

While reviewing the compendium provided by Martin and Hanington (2012) we cataloged all techniques and assigned them to the design phases of Conception and Prototyping considered

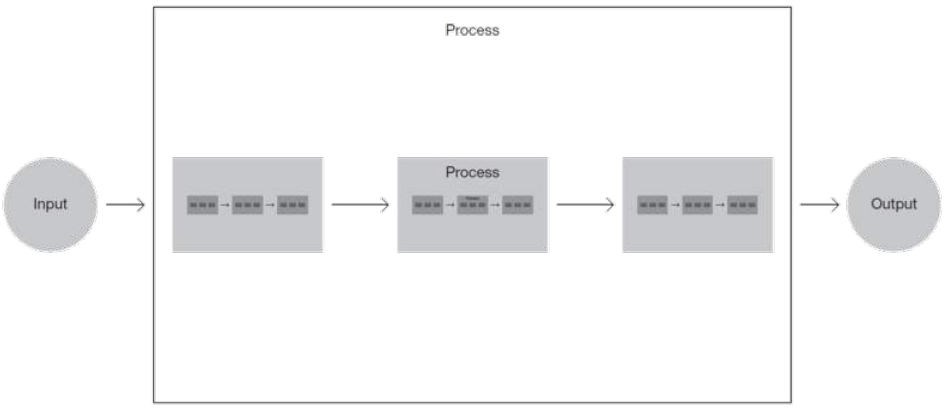

Figure 2.3: Hierarchical vision of processes, showing the possibility of recursive composition of the main process by sub-processes (DUBBERLY, 2005).

on our toolset. On the following sections we present the result of this review, showing which techniques we considered aligned with the goal of producing in-air gestural interfaces on each step considered on our toolset. In addition, we present some tools to aid specific techniques, for example for the *secondary research* technique we provide two catalogs, and for the *high-fidelity prototyping* technique we provided a tool for recording and recognizing gestures. These tools were created to save time of designers while using these techniques, given that the techniques are still presented on a high-level of abstraction and require a considerable effort on analyzing options and developing solutions to be used within the design process.

2.3 Guidelines

Our primary goal is to, given a need for an in-air gestural interaction, produce high-fidelity prototypes which can be later experimented or even launched. The toolset must receive as input a *target task* to be accomplished using in-air gestures as a way of interaction. Examples of target tasks are zooming, go to next page, start or open an application, exit, turn on or off a device, and so on. Once the target task is defined, we guide the flow of our toolset by the following ordered questions:

1. Is there an already used gesture for the target task?
2. Is this particular gesture suitable for the target task?

The first question aims to provoke understanding about the previous use of gestural interaction for the same target task. This way, we can consider previous experiences with similar problems. The found gestures can be useful in several ways. As inspiration, providing a start point for the discussion around the interaction. As counter-examples, exposing unwanted behaviors or issues as previously indicated by the two examples of *Fighters Uncaged* and the *CAVE Navigation* on Section 1.2.1. Pre-used gestures can also be replicable, easily reusable in similar or different contexts, or else adjusted to fit new requirements.

We add this question to encourage designers to consider previously explored interactions, and favor the repetition of interaction patterns. This way, once a particular gesture is well-accepted in scale by users to achieve a particular task, by consulting previous experiences of gestural interaction designers will likely be able to find such pattern and choose to reuse it or not. Therefore, by pursuing this guideline we aim to reinforce the search for interaction patterns.

Once a particular gesture can be part of a possible solution, we ask ourselves the second question (*Is this particular gesture suitable for the target task?*). It is important to notice that regardless of the understanding obtained while pursuing the first question, it is not a requirement to use a gesture from a previous solution. On this second question, we target a particular gesture that can come from any source, either found and reproduced or created from scratch. This question guides us mainly to identify if that gesture is suitable as a solution from the user viewpoint, regardless the technical challenges concerning its recognition. From a viewpoint of the interaction design, this question is directly related to the need of prototyping and testing the solution in order to find out as soon as possible if the proposed solution is suitable or not for the given challenge.

Technological challenges for gesture recognition may be cumbersome in this phase. There are usually different options of technologies and algorithms, each with particular limitations that can detour the primordial search for a successful interaction. Therefore, we consider the priority to explore and understand if a particular gesture is suitable for the desired interface, without compromising with the technical challenges to come. The goal is first to figure out what makes sense from the user perspective, and then focus on adapting the available technical resources to meet the expectations.

These are broad questions, comprising the open concepts of suitability and use, which may be impossible to resolve. Thus, our goal is not to fully answer them. Instead, we use these questions as guidelines to build our toolset. Combined with these guidelines we use the background concepts of sub-process (fractal quality), division on phases and steps, and the flow going from divergence to convergence to define our toolset.

2.4 Definition

Figure 2.4 shows the overview of our toolset including its input, output, and steps. Our *toolset* can be used a minor process (for building a specific in-air gestural interface) within a larger ongoing process (e.g. for creating a complete solution for a particular application domain). The figure shows a broader process as a line which somehow, after completing its path, transforms a curiosity (which also could be a question, belief or speculation, illustrated by the ? symbol) into value (shown by the $ symbol) (DUBBERLY, 2005). This description of a process is as comprehensive as it can be. Here we use this concept to show that to use our toolset there are no requirements tied to the larger process, it can be a random path that at some point demands the use of in-air gestures to control a given task, and at some other stage receives our output.

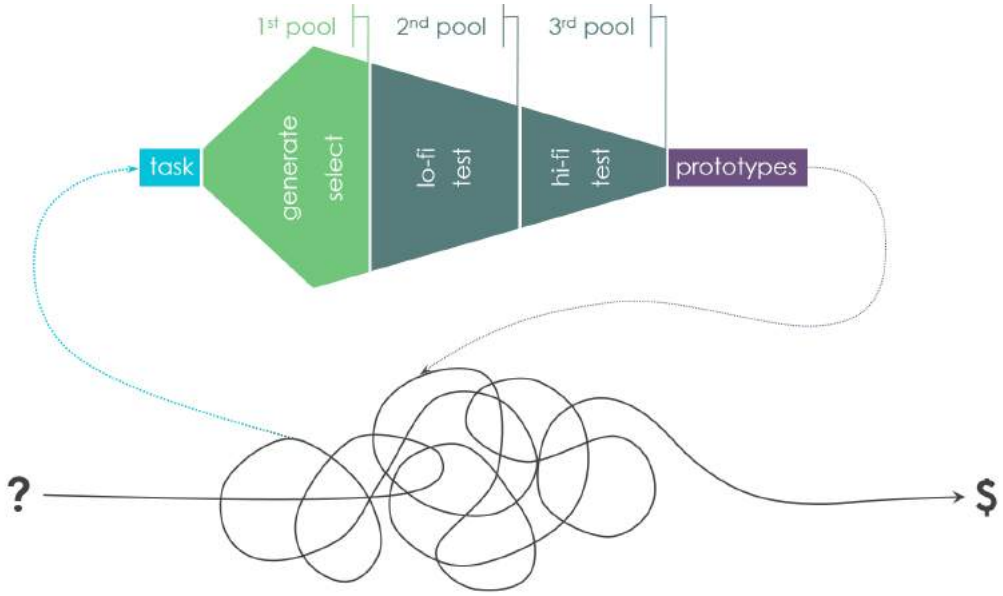

Figure 2.4: Overview of the proposed toolset used on a longer ongoing process.

Therefore, our toolset is applicable at any point of the larger process, once revealed the need for a gestural interface. The toolset requires as input a target task to accomplish through the gestural interaction. As output, the toolset produces validated prototypes that feed the ongoing process with insights (as learned lessons) and on best cases with in-air gestural interfaces as solutions for the given problem.

2.4.1 Phases color scheme

On our toolset (Figure 2.4) we use a similar color scheme as the one used on Figure 2.2, using as reference its process phases to compose our toolset. The task has a light blue color representing a result from an understanding phase. At some point of a larger ongoing process, a decision was made to accomplish a target task through in-air gestural interaction. That definition is a result of the understanding of a particular goal on the current project, even if it is not officially reported this way.

We use the light green color for the *Conception* phase. This phase comprises the generation of alternatives. Here is the time to diverge and create concepts for possible solutions. While on Figure 2.2 the creation process is divergent only, converging only after immediately jumping to the *Prototype* phase, here we include the first round of convergence for the selection of the created concepts.

By its turn, we use the dark green color for the *Prototype* phase, comprising the steps of prototyping and testing. Instead of allocating the first half of this phase for prototyping and the second half for testing (as shown in Figure 2.2), we break it in two cycles, the first for low-fidelity prototyping, and the second for high-fidelity. At last, we added a new component for the output (prototypes) of the toolset in dark purple.

2.4.2 Steps

At first the flow diverges (expands) on a *Conception* phase, aiming to generate gesture alternatives for the given task (*Generate* step). After that point, it starts to converge, first selecting which gestures are worth the effort of being prototyped (*Select* step). The selected gestures represent the *1st Pool*. This pool represents concepts of gestural interactions (gestures plus target task) either replicated from other sources or created using the provided tools.

Then the *Prototype* phase takes place. The following step is to produce low-fidelity prototypes (*Lo-fi* step), and test (first *Test* step) each selected gesture to discard those which present significant issues, and keep the ones which are preferred and accepted by the users. These remaining gestures compose the *2nd Pool*. The *2nd Pool* represents gestures validated by users, as indications of accepted interactions. Technological limitations are yet not considered; here we seek for low-fidelity prototypes that can simulate the interaction without compromising with any device or algorithm specifications and constraints.

These remaining gestures are used as input to create high-fidelity prototypes (*Hi-fi* step), capable of automatically recognize user movements in real-time and provide proper output (e.g. visual, and audio) as feedback. High-fidelity prototypes require working solutions and therefore are essential to determine which gestures implementation are practical with the available resources, and which are out of range and can not be implemented considering the available time and resources. We test these prototypes as well (second *Test* step), and the prototypes which are successful represent the output after using the toolset. Additionally, we store the gestures used on these prototypes on the *3rd Pool*. The *3rd Pool* represents gestures, previously validated with users, and successfully implemented as high-fidelity prototypes.

In addition to the prototypes, the three gesture pools are part of the result from the toolset. These pools work as a log of its execution, showing at which step each gesture is discarded and thus, indicating why it happened. A gesture discarded on the *2nd Pool* represents an assumption that it would work for users, which was not well accepted in the end. A gesture discarded on the *3rd Pool* contains gestures previously validated by users but, at the same time, not feasible to be implemented with the available time and resources.

The pools are also reusable. For example, while using the toolset on a new iteration, using the same target as input, the *1st Pool* can grow by adding new (or modified) gestures. The *2nd Pool* can also be reused, for example, in cases where an unavailable resource becomes available, such as the acquisition of a new depth sensor and tracking system. In this case, the gestures on the *2nd Pool* which could not be implemented on high-fidelity prototypes earlier, now gain a new chance to become part of the final solution.

Each one of the steps of the design process shown in Figure 2.4 can be fulfilled through different paths, using different tools and techniques. We propose the use of a specific set of techniques and tools to fit on each of the steps: Generate; Select; Lo-fi; Test; Hi-fi; Test. The following mentioned techniques and tools are suggestions, not requirements, which means the

toolset can be tweaked and adapted towards each team goal if a particular tool or technique presents advantages in comparison to the ones we suggest or provide in this thesis. Therefore, designers[1] have autonomy to choose between the suggestions, add new options or discard some tools and techniques that for some reason do not fit into their plan. Figure 2.5 illustrates in advance the goals aligned with techniques and tools we propose for each step of the toolset. On the following subsections, we discuss in detail these assigned techniques and tools.

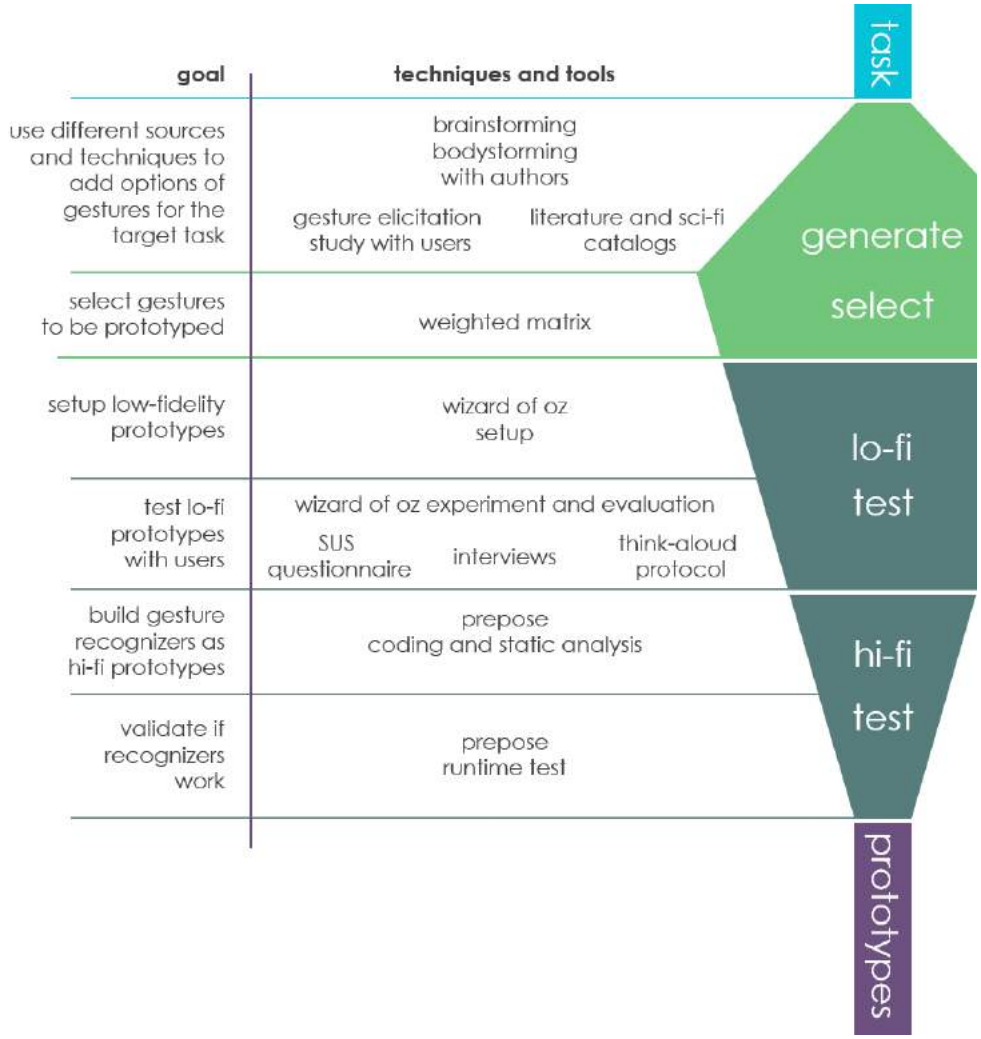

Figure 2.5: Summary of the proposed toolset, showing the steps of the design process with the assembled set of techniques and tools.

2.4.2.1 Generate

For the *Generate* step we propose the use of one or more sources. One possible option is to have the designers themselves producing their concepts from scratch. Here we refer to the term *designer* as any stakeholder directly enrolled on the process of using the toolset. This way,

[1]Here we refer to designers as those who are conducting the application of the toolset. This way, designers can be HCI researchers, interaction designers, game developers or any stakeholder that has the goal to define interactions and produce gestural interfaces while using our toolset.

designers can represent developers, interaction designers, UX researchers and so on. Another source can be the users themselves. Through Participatory Design techniques, users can lead the creation of alternatives. Alternatively, users can be the trigger to the generation of new options on a passive way, while they are observed and studied by designers.

In addition to these sources, there are also documented sources which can be useful in this exploratory phase on a *secondary research* (HANINGTON; MARTIN, 2012). For example, the literature on HCI already contains several cases of the use of in-air gestural interactions. There are examples of the use of in-air gestures on current existing applications. With the rise of sensors like Leap Motion, Microsoft Kinect, Myo Armband, among others, there are products and case videos showing the use of these technologies for specific purposes on different application domains, such as maintenance, education, and medical training. Even Sci-Fi content can be a source of inspiration. Sci-Fi titles are embedded with additional freedom to envision future interfaces, and so they have the power to depict gestural interactions that are yet to come.

Designers as source. There are some creative techniques to aid designers in the task of generating alternatives. We propose the use of brainstorming as a straightforward and well-known technique. Supported by Bella Martin and Bruce Hanington thoughts on their compendium of techniques, brainstorming rules[2] "create a safe forum for the expression and free association of creative ideas, and quell any inhibitions of the participants by providing a judgment-free zone to explore new concepts" (HANINGTON; MARTIN, 2012).

As an alternative to brainstorming, we also propose the use of another technique called bodystorming (HANINGTON; MARTIN, 2012). Bodystorming uses role-play to aid the creation process. It encourages participants not only to suggest but also to perform loosely configured simulations of that idea, acting as if such concept was real and discussing it simultaneously. Bodystorming is particularly interesting for in-air gestural interactions; it is usually simple to demonstrate how to perform the suggested gesture. This demonstration empowers the communication of the team, adding visual concepts and a notion of what would the user experience be while using the proposed concept.

Users as source. Users can help not only to validate but also to create concepts of possible solutions. The creation can be achieved either by involving users on an active or passive stance. Designers can use observation techniques to understand how users use their body language while engaged in the target context. In the case of a task already performed without gestures, it is possible to map user movements to the target task itself. For example, considering the activity of watching a TV, the target task of *pause* the movie can be triggered in different ways. First by pressing the pause button on the remote controller; or else by verbally asking the person with the controller to pause; or even, while answering a call, perform a gesture for the same person asking her to hit pause. Observing and analyzing such scenarios can produce meaningful

[2]Examples of brainstorming rules are *aim for quantity over quality* and *do not censure a colleague crazy thought*

insight regarding which gestures (or which gestural characteristics) should be used to accomplish the *pause* task. Some examples of techniques are Observation, Participant Observation, and Fly-on-the-wall Observation (HANINGTON; MARTIN, 2012).

Alternatively, the user can actively participate in the generation of concepts of gestural interactions. For this case, we propose the use of gesture elicitation studies. Gesture elicitation studies is a technique to generate user-defined gestures. For example, Jacob Wobbrock, Meredith Morris, and Andrew Wilson (2009) conducted a study with users to define touch-based gestures for specific tasks . They first showed users what would be the expected result of the task, and then, they asked them to perform a matching gesture. They recorded the entire experiment and further analyzed it, pairing gestures and the computing the agreement scores between users' suggestions. Such score giva a metric for further selection of which gestures were likely to represent a consensus as good input for the target task. There are also examples of the technique being used for in-air gestures, for example for TV control (VATAVU, 2012; DONG et al., 2015) and Augmented Reality (AR) common tasks (PIUMSOMBOON et al., 2013).

Literature as source. Elicitation studies are commonly published as a research material on HCI literature. If the target task used as input is present on a published gesture elicitation study, the plan to perform a new elicitation study may be skipped, saving time on the *Generate* step. In addition to these studies, HCI published research also contains several cases of in-air gestures used in diverse application domains. Therefore, we consider the literature as an additional source containing concepts that can be either directly replicated or used for inspiration on the *Generate* step.

With that in mind, we conducted a literature review of HCI scientific publications containing in-air gestural interactions covering in total ten years (from 2005 to 2014) on four different databases (IEEE, ACM, Science Direct, and Springer). We also organized our review on a web catalog (available at `http://cin.ufpe.br/~gesturescatalog/classification`), so it can be easily accessed. The catalog exposes each gesture (usually more than one per paper) and its corresponding task. This way, it can be used as a tool for quickly finding similar applications scenarios and therefore add input to the *Generate* step. Both the literature review and the resulting web catalog tool are further detailed on Chapter 3.

Sci-Fi as source. In addition to literature, we also reviewed several Sci-Fi titles, released from 1995 to 2015. We trimmed scenes containing gestural interactions from those titles and launched each scene on a web catalog similarly as the one containing research papers. The Sci-Fi web catalog is available at `http://cin.ufpe.br/~gesturescatalog`. The motivation behind selecting and grouping those scenes lies behind a set of factors. Sci-Fi titles are likely to portrait different technological scenarios, usually on a path towards predictions of plausible future interfaces for humankind. At the same time, these titles represent efforts on building such interfaces due to concerns of how it would appeal to the public. These efforts are directed to

Table 2.1: Weighted matrix archetype. The 4th gesture is selected given its higher score.

	1st Criteria	2nd Criteria	Total
Gestures \Weights	2	3	
1st Gesture	1	2	8
2nd Gesture	2	2	10
3rd Gesture	3	2	12
4th Gesture	2	3	**13**

bring polished concepts and visuals on prototypes. Such examples are useful for inspiration (as good examples or counter examples) or reproduction. This discussion, including the Sci-Fi review and catalog overview, is presented with more details on Chapter 3.

2.4.2.2 Select

Once the *Generate* step ends, the *Select* step takes place, which aims to reduce the number of gestures to prototype. If the number of generated gestures is already satisfactory and needs no reduction, this step can be skipped. However, there are cases in which the generation produces an excessive number of concepts, and some are clearly unworthy the effort of being built as prototypes. In these cases, we propose the use of a Weighted Matrix (shown by Marting and Hanington (2012)) to rank and filter the concepts which should lead to the next step.

To build the Weighted Matrix, designers should place each idea in a row, and success criteria in a column, as shown in Table 2.1. The chosen criteria can be diverse, depending on the goal of the solution and the concerns towards the desired user experience. For example, the physical effort can be positive if the aim is to make the user exercise herself while using the interface, or else, it can be negative because fatigue is undesired. Additional examples of criteria are: intuitiveness; memorability; ambiguity (if the gesture is similar to another postural behavior on the same activity); and social comfort (while using the gesture)[3].

Once all criteria are defined, it is time to attribute weights for each one (e.g. weights varying on a scale from 1 to 3). We suggest that all designers assign scores (e.g. from 1 to 3) for each concept and then the average score of each concept can be used for ranking. With this, we mitigate the impact of a possible bias introduced by a designer blindly upvoting his favorite gesture. The team can discuss significant discrepancies on assigned scores before discarding any idea, because these differences may indicate that part of the team misunderstood some aspect of that concept. After ranking all designs, the team must choose a cutting point, determining a threshold for which scores to accept and which to reject. Then, the team stores the accepted gestures for the given target task on the *1st Pool*.

[3]We advise not to add (yet) criteria linked to the implementation of the gesture, like the feasibility or cost to implement it. We are postponing these concerns to the *Hi-fi* step, giving a chance to first understand what gestures are better solutions for users, to later invest time and resources to make it happen.

2.4.2.3 Lo-fi

The *1st Pool* of gestures is the input for the *Lo-fi* step, which handles the setup of the low-fidelity prototypes. On this step, we propose the use of the Wizard of Oz technique (HANINGTON; MARTIN, 2012). The essence of this technique is to replace parts of a computer system (which are not available yet) by humans (Wizards). On our case, the Wizard is called to change the gesture recognition part of the system. This way, once a user performs a particular gesture, the Wizard will visually identify it and press a key to determine that input. This technique power is to rapidly test different scenarios which otherwise would require implementation of complex systems.

To set up the prototypes, designers must first provide a proper output system. For example, on the activity of watching TV, designers must have access to a TV. The second part is to provide a place for the Wizards to observe the user, allowing them to identify the gestures performed by the user. At the same time, the Wizards must have access to an input device capable of performing the same target task. For example, on the watching TV activity and for the pause task, a possible input device would be the remote controller.

The setup may or may not allow users to see the Wizards. This option presents advantages and disadvantages on both sides, and the choice is up to the designers. If users do not see the Wizards, then the designers can play the card that the system is working and that they want to test it with users. This setup adds realism to the simulation and facilitates the role-play of the users. On the counterpart, hiding Wizards from the users usually requires additional effort for the setup, and in some cases is hardly possible. Moreover, once Wizards are visible, users can directly communicate with them, the Think Aloud Protocol technique naturally takes place, and usually users report difficulties and make suggestions transforming the experiment session into a Participatory Design environment (HANINGTON; MARTIN, 2012). We discuss in detail and present a validation experiment of the use of the Wizard of Oz technique on low-fidelity prototypes of gestural interactions on Chapter 4.

2.4.2.4 Test

Once the setup for prototyping is ready, the experiment can take place. Users must be called to test the selected gestures from the *1st Pool*. Designers should take into consideration how much time each user will spend on a session, and how many gestures they will test. Users can feel fatigue or boredom if the task is too long or repetitive, which will compromise their performance and the overall evaluation. Designers must set these parameters and the evaluation techniques to be used before the test starts.

As a first technique, we suggest the use of questionnaires in between each gesture execution. Questionnaires are a common way to assess users' perception of usability of a particular system or feature. We propose the use of the System Usability Scale (SUS) questionnaire introduced by John Brooke (1996). The SUS questionnaire is composed of 10 questions answered on

Table 2.2: Adjectives of user perceived usability associated to SUS questionnaire scores (BANGOR; KORTUM; MILLER, 2009).

Adjective	Mean SUS Score
Worst Imaginable	12.5
Awful	20.3
Poor	35.7
OK	50.9
Good	71.4
Excellent	85.5
Best Imaginable	90.9

a five-point Likert scale (from 1 to 5), 1 meaning “strongly disagree” and 5 “strongly agree”. Given the number of closed questions, it is also a fast option for validation of the proposed solution. It also balances positive and negative items (detailed below) reducing the positivity bias of participants which are inclined to agree with the written affirmatives.

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The answers can then be used to build a score. The negative items are inverted and then computed, as explained by Brooke: “To calculate the SUS score, first sum the score contributions from each item. Each item’s score contribution will range from 0 to 4. For items 1,3,5,7, and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU. SUS scores have a range of 0 to 100”. Aaron Bangor and colleagues conducted additional research to assign adjectives for the SUS scores (BANGOR; KORTUM; MILLER, 2009). Table 2.2 shows the mapping between adjectives and scores. Therefore, the scores can not only be used to rank the tested gestures but also to understand the acceptance. This way, designers can, for example, accept only gestures classified as *Good*.

The SUS questionnaire aims to validate the system usability but does not perform a diagnosis. If desired by the designers, supplementary techniques can be used in a complementary approach to better understand the causes and effects, the positive and negative points of each gesture and about the prototype setup. A simple start would be to add an open space after the

SUS questionnaire letting users freely comment about their experience. An interview can be conducted at the end of the experiment, giving users time to think about their experience and share their vision. Designers also can record the experiment session for further analysis. At last, the Think-aloud Protocol can be used to gather users thoughts at the moment they are using the prototype (HANINGTON; MARTIN, 2012); this can trigger insights and aid the task of diagnosing issues and strengths of the prototype setup and the tested gestures.

Diagnostics can be used to remodel the prototype or search for a different set of criteria (and therefore different gestures) on a second iteration of the process. Nevertheless, to advance to the next step a set of gestures must be selected. This selection can be made by directly using SUS scores as previously mentioned, or additional factors from observation and analysis of the experiment can be used to up-vote or down-vote the gesture, pushing it to the next step of high-fidelity prototyping. Designers then store these remaining gestures on the *2nd Pool*.

2.4.2.5 Hi-fi

The gestures on the *2nd Pool* are the input for the *Hi-fi* step, which handles the creation high-fidelity prototypes. This step aims to create prototypes capable of recognizing the selected gestures in an automated manner. This way, Wizards are no longer applicable. As the first procedure, we recommend designers to check for available devices (e.g. Microsoft Kinect, Leap Motion) and available software (e.g. device SDK, or open-source tracking algorithms) which are possible solutions for sensing and tracking the user, retrieving user joints position in real-time. If one of these solutions (sensor + algorithm) presents the capability of tracking the user while she is performing the gesture, then there is an indicative of an available promising solution to recognize such gesture.

In case no available tracking solution shows promising results, then our toolset finds a hindrance to advance on further steps. Thus, designers should seek for additional solutions to recognize gestures without acquiring skeletal (and joint) tracking data. There is a considerable amount of research on using color frames and depth maps to recognize in-air gestures. For instance, Jesus Suarez and Robin Murphy (2012) reviewed in total 37 research papers aiming in-air gesture recognition using depth information from sensors. Some of these papers skip the tracking phase by using the depth data as input for the recognition. For further information, we discuss this choice of requiring skeletal data in detail in Chapter 4.

In case there is an available tracking solution then we propose that the designers use our tool called Prepose[4]. We introduce Prepose as a language for intuitive gesture description, which allows non-programmers to define gestures rapidly and easily by using the knowledge about how the body should be positioned and moved. Prepose is a DSL which allows designers to

[4]It is important to notice that, although Prepose language is broad, in its current state it only supports the Microsoft Kinect as the input sensor. Prepose language and runtime are easily extensible, and in case the Kinect device is not suitable for the task, but other sensors are (e.g. Leap Motion), we encourage the team to consider the development of Prepose extension to support such devices. Prepose will be described in detail in Chapter 4.

write gesture descriptions using natural language. This way, the proposed DSL is classified as a Natural Programming Language. A simple example of a Prepose code can be the following: put your right hand above your head. That snippet can be used for example to describe an attention gesture, being performed when the user wants to call the system focus to herself instead of other people in the room.

Written gestures also allow easy editing. Once noticed a mistake on a gesture description, designers can rewrite and fix the identified issues. This characteristic differs from the usual procedure present on gesture recognizers based on training. Training approaches usually require the recording and tagging of different users performing the gestures in different scenarios. In order to edit a trained gesture, this procedure must be redone.

On the other hand, training approaches present advantages, usually saving time while describing complex gestures. With that in mind, we added a training module to Prepose, being capable of writing a gesture by receiving the skeleton joints data related to a single execution of the gesture. This one-shot training algorithm becomes as a complementary approach to the direct writing option, focusing on speeding up the description of complex gestures with minimal training time. Since the output of the training is also a description, it also can be easily edited.

We also provide a static analysis module, capable of resolving expressions within gestures descriptions. This analysis can provide valuable answers for questions such as: *is there a possible conflict between these two gestures that I am combining?* Conflicts usually represent a possibility for the occurrences of false-positives, allowing undesired behaviors because they mean that two gestures can be triggered at once, with a single user movement. Without static analysis, these *conflicts* can only be detected during the online testing phase, while designers or users perform the gesture. Once added the static analysis feature, designers can check immediately after writing a gesture if the current gesture can raise potential *conflicts* with another gesture written previously. Prepose language, the one-shot training algorithm, and the static analysis module are described in detail on Chapter 4.

2.4.2.6 Test

In addition to Prepose language, we introduce Prepose runtime module. Prepose runtime is capable of interpreting written gestures and comparing these input descriptions with the skeleton data received in real-time. This way, the gesture description becomes a gesture recognizer while using Prepose runtime. Therefore, designers can instantly test the written gestures on the previous step, and once the tests present any recognition issues, the designers can iterate back to the previous step to edit and enhance the gesture description.

This fast cycle of testing and tuning is necessary because gesture descriptions are plural. In other words, designers can describe the same gesture in different manners, some more accurate (better precision of the classifiers), some more open (better coverage of the classifier). Once users validated each gesture previously on the *Lo-fi* step, designers can initially test Prepose prototypes by themselves. If the prototypes show promising recognition performance, designers

should conduct user tests once again.

Currently, Prepose runtime only supports the input skeleton data from the Microsoft Kinect SDK 2.0. On the other hand, once both language and runtime are based on describing and relating body joints, their design can be expanded to incorporate new sensors (e.g. Leap Motion) and skeleton data (e.g. hands skeleton or face landmarks). We describe Prepose runtime in detail on Chapter 4.

2.5 Conclusion

On this Chapter, we introduced our toolset to build in-air gestural interfaces. The main contributions so far are represented by how we link each technique and tool from one step to another. According to our knowledge, this is the first toolset that handles the *Conception* and *Prototype* phases for building in-air gestural interfaces. On the following Chapters, we introduce our proposed tools in details, including the implementation choices and results.

3

Conception

The conception of a gesture can derive from different means. The research literature on HCI and Gesture Recognition commonly shows authors as protagonists on the creation of the gestures used for interacting with their systems. Gestures can also be mimicked from success cases on previous works that explored similar tasks. Also, the inspiration for conception can come from available applications on the market or even from fictional content on visual media such as movies, series, and so on. In particular, the Science Fiction (Sci-Fi) industry has already proven its value on anticipating new interfaces and interaction paradigms. This way, evidencing the relevance of studying its effect and areas such as speculative and futuristic design (SHEDROFF; NOESSEL, 2012). At last, potential users actively help to create gesture on an elicitation process. The elicitation concerns of extracting the key concepts of the gestural interactions from the users, capturing their vision of how the interaction should work in practice.

This Chapter details the *Conception* phase of the design process detailing the tools we provide for this phase on our toolset. The goal of this phase is to produce concepts as solutions for in-air gestural interactions for the target task. The *Conception* phase divides itself into two steps: *Generate* and *Select*. In the toolset description (Chapter 2) we propose the use of different sources for the *Generate* step and detail the options of generating gestures through designers and users. On this Chapter, we focus on detailing the generation of gestures from secondary research on HCI literature and Sci-Fi titles.

As explained by Bella Martin and Bruce Hanington (2012), “Secondary research consists of information collected and synthesized from existing data, rather than original material sourced through primary research with participants.” Regarding the output, Martin and Hanington also stated that “Secondary research is traditionally summarized in systematic reviews, or literature reviews, with full citations of sources. While these reviews are most commonly communicated in written reports, in design, secondary research can also be collected into visual summaries for shared viewing, sorting, synthesis, and the crafting of narratives. Recently, blogs have become common repositories for collecting secondary research, facilitating the organization of the text, visual references, and source links, in a format convenient for sharing”.

3.1 Goals

Therefore, as the first goal, we aim to facilitate the access and analysis for the secondary research performed by designers using our toolset. Designers should be enabled to instantly search and find if and how researchers have previously tried, through in-air gestures, to control the same target task they are targeting. This way, as specific goals, we seek to collect, review, classify and publish research materials that experimented the use of in-air gestures to accomplish a task.

The publication channel must be online and collaborative. Although journals and conferences can give online access, we consider these as publication channels with restricted access. Also, the published content offers a limited interaction, usually presented as a .pdf file which allows readers only to go back and forth on the pages. We envision an interactive tool as a catalog, through which designers can select their desired criteria for each found interaction (e.g. target task) and filter occurrences. We also understand this as a collaborative work; this way new research material can be added by others, crowd-sourcing the responsibility of maintaining and expanding the database.

While the literature on HCI brings a vision focused on research, usually aiming to ground or consolidate new concepts, Science Fiction (Sci-Fi) travels further on imagining disruptive interactions between humans and machines. We also propose the use of these concepts as a valuable entry for the creation of gestural interactions. This way, we give designers inspiration and insight on both scenarios, first on ideas grounded on HCI research, and second on fictional concepts detached to technological limitations. This choice follows brainstorming principles such as *"go for quantity over quality"* and *"encourage out of the box ideas."* Similarly as our goal to exposing literature content, we also seek to expose Sci-Fi content on a collaborative catalog (further detailed on Section 3.4).

3.2 Literature gestures

In recent years, gestures have been widely used while aiming to achieve naturalness in interactions between humans and machines. With the advancement of vision-based tracking technologies, the use of in-air gestures is encouraged on new scenarios that were not suitable before. While using such devices, the setup time decreases, in comparison to the time required for wearable solutions such as data gloves. At the same time, it improves the freedom and comfort sensations. As discussed on the context (Section 1.1) of the introduction (Chapter 1), the current scientific and technological scenario boosts the birth of new research towards in-air gestural interfaces.

The efforts on this work are to systematically classify in-air gestures found in scientific papers to map how the HCI research community has been using in-air gestures in the last ten

years[1]. As a research area matures there is often a sharp increase in the number of reports and results made available, and it becomes necessary to summarize and provide an overview. The mapping provides a tool for future consults and inspirational material in the *Conception* phase of further developed gestural interfaces. Additionally, this mapping enables further analysis regarding the use of in-air gestures by the HCI community. We later expose our mapping results on a web catalog (Section 3.4) for interactive consultation and analysis.

3.2.1 Papers gathering and screening

The methodological choices of our Systematic Mapping are based on the process proposed by (PETERSEN et al., 2008). The process steps are illustrated in Figure 3.1 and described in the following subsections.

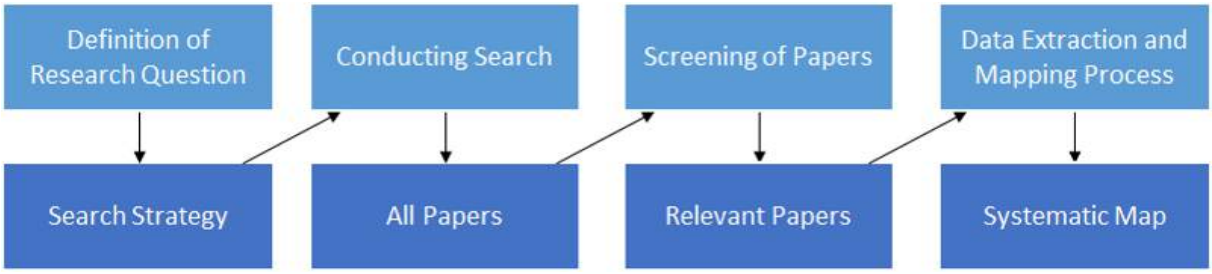

Figure 3.1: Systematic mapping process. The research question guides the definition of the search strategy, which is used to collect the works. Some criteria are defined to select the relevant studies that are classified in order to provide the systematic mapping.

The goal of this Systematic Mapping is to provide an overview of the current researches on the use of in-air gestures by the HCI research community. In particular, we are interested in hand gestures, which can include all sorts of hand movements and shapes. Full-body gestures, leg gestures and head gestures, for example, are not of interest to this research. Besides, our research targets gestures that are being explicitly used to accomplish some task on an application interface. Cases in which gestures are solely recorded and analyzed are out of scope. We define the overall objective in the following research question: *How has the HCI research community been using in-air hand gestures in the last ten years?* Four scientific databases are considered on this Systematic Mapping:

- IEEE Xplore Digital Library;
- ACM Digital Library;
- Science Direct;
- Springer.

We chose these databases since they house scientific, peer-reviewed works from the most important conferences and journals in the HCI area as Magazine Communications of the ACM

[1]Ranging from five years before and five years after the launch of the 1st version of the Microsoft Kinect sensor.

(impact factor 3.621) and International Journal Of Human-Computer Studies (impact factor 1.293) for example. The search string presented in Code 3.1 is defined to search the selected libraries automatically.

Code 3.1: Search string used to gather papers on the cited databases.

```
("hand gesture") AND
("computer interaction" OR "user interface") AND
("user" NEAR "experience") AND
("validation" OR "evaluation" OR "experiment")
```

The first sentence regards to in-air hand gestures. The second one regards to the HCI domain, and the third line concerns the particular UX topic. The following sentence indicates the existence of HCI evaluation. These latter terms ensure that the gestures were used by people and not just used for testing a recognition algorithm, for example. After collecting the papers and removing duplicates, we excluded the ones that did not fit pre-defined parameters. We performed the exclusion on four screening phases. On the first phase, we used four criteria of exclusion:

- The study must be written in English;
- The study must have been peer-reviewed;
- The study must have made use of in-air hand gestures;
- The study is not a duplicate.

Due to the absence of subjectivity, a single reviewer performs this phase. The reviewer reads the publication venue where the paper was published and the dominant language of the work to decide if it will be rejected or not. The duplicates are automatically checked by comparing study titles and abstracts. The second screening phase aimed to exclude papers based on the defined scope based on simple criteria:

- Does not explicitly mention the use of gestures;
- Explores exclusively natural interactions focused on other body parts, not the hands;
- Does not explicitly point to concerns about interaction, user experience or interface design;
- Uses in-air hand gestures strictly for sign language;
- Uses touch-based gestures strictly for tangible interactions;
- Uses gesture recognition strictly for motion or social analysis.

In the second phase, to circumvent the subjectivity of inclusion or exclusion decisions, two reviewers screened each of the candidate papers. The reviewers read the title and, optionally,

the abstract to identify if the work falls in some of the topics listed above or classify it as a valid work. Papers having conflicting decisions from the two reviewers were reevaluated and, if needed, later discussed to reach a consensus.

In the third screening phase, two reviewers revisited each remaining paper classifying it in one of the three categories: relevant, uncertain and irrelevant. The paper is considered relevant if it describes the use of in-air hand gestures for interaction. The main difference of the third screening phase is that, instead of focusing on the easily identifiable criteria defined for exclusion on the second phase, they had to concentrate on the main criteria for inclusion, which sometimes is not visible at first sight. For example, the mentioning of specific suggested the paper presented related content to this research: Examples of these terms are: “multimodal interfaces”; “vision-based interaction”; “interactions in AR systems”; “3D sketch-based input”; “non-verbal communication”; “movement-based interactions”; and “solution that employs NUI”. These cases demanded a careful analysis of the Abstract and Title content followed by a discussion between the reviewers to determine if the paper should be included or not. On uncertain cases, the paper is sent to the next phase.

The remaining papers are candidates to be classified according to pre-established categories. The classification procedure is part of the data extraction step (Figure 3.1) and aims to facilitate further analysis and synthesis of the collected data. Two reviewers performed the classifying activity. We detail each category on the following subsection. Additionally, as fundamental requisites, we identified and noted a full description of the performed *gesture* and the accomplished *task*. In case one of these two descriptions is not detected, it means the paper is not relevant for our research. This way, while classifying, as the last screening step, for each paper, each reviewer double checked if that paper is relevant, once on this phase reviewers have access to the full content of the work. If the exclusion is considered, the reviewer discusses with his pair over the paper material to achieve a consensus.

3.2.2 Classification categories

A set of categories were defined to map all the gestures. Part of the defined categories is directly related to the use of the toolset, helping designers on their search and definition of which literature gestures should be considered on the *Generate* step. There is also part of the categories that are indirectly related to the toolset, or even unrelated to it. These last are included aiming the additional goal of helping the HCI community to understand how the paradigm of in-air gestural interaction has been handled so far, and what could be the next steps towards well-established interaction models. The classified categories are *Task*, *Source*, *Pattern*, *Form*, *Nature*, *Binding*, *Flow* and *Domain*. Each category description and use are presented below.

Task specifies what the expected system outcome is. The task here is defined as an atomic response. The task differs from the activity, as well as from the application domain. Consider the following example: designers are using our toolset to build an in-air gestural interaction

to provide a smart living room environment, allowing the user to advance the channel while watching TV by using a gesture. In this example, we classify the application domain as *Domotics* (smart houses), the activity as "watch TV", and the target task as "go to next channel".

Wobbrock et al.
Move a little
Move a lot
Select single
Rotate
Shrink
Delete
Enlarge
Pan
Close
Zoom in
Zoom out
Select group
Open
Duplicate
Cut
Paste
Previous
Next
Insert
Maximize
Minimize
Accept
Reject
Menu access
Help
Task switch
Undo

Piumsomboon et al.		
Transforms	Move	Short distance
		Long distance
	Rotate	Roll
		Pitch
		Yaw
	Scale	Uniform
		X
		Y
		Z
Menu	Horizontal	Open
		Close
		Select
	Vertical	Open
		Close
		Select
	Object-centric	Open
		Close
		Select
Editing	Insert	
	Delete	
	Undo	
	Redo	
	Group	
	Ungroup	
	Accept	
	Reject	
	Copy	
	Cut	
	Paste	
Simulation	Play/Resume	
	Pause	
	Stop/Reset	
	Increase speed	
	Decrease speed	
Browsing	Previous	
	Next	
Selection	Single selection	
	Multiple selection	
	Box selection	
	All selection	

Figure 3.2: Collection of 27 tasks used on an elicitation study for surface interaction (WOBBROCK; MORRIS; WILSON, 2009) (on the left), and 40 tasks on a similar study for AR interaction (PIUMSOMBOON et al., 2013) (on the right).

Although the definition of a task can be broad, there is some background work on synthesizing common tasks for interactive systems. Jacob Wobbrock and colleagues compiled 27 different tasks from desktop and tabletop systems for an elicitation study of touch gestures (WOBBROCK; MORRIS; WILSON, 2009). Piumsomboon and colleagues performed a similar study surveying 40 different tasks (divided among six categories) to interact with AR content through in-air gestures (PIUMSOMBOON et al., 2013). Both set of tasks are shown in Figure 3.2.

We introduce a revised set of tasks. The process of analysis and synthesis of our set of

tasks was performed in two steps. First, we analyzed the tasks shown in Figure 3.2 seeking for possible ambiguities. For example, the list on the right side of Figure 3.2 uses *Select* for both menu and selection categories, which favors ambiguity. While performing the elicitation studies, authors from both works (WOBBROCK; MORRIS; WILSON, 2009; PIUMSOMBOON et al., 2013) explicitly explained to users what was the expected outcome for each task, thus reducing the risk of users misunderstanding the concept of the target task. On our case, we are performing the reverse process of identifying what task the user performed. Therefore, ambiguity becomes more present as an issue.

After this analysis of possible ambiguities, we merged and renamed part of the tasks producing a first set of tasks. Then we used that set to classify the task category on all valid entries of gestures found in the collected literature. Once an unlisted task was found, we noted it down for a second analysis and possible inclusion of that task. Therefore, the addition of tasks in the second step is introduced to fill gaps from the previous set, being based on the found occurrences of in-air gestural interactions on the reviewed works.

Our proposed set of tasks is presented on Figure 3.3. The tasks are divided into eight different categories. The 8th category called *Specific* is introduced to handle the classification of tasks directed to a particular application and not a candidate for generalizations. We added a textual description for each task aiming to reduce ambiguity due to possible misinterpretations of the single word used as the identifier for the task. At last, we introduced a relationship column (*You may also look at...*). This intervention is a recommendation we leave for designers to while seeking for gestures for a particular task, give a chance considering the related tasks. These related tasks may present promising gestures or insights about interaction possibilities. These relationships are merely suggestions of where designers may find additional relevant content and can be considered a result of our acquired insights through the revision process.

Domain refers to which area of application the gestures are applied in. Apart from specifying the use of the gesture, the application domain differs from the target task because it specifies the environment on which the gesture was meant to be used, rather than the particular expected response from the system. While using the toolset, and consulting the literature catalog, the application domain information can be useful by providing insight to designers about gestures used on the same (or similar) domain as the one they target.

For this category we used the ACM Computing Classification System (CCS) (COMPUTING MACHINERY , ACM), a classification used to index works on the ACM database. We considered the category *Applied Computing*, and its subcategories as the source of this topic since we believe that all works analyzed are supposed to show applicability for their gestures. Some papers from ACM Digital Library are not indexed as *Applied Computing* by the CCS, i.e., the applicability is not specific and directed to a particular application domain. The ACM CCS can be found at `http://dl.acm.org/ccs/ccs_flat.cfm#10010405`.

For instance, if a work presents an evaluation of an algorithm for hand tracking and to do

Category	# Task	Description	You may also look at ...
Selection	1 Select single	Selects a single element of the context	Select multiple; Grab
	2 Select multiple	Selects multiple elements of the context	Select single; Grab
	3 Select box	Selects elements of the context within a defined box shape	Select all; Grab
	4 Select all	Selects all elements of the context	Select none; Grab
	5 Select none	Deselects all elements of the context	Select all; Release
Editing	6 Group	Group elements of the current context	Select box; Select all
	7 Ungroup	Ungroup elements of the current context	Select none; Release
	8 Insert	Insert a new element to the current context	Duplicate; Copy
	9 Delete	Delete an element of the current context	Cut; Close
	10 Cut	Remove a element of the current context and store it on the clipboard	Delete; Close
	11 Copy	Copy an element of the current context to the clipboard	Paste; Duplicate; Insert
	12 Paste	Insert the element stored on the clipboard to the current context	Duplicate; Copy; Insert
	13 Duplicate	Combined actions of tasks Copy and Paste	Copy; Paste; Insert
System	14 Activate	Activate (turn on) an entire system or a function/mode of the system	Open; Confirm; Play/Resume
	15 Deactivate	Deactivate (turn off) an entire system or a function/mode of the system	Close; Reject; Stop/Reset
	16 Help	Ask for system help (likewise hitting the F1 button on Windows)	Home; Open
	17 Home	Return to system initial state (likewise returning to Desktop on Windows)	Open; Help; Previous
	18 Switch	Switch between system modes, functions or views (likewise Alt + Tab on Windows)	Previous; Home
	19 Mute	Mute system audio	Stop/Reset; Close
	20 Unmute	Unmute system audio	Play/Resume; Open
Menu	21 Open	Opens a menu on top of previous context	Home; Confirm
	22 Close	Closes the menu and returns to the previous context	Reject; Previous
	23 Confirm	Confirm a choice offered on a menu (e.g. highlited menu option)	Open; Next
	24 Reject	Menu offers an option (e.g. incoming call) which can be rejected	Close; Previous
Configuring	25 Increase	Increase the value of a configurable numerical option (e.g. volume, speed, brightness)	Next; Move
	26 Decrease	Decrease the value of a configurable numerical option (e.g. volume, speed, brightness)	Previous; Move
Browsing	27 Next	Advance a fixed step changing the current option, view or context	Increase; Move
	28 Previous	Recede a fixed step changing the current option, view or context	Decrease; Move
Playback	29 Play/Resume	Starts or resumes a playback	Confirm; Open
	30 Pause	Pauses a playback	Stop/Reset
	31 Fast-forward	Fast-forwards a playback	Increase; Next
	32 Rewind	Rewinds a playback	Decrease; Previous
	33 Stop/Reset	Rewinds a playback to its start and pauses it	Pause; Close; Home
Transform	34 Move	Move object or view on 2D and 3D contexts (e.g. move a 3D avatar, scroll on a 2D map)	Manipulate; Increase; Decrease
	35 Rotate	Rotate object or view on 2D and 3D contexts (e.g. rotate a box, or a virtual camera)	Manipulate; Grab
	36 Scale	Scale object or view on 2D and 3D contexts (e.g. enlarge a 3D shape, zoom in on a map)	Increase; Decrease
	37 Manipulate	Perform the move and rotate tasks with a single gesture	Move; Rotate
	38 Grab	Attach 2D or 3D object or view to a controlling interface (e.g. hands, wand, joysticks)	Select single; Move; Rotate
	39 Release	Detach 2D or 3D object or view from a controlling interface (e.g. hands, wand, joysticks)	Select none; Stop/Reset
Specific	40 Other	Other specific tasks (e.g. make a call, crop image, make avatar crouch and shoot)	-

Figure 3.3: Proposed set of tasks, including their categories, description and relationships (which may be considered on the *Generate* step).

that it presents a case in which a set of gestures are used to control a game, these gestures can be classified as *Computer Games*, which is an *Applied Computing* category. On the other hand, some papers approach the topic of in-air gestures, but with no mention to application domains or any suggestion of use. In these cases, the correspondent gestures were classified as *Not given*.

We also extended the application domain classification with two more subclasses: *Automotive*, for cases in which gestures are used in a car environment; and *Domotics* for cases in which gestures are applied in home automation systems. This addition is applied to prevent classifying these cases as generic classes such as *Computers in other domains*, and this way avoid that this information gets lost in the classification process.

Source specifies the origin of the conceptualized gestures for each revised work. Likewise, we proposed to diversify those sources; this diversification can be found in the revised works from literature. Researchers can be the source of gestures in their work, by using creative techniques or arbitrarily suggesting their concepts. Additionally, the found gestures can have the origin in user studies (e.g. gesture elicitation studies). Lately, some works may take a gesture from the literature, or even from real applications. All these categories are used to classify the gesture source; if the revised work does not mention the origin of the gesture concept, we classify the source as not available (N/A).

Consulting the source of a particular gesture concept can be useful for designers. For

example, if the source is classified as *Users* it may reveal a gesture elicitation study previously conducted aiming a similar scenario for the same target task. These findings may save the effort of performing additional elicitation studies. If the source is a *Application*, it indicates that the gesture is already used on a product, which can add relevance as it is being tested in real case scenarios.

Pattern represents the relation between the performed gesture and the accomplished task. Reusing an established Pattern can potentially be well accepted by users. Therefore, designers may take these cases into consideration while selecting gesture concepts to store on the *1st Pool*. Even if it is not possible to point to an established model of interaction based on gestures yet, some interaction patterns can be extracted. (SHEDROFF; NOESSEL, 2012) highlight 7 Patterns of gesture-based interaction from Sci-Fi movies. Although these patterns come from the futuristic film industry, the same interaction can also be found in real applications. These patterns are taken to the present mapping. They are the following:

- *Wave to activate:* to shake hands over something to wake up or turn it on.
- *Push to move:* moving something by pushing hands towards this thing.
- *Extend the hand to shoot:* shooting by pointing the outstretched arm towards the target.
- *Pinch and spread to scale:* making a pinch movement with fingers or hands (moving them towards or away from each other) to zoom in/out or to resize something.
- *Turn to rotate:* rotating something by pushing hands or fingers to the opposite direction around a specific axis.
- *Point or touch to select:* pointing or touching an object with the fingertip or even the hand to select, indicate or turn it on.
- *Swipe to dismiss:* swiping the hand away from the body center to dismiss/close an informative window or discard an object.

Form is also considered. Wobbrock and colleagues classified surface gestures into 6 categories regarding their *Form* (WOBBROCK; MORRIS; WILSON, 2009). They are:

- Static pose
- Dynamic pose
- Static pose and path
- Dynamic pose and path
- One-point touch
- One-point path

Their work focused on touch and surface gestures. Since our study is focused on in-air gestures, these categories were adapted. We propose a similar classification by separating static gestures, when *fingers*, *palm* and *arm* poses are held; and dynamic gestures, when there are changes in any direction of these body parts. Once a single body part is dynamic, the entire gesture is considered to be dynamic because at some point something moves. Thus, we introduce a revised classification to gather nuances more detail about the interaction. For a given gesture, we classify each of the three different body parts (fingers, palm, and arm for both body sides) as *static, dynamic or unused*.

By classifying if the arm is dynamic or not it is possible to identify the path concept explored by Wobbrock and colleagues (2009). Also, it is possible to define in advance which sensors and algorithms are likely to be capable of tracking the needed body parts for the gesture recognition. Therefore, if there are already mandatory technical limitations to be considered, the designers can use this information to aid the process of selection of gesture concepts to be prototyped. At last, it is also possible to identify if the gesture uses both hands (or arms), or if it only requires one side to be performed.

We present the information of each body part classification using a combination of characters as shown in Figure 3.4. The presented code is used to visually present the information of the six categories combined (forms of fingers, palms, and arms from both sides of the user). While classifying the *Form*, instead of referring to each side of the user body as *left* and *right*, we refer to the 1st and 2nd sides, being the 1st side the one first used or mentioned on the gesture description.

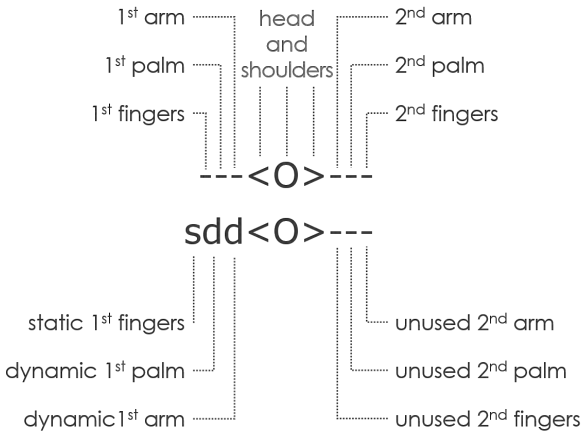

Figure 3.4: Code used to synthesize the gesture *Form* across different body parts. The code represents the user head (in the center represented by an "O"), shoulders (represented by the signs "<" and ">"), plus arms, palms and fingers (represented by a dash "-" when unused, a "s" when used statically on the gesture, and a "d" when used dynamically).

Flow is defined regarding the relation between the system responses and the user acts. It is analogous to the *Form* category. In a similar way the *Form* describes how dynamic, or static is the user gesture, the *Flow* class describes how static or dynamic is the system response. The

system response may occur either after an action is triggered by the user, or it may begin at a particular moment and keep going on while the user acts. Therefore, the interaction, regarding its flow, can be named as *discrete* or *continuous*.

The *Flow* is discrete if the system feedback initiates after the user finishes the movement. An example is to trace a question mark ("?") to bring up help. *Flow* is continuous if ongoing recognition is required, such as with manipulation gestures. It is important to notice that a Static gesture can still have a Continuous *Flow* outcome, which occurs on sustained actions while the user maintains a static pose through time, and the system provides continuous feedback. A common example is the *selection by delay* present on some Kinect-enabled games for Xbox. On these cases, the user must keep a static pose while pointing to the desired option, and the system usually provides continuous visual feedback (e.g. a circle being radially filled) until the action is fully completed.

Nature is a class responsible for the type of message or abstraction represented by the gesture. It relates the action to the accomplished task, and the gesture alone is not sufficient to define it. Wobbrock and colleagues (2009) classify gestures into four dimensions, including nature quality in which gestures are arranged according to their type of action needed to perform a task. Although their taxonomy is well known and applied by the community, their work is focused on touch-based interactions. Aigner and colleagues (2012) propose a set of gesture types focusing on in-air gestures.

On our work we analyze and combine these two sets into one, merging their similarities and keeping their differences, and hence making a classification that extends these two contributions as can be seen in Figure 3.5. Both categories Symbolic and Semaphoric were understood as the same type of gesture changing only the context where applied (tabletop and in-air). The same happened with Physical and Manipulation, both treating the same kind of interaction but in different contexts. The nomenclature kept was the considered most used to the community: Symbolic and Manipulation. The others categories kept the original name. All nature classes are detailed below.

- *Manipulation* gestures feature a tight relationship between the movements of the user and the movements of the object to be manipulated. To be classified as manipulation, the gesture should produce the same effect that would be produced if the movements were applied to a real object.

- *Pointing* gestures indicate objects and directions, which does not necessarily require an outstretched index finger. Gestures that resemble pointing but are intended to mean "yes, you got it" are labeled as symbolic, given the loaded meaning.

- *Metaphoric* gestures should act like something and not be applied to the realized task. Examples of metaphoric gestures are gestures in which the user body is part of the

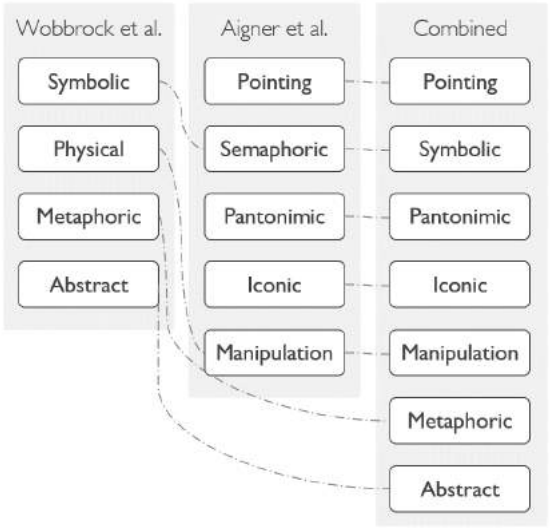

Figure 3.5: Classifications (WOBBROCK; MORRIS; WILSON, 2009; AIGNER et al., 2012) used to define the combined nature category, and the relationship between them.

object to be manipulated. Examples are moving the fingers as if walking, sliding as if turning pages of a book and shooting as if the hand is a gun.

- *Pantomimic* gestures are used to demonstrate or imitate a specific task to be realized. The gesture is performed exactly like the movement which should be performed to accomplish the task. An example of a Pantomimic gesture is to play an imaginary violin, pretending to hold it like it was real.

- *Iconic* gestures are used to communicate object information, as size or form, being usually slower than Symbolic gestures. To be classified as iconic the gesture should graphically represent the meaning. As an example of iconic gestures, gather equivalents fingers of both hands to represent a circle or draw a picture with the fingers.

- *Symbolic* denotes hand postures that have a specific meaning and visually represent a symbol. They depend on the background cultural knowledge of the user, for example, a thumb up meaning “ok”, shake hands to both sides meaning “no.”

- *Abstract* gestures have no relation with other types of classification and usually have no connection with the outcome of the gesture. For example, knock three times on an object to delete some feature of a system.

Binding relates the performed gesture to the context presented by the system. According to Wobbrock and colleagues (2009), can classify gestural interactions as:

- *Object-centric:* interactions that only require information about the object they affect or produce. An example is pinching two fingers together on top of an object for a

shrink.

- *World Dependent:* defined in the world, such as tapping in the top-right corner of the display or dragging an object off-screen.
- *World Independent:* interactions that require no information about the world, and generally can occur anywhere.
- *Mixed Dependencies:* interactions that occur for gestures that are world-independent in one aspect but world-dependent or object-centric in another.

The afore mention classification for *Binding* brings insight about how the gesture is related to the context (divided by the notions of Object and World). This classification may reveal patterns of particular associations between gestures and tasks (e.g. Manipulation tasks are usually Object-centric), and these patterns may be of interest to designers as insight for generating new gestures (and interactions), or else on the selection process.

Nevertheless, this classification is strongly related to multi-touch gestural interactions and part of the concepts are not directly applicable to the context of in-air gestural interactions. The concept of World for touch-based interactions is directly related to the used touch surface or display. On in-air gestures interfaces, there may be displays. However, the interaction is not limited to it. While using in-air gestures, it is not practical to have a well-delimited boundary between the virtual environment and real world. This way, we reduced the number of classes by removing the concept of World Dependency (for both positive and negative cases). Therefore, for this mapping, gestures may be *object dependent*, when the gesture location is defined by object features (e.g. pinching to shrink) or *object independent* when gesture location is not directly linked to the target object, menu or screen.

3.2.3 Results

Our search returned a total of 438 papers. The first screening phase eliminated 16 papers, 14 of them because they were published in a non-peer reviewed venue and two because they were duplicate works. In the second phase, 218 papers were rejected based on exclusion criteria. 204 papers passed to the third phase. On the third phase, 33 papers were removed. In the fourth phase, the reviewers had access to the full content of each paper, and hence the task of filtering them was more exhaustive. In the end, 65 papers were also excluded because falling into exclusion criteria after a more thorough reading. In total 106 papers were included and classified. Figure 3.6 shows the screening results per database, pointing the share of each database on the first set of gathered papers and the final set of filtered and included papers.

The 106 classified papers were published in 78 different forums. 37 (47.4%) being journals, 38 (48.7%) conferences and symposiums, and 3 (3.8%) workshops. Figure 3.7 shows an increasing trend of research focused on in-air gestural interaction. It is possible to see a

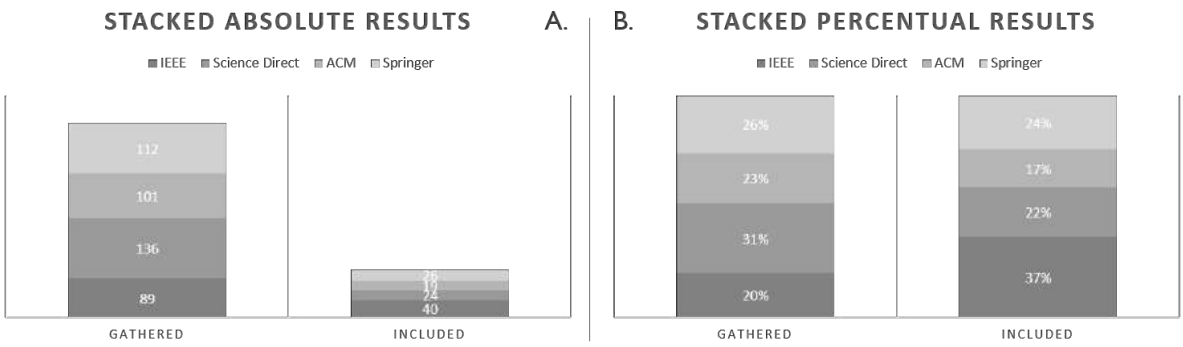

Figure 3.6: Screening results per database. On part A the number of papers gathered and included (or filtered) per database is shown as on stacked bars. On part B the same result is shown in percentages of the total, to highlight the participation of each database on the total of gathered and included papers.

significant growth in the number of published works starting from 2010. This indicates an increasing interest in the use of gestures by the academy in recent years, coinciding with the launch date of the Microsoft Kinect sensor. On each paper, more than one gesture can be used. Each one of the identified gestures was classified separately, which resulted in a total of 639 gesture entries (an average of 6 different gestures per paper). In addition, we show the classification results on figures 3.8 and 3.9. We describe the results for each category on the following paragraphs.

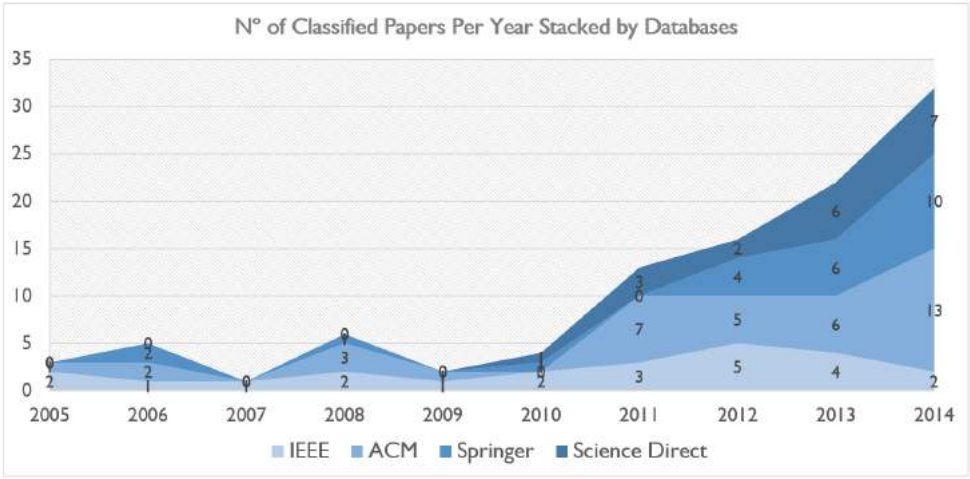

Figure 3.7: Classified papers over time, separated and stacked by its origin database.

Task The top of Figure 3.8 shows the distribution of the task categories on each reviewed interaction. Most part (37%) of the experimented tasks were *Transform* tasks, such as *Move* and *Rotate*. We found that researchers propose interactions for both changing an object of the current context (e.g. moving, rotating or scaling the object), but also to changing the viewpoint (e.g. moving, rotating or zooming a camera). Additionally, 8% of the interactions had as the outcome a task that was considered too specific. Although these cases were diverse, we observed some

Figure 3.8: First set of main classification results.

repetitive occurrences, for example, the use of a gesture to accomplish the task of *shooting* (e.g. on a game). These cases may represent interaction patterns pursued by the community and thus, deserve a further expansion of the tasks categories.

Source Regarding the sources of hand gestures found in the works analyzed, we observe in Figure 3.8 part A that the majority (60%) of hand gestures are conceptualized by researchers (authors of the reviewed papers). While examining these cases, few of them mentioned the use of creative techniques to generate the gestures, which may indicate that authors usually suggest gestures without an explicitly defined methodology for the *Conception* phase. There was also a range of works (23%) which engaged users in the process of conception. Few works reproduced gestures found in the literature and existing applications. Lastly, more than 7% of the analyzed cases do not report how the used gestures were conceived.

Pattern In total, only 15.3% (Figure 3.8 part B) of the classified gestural interactions did fit in some of the patterns established by Shedroff and Noessel (2012). The vast majority were found being too complex, abstract or not yet established into the patterns list. This may point out that there is no clear model of in-air gestural interaction yet. Regarding the cases that fit in one of the patterns, there was a predominance in the use of *"Point or touch to select"*, occurring on 8.74% of the cases.

Nature In the gesture nature classification, part C of Figure 3.8 shows that a considerable amount (22%) of abstract interactions were found, being the second most common case. Abstract gestures are the ones considered with no defined relationship towards the intended action, representing the cases where no other nature was identified. Abstract gestures usually present compromised memorability (being hard to associate the gesture with the target task concept). This may present us a concern towards the generation process adopted by several works on the Academy.

Form Regarding *Form* of the performed gestures, the majority (75%) were classified as dynamic (Figure 3.8 part D.1), i.e., the system needs to detect a trajectory to trigger an action. To be named as dynamic, a gesture should require at least one moving part, which can be the user fingers, palms, or arms. On Figure 3.8 part D.2 it is shown that only 14% of the gestures required both user sides (left and right). On part D.3 of Figure 3.8 it is illustrated the stacked percentage of each body part used to perform the gesture. The most of the Dynamic cases occurred on the 1st (or dominant) arm, while palms and fingers were static or unused. These findings may point to which tracking devices and technologies are being explored by researchers. For instance, the 2nd version of the Microsoft Kinect is capable of tracking the user arm, palm but not all fingers (just the thumb and hand tips). Moreover, it is noticeably more precise while tracking the arm than the other parts.

Binding Classified gestures are almost 50/50 (Figure 3.9 part E) spread among dependent and independent regarding the object binding.

Flow Although gestures are mostly dynamic according to their *Form*, 70.65% of the gestures *Flow* is discrete, as shown in Figure 3.9 part F. This means that even if some gestures need that users make movements, the feedback from the system is only prompted when the user ends his actions.

Domain The gestures were spread among 18 application domains. On part G of Figure 3.9 it is shown the distribution of these domains, specifying the seven most common cases. The vast majority of them were classified as *Computers in other domains*, which includes entertainment applications and applications supported by robots. *Personal computers and PC applications,*

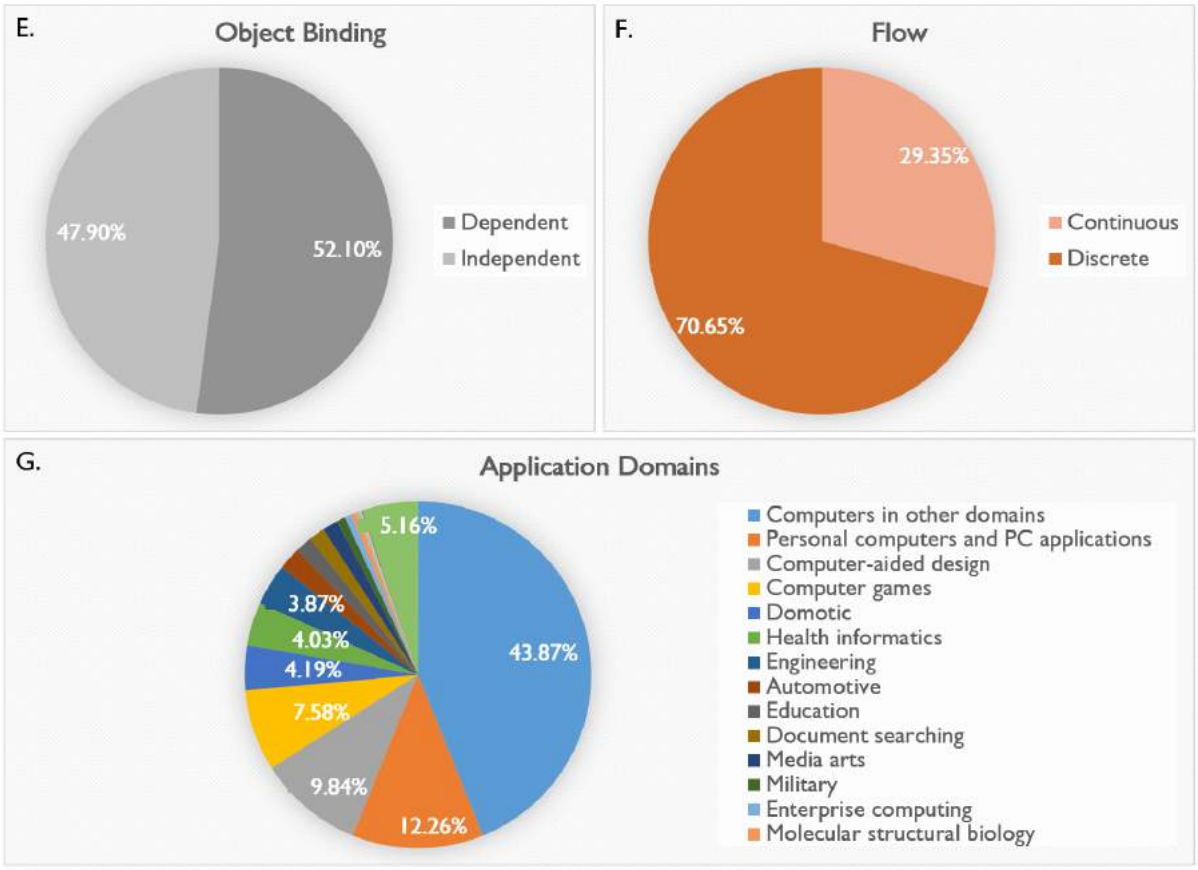

Figure 3.9: Second set of main classification results.

Computer-aided design, Computer games, Domotics, Health Informatics, and *Engineering* are also present in a considerable amount of works. 5.16% of the entries did not provide enough information to be classified regarding its application domain.

3.3 Sci-Fi gestures

In addition to reviewing and classifying the use of in-air gestures on the HCI literature, we also examined this use on Science Fiction (Sci-Fi) titles. In Sci-Fi movies, filmmakers try to anticipate trends and new forms of interaction (SCHMITZ; ENDRES; BUTZ, 2008). Some producers and directors often use emerging interaction paradigms to create a plausible vision of the future. Metaphors are created allowing their characters to interact with futuristic devices and environments, such as gesture-based interaction. These devices and metaphors are targets of research considering they have proven to be useful before. In this case, the goal of our study is to collect and classify a compilation of gestural interactions in Sci-Fi titles, as shown in Figure 3.10, providing a catalog (detailed in Section 3.4) as a resource for the *Conception* phase.

Given that moviemakers have resources and freedom to create characters, stories, and scenarios, without current limitations of the real world, Sci-Fi movies can present a particular vision of the future that we are not familiarized with. This can help to make new technologies

Figure 3.10: Examples of movies that compose the used dataset in this study.

and interaction models widely known to the general public, contributing to popularize their adoption and to highlight aspects that can be improved by the industry and academy. On the other hand, the Sci-Fi media has potential to reveal or emphasize research trends regarding particular interaction paradigms, devices, and interfaces for specific tasks and application domains. Once singular visions of filmmakers are well accepted by the audience, the research on the same topic is boosted by additional motivation, the anticipated technology starts to be part of the public imaginary and an inner curiosity grows with questions like "how would this work in real life?".

In the other hand, on the filmmaking industry, a set of additional concerns may emerge over the Future User Interface (FUI) design. Examples are the visual appeal of the scene; the plot development; and how the character will look like while using the new interface. These concerns may not intersect with goals like realizing the targeted task efficiently or providing a good experience for the user. Despite the industry influence on the FUI design, the scene content is valuable for a range of research purposes, for example, the design process can take it as a source of inspiration. The FUI presents pre-made concepts to designers and developers to create solutions that can be both built using current technologies, or at least a start point to explore new concepts. Prototyping techniques like the Wizard of Oz can benefit from visuals to aid the setup definition including the particular task of designing FUIs, which demands knowledge about similar interface concepts that have been designed before.

We also found previous related work presenting efforts on surveying and analyzing content from Sci-Fi titles. Aaron Marcus (2015) selected scenes containing the use of FUIs from around 20 Sci-Fi movies along the last 100 years. His study is organized chronologically according to the prediction date of an interface (date of which the interface first appeared on the screen) and release dates. Schmitz and colleagues (2008) conducted a similar analysis, categorizing the scenes according to their application domains and relating them to current

technologies and prototypes under research. Shedroff and Noessel (2012) analyzed around 30 Sci-Fi movies and present lessons and insights about the interactions shown on them to HCI, pointing to the existence of seven patterns. Tulloch and colleagues (2013) focus on good practices for observation techniques aiming the creation of better interactions. For this purpose, their work considers Sci-Fi media (e.g., scenes from the StarTrek movie), analyzing gestural, like touch and voice interactions.

Additionally, there are works that explicitly expose the connection between research and Sci-Fi productions. For example, on "Twelve Tomorrows" (PAOLA ANTONELLI; FINLAY et al., 2016) writers create science fiction inspired on a technology review performed by MIT. In "Future Visions: Original Science Fiction Inspired by Microsoft" (ELIZABETH BEAR GREG BEAR; KRESS, 2015) writers had access to leading-edge work from Microsoft Research as inspiration, providing short stories relating predictions of future technologies and interfaces to human relationships. The cycle of influence between fiction and reality on the development of technology and science is becoming more explicit, motivating the research of this connection and its use on the conception of new interfaces.

3.3.1 Search and screening

We recruited 25 volunteers using email lists and asked them to inform any movie they remembered that had any gestural interaction using hands or arms. We performed the same exercise. This procedure, although being subjective, has the advantage of pre-filtering titles by consulting the audience imaginary. We have no goal of reviewing all Sci-Fi titles. Instead, we focus on providing a first valid set of selected movies, to represent an initial set that is large enough to provide an analysis basis as well as a test set for the proposed tool.

At the end of this first step, 10 collaborators indicated over 200 titles. These indications were checked to be classified as Sci-Fi at IMDb portal (`www.imdb.com`), which is a popularly known movie database. We also considered one alien and two superhero movies. These movies have scenes with a type of interaction explained through the use of telekinesis[2], and supported by hand-based gestures. It was also checked whether they had the kind of interaction this research was looking for.

The remaining titles were restricted by their release date. We considered 20 years of Sci-Fi titles, from 1995 to 2014. We focused on these earlier years aiming to gather updated FUI concepts. Therefore, the latest titles were prioritized to reduce the scope of the work. This choice was made because newer movies are often inspired by the new emerging technologies,

[2]The concept of telekinesis can be defined as "the power to move objects from a distance without physical contact". That means some characters can have the ability to move matter using their minds and hands. We collected these scenes because we believe these interactions may show gestures beyond the usual paradigm. In many cases, the hand gestures from these movies are not related to computer interfaces. In these cases, they do not have any input or output devices since the most gestures are performed in fight scenes and use telekinesis approach. Nevertheless, some gestures use established patterns of gesture-based interfaces, for example, "Extend the hand to shoot" and "Push to move." In these movies, we have the opportunity to look forward to new forms of interaction, and we consider that these scenes may provide us a rich source of inspiration to create innovative solutions.

such as gesture-based interaction, giving them new insights to go even further. Additionally, there are already studies regarding the older titles (in the future, they may also be added to have a comprehensive database). Finally, the resulting subset contained 24 movies.

In sequence, we watched the selected movies, aiming to identify the exact time frame the gestures occur during each film. To analyze them all in a reasonable rate, they were watched up to 8x of the normal speed. For each scene with hand gesture found, the starting and ending time were annotated. With this in hand, a script was written on top of the the FFmpeg library (`www.ffmpeg.org`) to isolate the scenes noted before. This allowed us to analyze the scenes separately. In total, we extracted 229 different scenes containing hand gestures.

3.3.2 Classification and results

For the acquired Sci-Fi scenes we performed a similar classification as the one performed for the literature papers (described on Section 3.2.2). Some categories were not considered for the Sci-Fi scenes for different reasons.

The *Source* category is untraceable in most cases because the movie description does not detail which was the source for the background material to create the gestural interfaces. The *Nature* of the gesture presents a similar obstacle because filmmakers do not express the gesture definition in words; the only instance of the available content of the gesture is the collected visuals. Therefore, it is usually difficult to decipher a hidden metaphor for example. Considering these limitations, we focused the classification of each scene on the following categories: Task, Pattern, Domain, plus Input and Output devices.

As pointed above, we added two categories for the Sci-Fi scenes in comparison with the literature catalog. These classes are *Input* and *Output* devices. *Input* devices are identified if the scene presents a device to track user actions (e.g., Haptic Gloves). *Output* devices are identified by the device used to give feedback to the user/character (e.g., Holograms, HMD, and CAVE). These categories may represent insight about the required devices and technologies to make such interaction a reality. We also have some categories pending classification, which are: *Form*, *Flow*, and *Binding*. The *Input* and *Output* categories are pending classification for the literature catalog. The results of the classification are shown on Figure 3.11 and detailed below.

Task The top of Figure 3.11 shows the tasks categories divided by the percentage of occurrence in each case. The most present (42%) type of tasks was *Transforms*, followed by *System* tasks, such as activate a system or switch system modes.

Pattern Figure 3.11 shows the occurrences of each pattern on the cataloged scenes. We identified a 8th pattern while analyzing the scenes (named “Knock-knock to turn on”). During the categorization process, 37.83% of the analyzed scenes contained some interaction that could be embedded into a pre-established pattern. On the other hand, the majority of interactions found were too complex or not established. There is a part of the uncategorized gestures related to

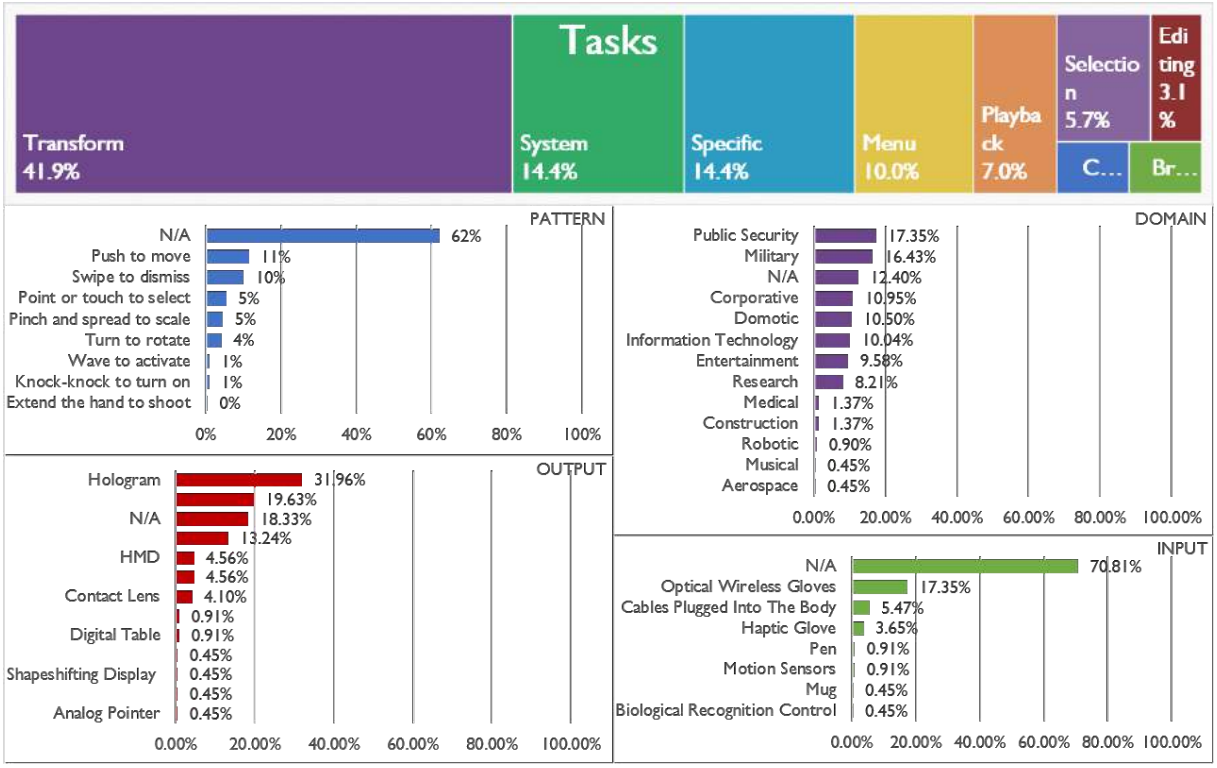

Figure 3.11: Occurrences for each classification topic for the collected scenes.

specific activities that humans are used to perform, and these gestures were, mostly, exactly as it would be done in the real world. This finding may point to a broad and adaptable pattern that describes in-air gestural interactions as real world physical interactions. Another significant part of the interactions without patterns shows the use complex gestures using full movements of arms and hands.

Domain We found an extensive set of application domains within the analyzed dataset. The predominant application domains were Military, Public Security, Corporative and Domotics (see Figure 3.11). In addition to these domains, it was also perceived the targeting of HCI applications in the areas of Information Technology, Entertainment and Medical applications. This analysis shows the main areas filmmakers are directing their attention regarding FUIs.

Input Regarding the input devices, our findings show that the most common case is the absence of identification of any input device (70.81% of the scenes). This result can be interpreted as the idea that the tracking and recognition are anticipated as being invisible for the audience (and possibly from the user/character). This may point to an expectation of filmmakers that, in the future, we will not need any attached physical device to interact with, nor the tracking device will be explicitly visible in the surroundings of the user. In the cases a device is identified, the most used were gloves, optical wireless and haptic ones (21%). Figure 3.11 illustrates the occurrences of each found input device.

Output The output devices showed a wider variety, as shown in Figure 3.11. In part, these devices coincide with what, nowadays, are considered high-tech display environments, such as HMDs and CAVE environments. The output device mostly proposed by filmmakers is the volumetric projection or, as it is commonly known, the hologram projection, found in 31.96% of the scenes. The hologram has been studied as a substitute for physical prototypes and as an alternative to 3D displays, and it shows a supposition of what the filmmakers guess the society expects from new output devices.

Other aspects were also classified, as, the type of feedback (being audible, visual or tactile) and if the gesture was used for manipulation tasks. At last, we also added a full description of the found gesture and the accomplished task. These and other aspects can be filtered on the website tool at `http://goo.gl/XSX5fn`. Further detail about the cataloged data can be found in the paper entitled "Sci-Fi Gestures Catalog" (FIGUEIREDO et al., 2015).

3.4 Catalogs

We present all the collected, reviewed and classified data from both HCI literature and Sci-Fi titles on online catalogs. The main purpose of these catalogs is to give easy access to designers using our toolset. The literature catalog is available at `http://goo.gl/frpBpA` and the Sci-Fi catalog at `http://goo.gl/XSX5fn`. These catalogs are open source web applications, allowing collaborators to not only consult the presented data but also to contribute to its implementation. The source code is stored in a GitHub repository and can be found at `http://goo.gl/IpcAfl`. Also, both catalogs are collaborative systems, allowing visitors to contribute by suggesting the addition of new data to it.

We stored the gathered data in its completion using Google Spreadsheets (`www.docs.google.com/spreadsheets`). In general, spreadsheets are an easy platform to collect, organize and edit data. Therefore, we wanted to maintain the use of a spreadsheet as a storage platform even after the classification procedure was finished. This way, we can easily add new entries, refine the stored data, and edit it once issues are found. Additionally, the use of spreadsheets is well spread, and all the researchers enrolled on our revision and classification process were skilled regarding its use. The same does not apply to other data storage technologies.

Following this goal of using a collaborative spreadsheet as a data source for the online catalogs we developed a library using *tabletop.js* (SOMA, 2014). Our library provides real-time access to the spreadsheet data, using it as database resource to the web page interface. Therefore, any changes made on the spreadsheet are automatically presented on the page. Figure 3.12 shows snippets of the used spreadsheets, having one interaction entry per row, and one category per column. In addition to storing the data, by using our library, the spreadsheet also becomes a configuration tool, defining some aspects of the web interface. The configuration tags and corresponding options are highlighted on Figure 3.12. These options are listed below:

- Title: defines the column content as title of the row.

- Filterable(select OR checkbox): allows viewers of the web interface to filter the column content by typing (select) or to check (checkbox) the desired options.
- Sortable: allows viewers to sort the content based on the category of that column.
- Type(Text OR Numeric): allows the web interface to provide the feature of creating charts for the specified type.
- Placeholder(text): showing a placeholder text for the a *text box* filter.
- Video: uses the Youtube (`www.youtube.com`) code to show the video content on the web interface. This tag is useful to allow the presentation of scenes collected from the Sci-Fi titles.
- Hidden: allows a column to be present on the spreadsheet but entirely ignored by the web interface.

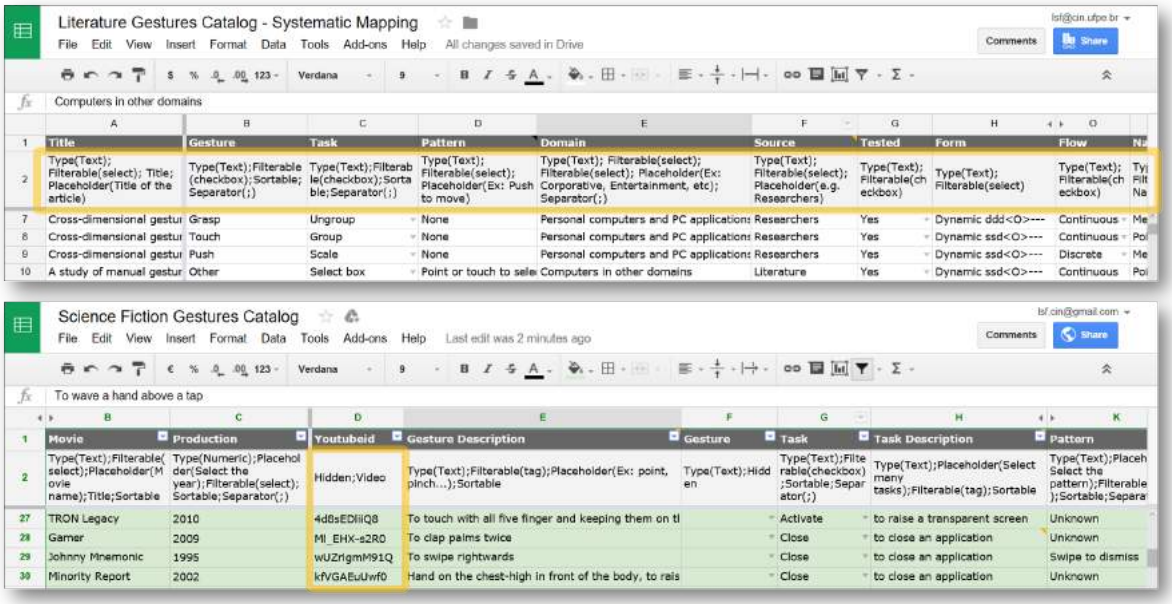

Figure 3.12: Snippets of the data source spreadsheets.

As additional features, the provided library creates a web page with two visualization options for the stored data: a table, mimicking the spreadsheet visuals (top of Figure 3.13); or a set of cards, being each card a representation of a classified interaction (bottom of Figure 3.13). While using cards, it is possible to see the provided Youtube videos as highlighted in Figure 3.13. Our interface also contains selectable filters. The developed library is available in an open source GitHub repository at `www.github.com/voxarlabs/catalogs-api`.

We also understand that to create an expressive compilation of both literature works and Sci-Fi scenes containing gesture interactions is an ambitious goal, and our initial dataset represents a small sample of the target goal. Taking it into consideration, we introduced the “Add

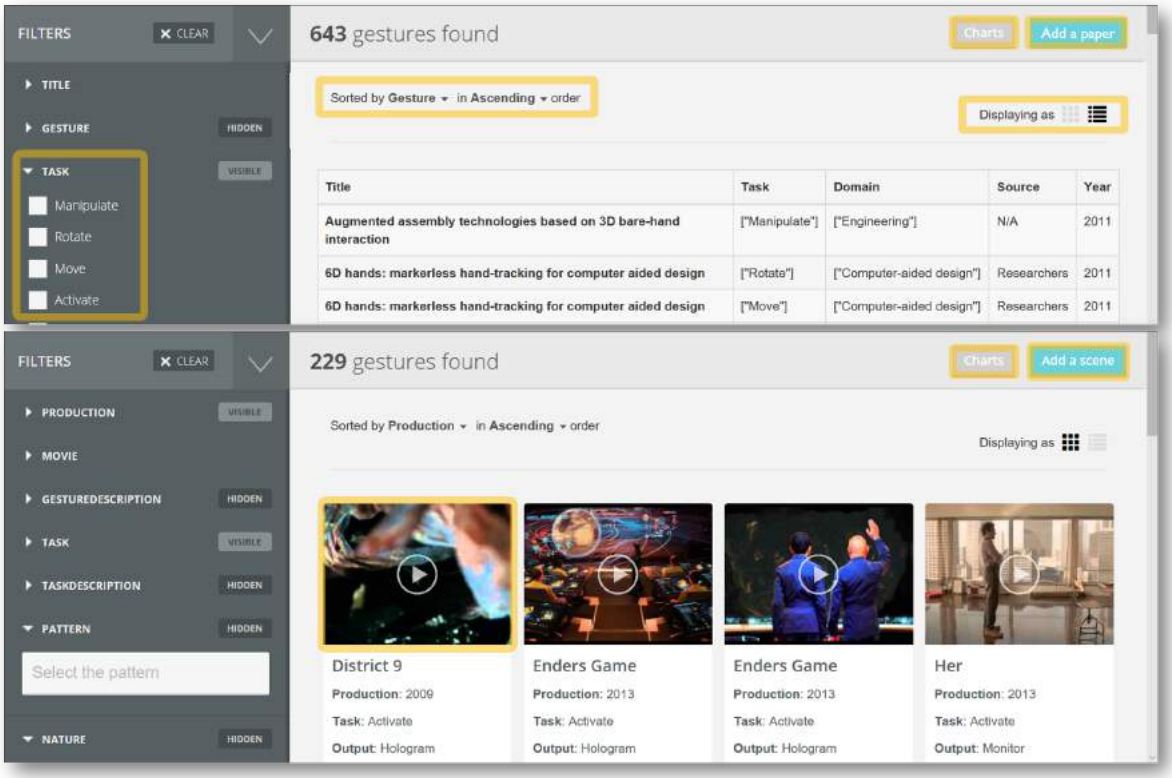

Figure 3.13: Snippets of the web catalogs.

a paper" and "Add a scene" button on the top part of each interface (Figure 3.13), through which collaborators can send new entries of gestures found in papers or movies scenes. Each new entry is revised by us to check if it fits the screening and revises the classification criteria, to later paste it on the online spreadsheets used as data source for the catalogs.

At last, the web catalog introduces a feature for generating charts, which can be further used for analysis. The "Charts" button (shown on Figure 3.13) can be used to switch to the graph generation mode. Figure 3.14 shows a chart being created using the year category of the mapped literature entries. Charts can also be configured to cross categories or to represent the frequency of a single one; they also can group (merge) smaller occurrences or ignore them. The visual of the chart can be chosen on the "Chart type" selector. Viewers can stack the created charts by hitting the button "Add chart," or download it in different formats.

3.5 Conclusion

In this Chapter, we focused on providing tools for the *Generate* step of the design process. The provided tools are presented as two web catalogs, which can be used for consultation, analysis, and inspiration while creating gesture concepts for the target task. To build these catalogs, we collected several examples of used gestures both on HCI literature and on Sci-Fi titles. We reviewed and classified 106 papers, and 24 movies, which contained in total 868

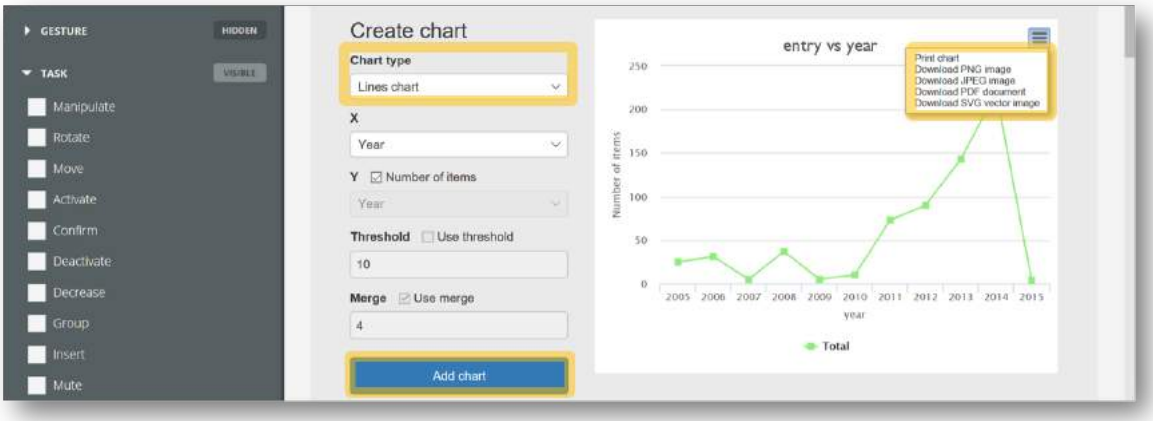

Figure 3.14: Screenshot of the chart maker session on the developed web interface.

entries of in-air gestural interactions, being 639 from the literature and 229 from Sci-Fi titles. These examples were cataloged according to our expanded taxonomy based on the works of Wobbrock and colleagues (2009) and Piumsomboon and colleagues (2013). All the acquired data is available on a web platform that is designed to empower researchers on the task of filtering and analyzing text and video content. We also provide a JavaScript library that allows the direct visual presentation of contents available on collaborative spreadsheets. This library is already being used for other applications as shown in this catalog of revised papers about mobile tracking algorithms `www.cin.ufpe.br/~rar3/tracking_sm`, and in this portfolio of technologies developed at the Informatics Center of the Federal University of Pernambuco `www.cin.ufpe.br/~cinove/portfolio`.

4

Prototype

After generating and selecting a set of gestures, we propose designers use techniques and tools for prototyping. This Chapter details our directions for low-fidelity and high-fidelity prototyping. We suggest that designers first conduct a low-fidelity prototype experiment using the Wizard of Oz technique (HANINGTON; MARTIN, 2012), testing the usability (and other desired aspects) with users. Later, for the interactions that were approved by users, we propose the use of a high-fidelity prototyping tool called Prepose. We introduce Prepose as a tool for rapid prototyping allowing designers to create gesture recognizers using natural language through a Domain-Specific Language (DSL). Once high-fidelity prototypes are built and tested, the designers have the outcome of our toolset. We approach the details and limitations of the use of Wizard of Oz and Prepose on the following sections.

4.1 Low-fidelity prototyping: Wizard of Oz

The Wizard of Oz technique is well-known for its power in replacing not implemented features of the system by human actuators. Although it is often used for more turn-based interactions such as voice interfaces (as shown by Mignot and colleagues (MIGNOT; VALOT; CARBONELL, 1993)), Henze and colleagues also showed its application for gesture-based interfaces (HENZE et al., 2010). Nevertheless, according to our knowledge, there are no defined boundaries from the literature regarding the use of the Wizard of Oz technique for low-fidelity prototypes of in-air gestural interfaces, detailing which types of gestures it can handle, regarding aspects such as *Form* and *Flow*. Therefore, we explore the technique and test the extent of its applicability to interactions with continuous *Flow*. Moreover, the Wizard of Oz technique can assist the Participatory Design strategy, allowing users to create and perform commands in real-time by teaching the Wizards their desired interaction on the spot.

Several related works are using the Wizard of Oz technique to substitute a tracking and gesture recognition tool. The technique has been used to recognize gestures on different application domains such as robotics (to control a drone) (CAUCHARD et al., 2015), security (using gestures for access control) (KARLESKY; MELCER; ISBISTER, 2013) and entertainment

(to control a TV menu through gestures) (WU; WANG; ZHANG, 2014). Particularly, H́oysniemi and colleagues employed Wizards to recognize gestures in four different games played by children (HÖYSNIEMI; HÄMÄLÄINEN; TURKKI, 2004). This last was the only work found which targeted a fast pace interaction, presenting a timing constraint associated with the task. Other examples (WALTER et al., 2014; PERUSQUÍA-HERNÁNDEZ et al., 2014; CAUCHARD et al., 2015; BRAGDON et al., 2011) targeted turn-based interactions, in which the user is not encouraged to perform and repeat the gesture in a short amount of time, but to pause after an attempt and wait for feedback.

Until the moment, no work was found exploring a more sustained task (e.g., navigation in 3D environments). Furthermore, no work was found neither explaining the training method for the Wizards nor presenting a direct concern to measure the latency of the Wizards and to evaluate its impact on the interaction. These three topics are concerns in our experiment. We aim to push forward the use of the Wizard of Oz technique, to validate its use on continuous interactions, i.e. Continuous *Flow*[1]. Our concerns are directed to a more broad use of the technique, exploring cases that require low latency for the feedback. Therefore, the goal of this section is to examine and validate the utilization of the Wizard of Oz technique on such scenarios.

4.1.1 Task

Our target tasks in this case are *Move* and *Rotate*. We address the activity of navigating in virtual environments with fixed display visualizations (e.g., projections and TVs). To propose a more controlled experiment we use a specific path for users to follow. This prevents users of performing different navigation operations. The chosen scenario provides a controlled static and planar environment, with no stairs, lifts or another vertical navigation possibility. This way, the case study focuses specifically on the tasks of moving (forward and backward) and rotating (to the left and the right) the virtual camera horizontally.

In this case, we have a combined set of tasks that instantly affect one the other. The choices of the user about moving and rotating can happen separately, but can also occur at the same time, being associated with a single desired motion. Therefore, instead of experimenting each gesture individually, here we explored the option of prototyping at once a particular set of gestures. This represents no change in the definition of the toolset; the adaptation requires only that the designers select the gesture sets to be prototyped.

4.1.2 Gestures

For this experiment, we performed the conception of the gestures along with experts (interaction designers and HCI researchers). We generated several concepts through brainstorming. Additionally, a gesture set (Tapping-in-Place (NILSSON et al., 2013)) from the state of the art of

[1] Detailed on the Section 3.2.2 of Chapter 3

navigation techniques was added. We ranked the ideas using a comparison matrix and giving scores (from 1 to 5, being higher better) regarding two criteria: intuitiveness and required effort.

The required effort was presented due to a concern that the gestures should provide the user a way of turning the virtual camera direction without losing the view of the screen. Moreover, the user must be able to move long distances in the virtual environment without trespassing real world boundaries and without becoming fatigued. The use of gestures may be tiresome if the gesture is used long enough. Furthermore, besides the physical tiredness, there is also the cognitive effort, i.e. the gesture may be hard to associate with the command, and so it requires the users to keep trying to remember what they should do instead of just doing it. We associated this last concern to the intuitiveness criteria.

Later, we selected six different navigation gesture sets (for moving and rotating), being each based on a specific metaphor. Figure 4.1 illustrates the six selected metaphors and gestures.

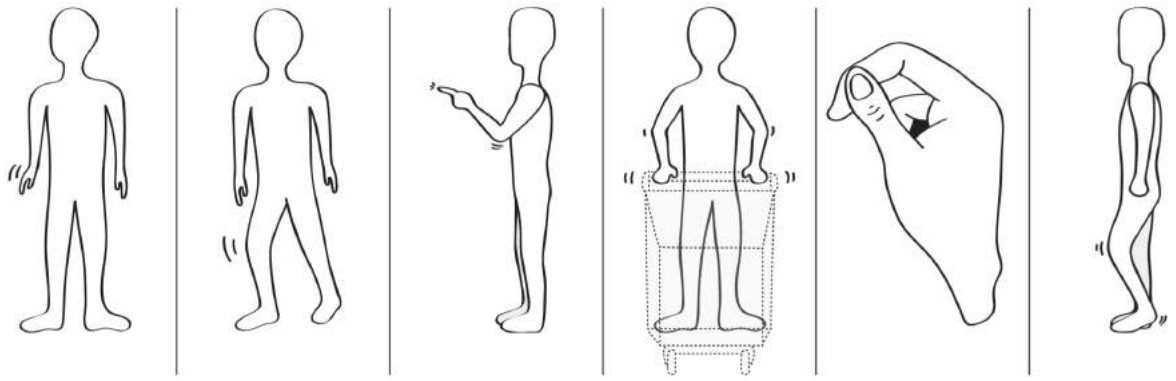

Figure 4.1: Examples of the prototyped gestures: Hand as Joystick, Equilibrium, Taxi Driver, Shopping Cart, Thumb-based, Tapping-in-Place, respectively, from left to right.

1. *Hand as Joystick.* The first metaphor was inspired by the use of an ordinary Joystick where the user could navigate forward, backward and rotate to the left and right using a single digital directional part of the joystick. In the metaphor, with one of the hands, the user signaled ahead to move forward and indicated sideways to rotate. For continuous interaction, the user should maintain the gesture for the duration of the desired action.

2. *Equilibrium.* The second set of gestures benefits from users gravity center for indicating the direction, forward or backward, along with the rotation movement of shoulders to indicate the rotation side. Like the previous one, the user should maintain the gestures for keep moving.

3. *Taxi Driver.* Mimicking real-world gestures, the third metaphor was based on the way instructions are given to a Taxi Driver, which also gives the metaphor name. By pointing forward just once, the user moves ahead; by showing an open hand, the

movement stops. Pointing, while maintaining to the right or left, rotates the virtual camera. This gesture employs a trigger-based interaction, minimizing the effort on the continued walking forward activity.

4. *Shopping Cart.* Still referring metaphorically to real-world gestures, the next set imitates the action of guiding a shopping cart. By slightly stretching arms forward, the user pulls the invisible cart and then walks forward; to keep walking and turning to one of the sides, she must retract one arm a bit, and by retracting the entire arm until touching the body, the user just rotates without walking forward.

5. *Thumb-based.* The last set designed by us uses only the thumb and forefinger of one hand. With the thumb, the user touches one of the phalanges of the index finger to navigate. The middle phalange indicates walking forward, the right one would turn to the right side, and the left one would turn to the left side. This gesture was chosen due to its minimization of the required effort.

6. *Tapping-in-Place.* We also selected a set of gestures from state of the art about in-air gestures navigation, which is called "Tapping-in-Place" (NILSSON et al., 2013). The movement of going forward is generated by tapping each heel against the ground. The user lifts alternatively one heel over the floor without breaking contact with the toes and continues with the gesture for keep walking. For rotating to one of the sides, she simply needs to turn the head right or left.

4.1.3 Setup

As a previous step, the setup of the experiment is set to ensure that the Wizards will not see the feedback of their actions as shown in Figure 4.2. This way, the Wizards act more precisely like a recognition system, reducing the bias occurred due to the Wizard observation of the display itself, and the virtual environment and the target path.

Additionally, two Wizards are assigned, one to interpret the rotation gestures and other to translate the moving gestures. By dividing the tasks, the cognitive load on each Wizard is diminished preventing hesitation (higher latency) and miss-clicks (false-positives). This decision of using two Wizards was a result of prior frustrated attempts to make a single Wizard handle all tasks. Therefore, we propose for designers to consider as a good policy to review the number of Tasks per Wizard.

It is important to notice that using our Task classification the granularity of the tasks definitions may be translated into more than one command for the final application. For instance, both Tasks resulted in a pair of commands, moving forward and backward and rotating to the left and the right. Therefore, each Wizard handled a couple of commands, having one keyboard key for each.

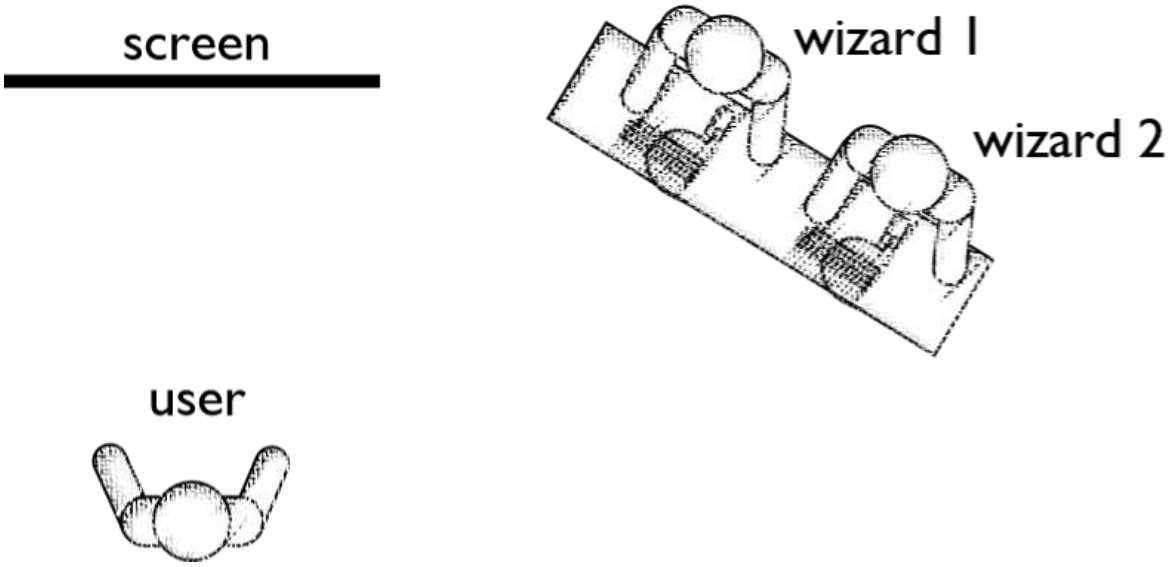

Figure 4.2: Wizard of Oz experiment setup.

4.1.4 Training

Later, we conducted an iterative process of Wizards training and evaluation. On this previous assessment, our concerns were directed not to the gesture intuitiveness or required effort, but to performance goals, such as the response time, false-positives and false-negatives errors. We compared the Wizards performance to a gesture recognition solution implemented using the skeleton data tracked from a Microsoft Kinect device. Therefore, we created a generic test set of gestures to be straightforward for the tracking and implemented recognition thrive in all cases.

Once we set the tracking and recognition solution, we called the Wizards and began the test and performance measurement. While a tester played the role of the user and performed the gestures, we set the implemented system to run its recognition on the background. At the same time, the Wizards were actuating on the foreground, performing their role in the same way they would as if a real user were executing gestural commands.

We stored all the recognition results from the interaction, from both Wizards and the Kinect-based solution. We also used a camera to videotape the test. Later we used this recording to review and discard failures from the Microsoft Kinect implemented recognition. Once we rejected these cases, we compared the remaining ones considering the implemented system as ground truth and its timestamps as a guideline of latency. We continuously iterated through this procedure of test and evaluation until the Wizards achieved the following requirements: the tester considers the interaction successful, the number of false positives and false negatives are rare, and the average latency is lower than 100 milliseconds[2].

[2]This latency restriction follows Jakob Nielsen guidelines presented in his book called Usability Engineering (JAKOB, 1993), and summarized in his online article at www.nngroup.com/articles/response-times-3-important-limits. According to Jacob "0.1 second is about the limit for having the user feel that the system is reacting instantaneously".

After a few iterations, the Wizards achieved the performance goals and were ready for the experiment. At the end of the Wizards training the mean delay between recognizing the gesture start and pressing the corresponding button was 0.086 seconds if compared with the same gesture being tracked by a Kinect device. The average releasing delay was about 0.101 seconds when releasing the button, surpassing the stipulated threshold by 1 millisecond; we considered it to be acceptable. The total average latency was then 93 milliseconds. The rate of false-positives was nearly zero (0.007), and zero false negatives occurred. Overall, the Wizard technique fulfilled the research needs and objectives.

4.1.5 Structure

Once we validated the low-fidelity prototyping using the Wizard of Oz technique, we structured the user experiment to test our setup and gesture sets with users. The test structure is defined as follows. First, we conduct an elicitation session, in which the participants are encouraged to idealize and test three natural and effortless sets of gestures for navigation in the virtual environment. The target tasks were explained as walk forward and backward, and rotate left and right. We gave users the chance to develop their gesture metaphors to create an opportunity for new ideas. At the same time, we give the Wizards the challenge to learn a new gesture rapidly and recognize it. This way, on the next step the Wizards were trained by the users to recognize their suggestions of interaction. We also explained to users that they should follow a path (composed of blue and red balls).

In the second moment, we asked the participants to use three sets (randomly sorted) of our pre-defined gestures. After this step, a second elicitation phase took place in which users could suggest another set of gestures based on their experience. The goal of this task was to allow the participants to create final gestures after they used the set of movements previously generated. The premise is that the participants could have their cognitive repertoire augmented after the second session, increasing or redirecting their creativity.

We collected quantitative data from a Likert questionnaire performed after each task. Then users score statements about some usability aspects. The qualitative questions regarding user experience were gathered using a semi-structured interview. Through observation analysis, we could also define which body parts were involved in each type of movement performed by the participants. Additionally, the followed path and the time consumed by each trial were analyzed, to compare the proposed gestures as well as understand if there was a significant difference on Wizards performance regarding the user-defined gestures in comparison with the pre-defined gestures.

4.1.6 Experiment and results

Ten participants (nine males and one female) aged from 21 to 40, took part in this experiment. Two of them were left-handed. None had motor disabilities. Only one subject had

no experience with Virtual Reality (VR), eight had some, and one was high experienced. Five users had a lot of experience with 3D games, while five had some. Four of them were very familiar with gesture-controlled games; the other six were not that familiar but had used it at least once. The mean experiment time per subject was one hour, including going through an introduction to the test, pre-questionnaire, warm up, instructions and the six trials to complete the path using gestures while attending to a questionnaire after each trial.

As the main result, we validated the use of the Wizard of Oz technique for real-time gesture interaction prototyping, especially regarding the emergence of natural gestures defined by users and the capture of sustained (continuous *Flow*) and discrete body commands. For each gesture execution, we introduced two questions on our questionnaire to assess both the response time of the Wizards and the precision of their recognition. Users could answer using a 5-point Likert scale. The average score for response time and precision for user-defined gestures were 4.4 and 4.42 respectively, while the same measures for the pre-defined gestures were 4.1 and 4.2. Both cases highlight the approval of users regarding the recognition method. Interestingly, users found that Wizards performed better while recognizing their gestures. This finding may indicate that user gestures were simpler to be identified. There is also a possible bias, on which users do prefer to rate their gestures better than the predefined ones. We also added an open question to let users answer if the presence of the Wizards somehow disturbed their performance. All users answered "no" to this question meaning the potential social discomfort that could be caused by the Wizards was not an issue.

Figure 4.3 shows the paths the users walked through using each of the predetermined movements; each different lines is used for each user. It also shows the blue and red balls, which were used to guide the navigation through the scenario. We observed that all paths are well concentrated nearby the target balls, demonstrating users controlled well their paths and therefore the gesture recognition was successful.

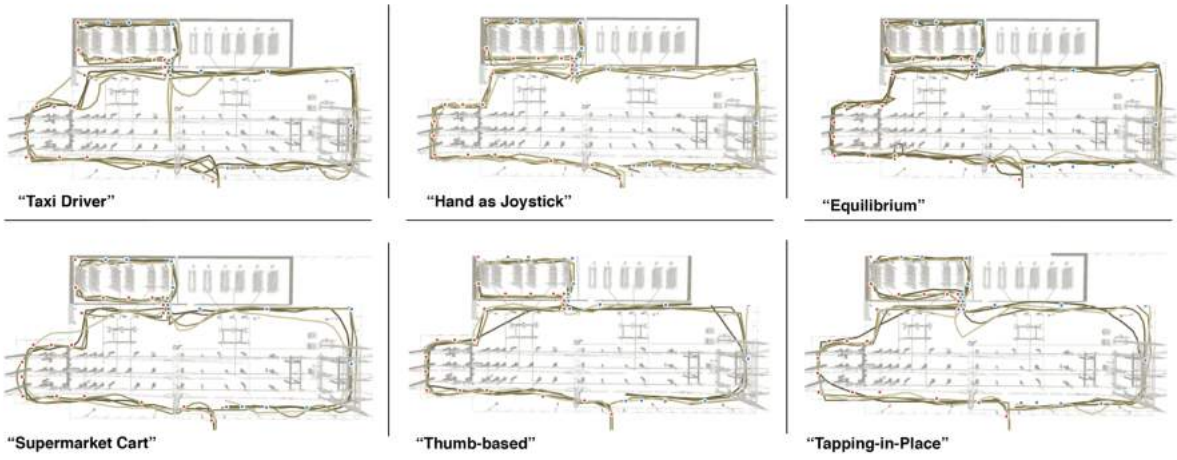

Figure 4.3: Paths performed by all tested users while trying the six different metaphors.

4.1.7 Selection

To select the options of gesture sets to be built as high-fidelity prototypes designers can use different techniques. On our toolset definition, we propose that designers test the low-fidelity prototypes with users and assess the usability through the utilization of the SUS questionnaire, among other techniques. On this first experiment with the Wizard of Oz technique, we had yet no guidelines for this step. In fact, this first experience was a learning step towards the toolset definition. In this experiment, we created a questionnaire with specific questions assessing our concerns of natural and effortless interaction. We also evaluated the gesture sets regarding the user's performance (paths and required time) while using them. These are further assessment and evaluation options for the ones proposed on the toolset definition.

Regarding the time taken to complete the given circuit, Hand as Joystick had the worst performance with an average of 290 seconds, followed by Equilibrium and the Taxi Driver that had 277 and 260 seconds, respectively. On the other side, Tapping-in-Place provided the fastest way to navigate having an average of 192 seconds, followed closely by Shopping Cart and Thumb-based that took 199 and 211 seconds, respectively, to complete the same task.

Another observation regarding the paths shown in Figure 4.3 is that in gestures that allowed multiple inputs, i.e., walk forward and rotate, (except the Equilibrium) the users tend to make curves while still going forward. Although that seems to be bad while looking for the path traced as it sometimes does not pass near the balls, it makes a more continuous movement and thus, the path can be completed earlier since the user does not have to stop, rotate and then walk again to make a curve.

The experienced degree of naturalness was assessed by two questionnaire items requiring the participants to rate their level of agreement (from 1 to 5, being higher better). The two items measured the cognitive effort and the walking sensation. It was noticed that, except for the Shopping Cart gesture, all the others were considered to have little cognitive effort and two of them, Equilibrium and Tapping-in-Place, were agreed to provide a good walking sensation, possibly because of the use of the legs or foot for moving forward. The Thumb-based gesture was considered the most comfortable, as shown in Figure 4.4.

The results obtained from the questionnaire item related to the physical effort are shown in Figure 4.4. Significant differences were found regarding the techniques that used more than one body part: Hand as Joystick, Equilibrium, Shopping Cart, and Tapping-in-Place. These were considered more strenuous and had a mean of 2.9 on a range from 0 to 5. In qualitative data, users said the Thumb-based was the less tiring gesture, but Taxi Driver was most intuitive and let them free to do other movements since it works as a trigger in the act of moving forward, and thus, being more comfortable when navigating long distances.

After users had performed all gestures, we conducted a semi-structured interview. When we asked them which gesture was more appealing to be executed, the most frequently cited were Taxi Driver and Thumb-based mainly due to the lesser effort to perform them; users considered

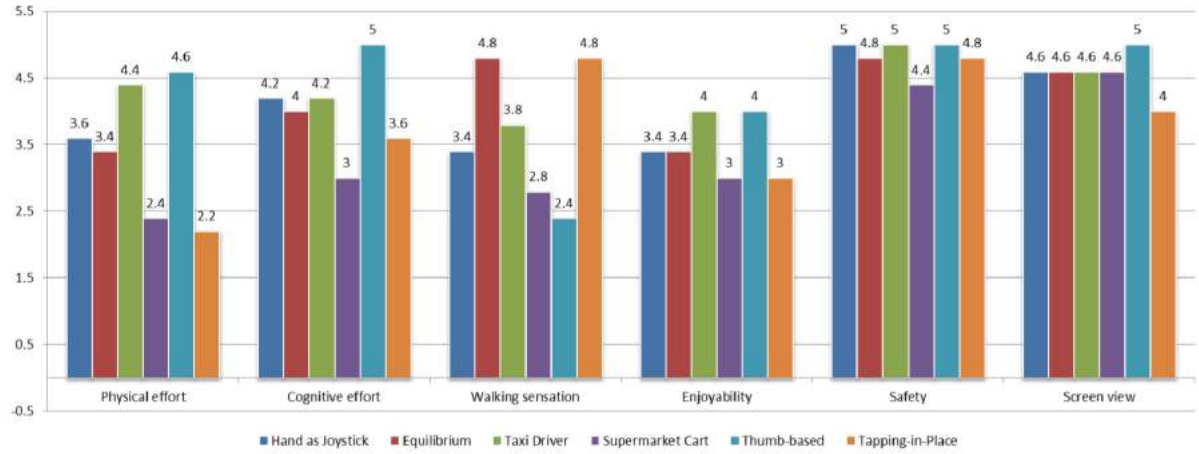

Figure 4.4: Comparison between all tested metaphors regarding six parameters evaluated.

these gesture sets to be more natural, as well.

4.2 High-fidelity prototyping: Prepose

After the steps of low-fidelity prototyping and testing, the following step is to produce high-fidelity prototypes for the selected gestural interactions. These high-fidelity prototypes must perform the automatic recognition of user performed gestures by using available sensors and algorithms. Recognizers can make use of several different input data, including sound (GUPTA et al., 2012) and wireless signals (PU et al., 2013). There are also multimodal approaches capable of combining different input sources such as depth data, color images, skeleton joints, and audio, as shown by Pitsikalis and colleagues (PITSIKALIS et al., 2015).

Particularly, depth and color data are more sensitive to orientation and scale changes once they directly depend on the positioning relationship between the sensor and the user, i.e. translation and rotation variance. On the other hand, the skeleton data is by definition invariant to translation and rotation. Although on some cases (e.g. Microsoft Kinect (MICROSOFT, 2013) and Leap Motion (MOTION, 2013) tracking solutions) the computation of the skeleton uses the depth data, the tracking procedure is built to handle these issues, being invariant to translations and rotations. In a general sense, tracking solutions are employed to compute the skeleton data.

Tracking can be described as the task of defining the localization of the target at a given time. It can be used for several purposes, for example, for security by using gestures for authentication (AUMI; KRATZ, 2014), or on crowd analysis to understand people behavior by post processing captured video files to trace a path of subjects on a scene (GE; COLLINS; RUBACK, 2012). While focusing on interactive solutions, to empower the user with in-air gestures controls, the tracking must be performed in real-time. The output of the tracking can be any notion of the user location, ranging from a 2D bounding box of a tracked face to the 3D position and orientation of several of the user's joints. Tracking joints in the 3D space is particularly useful for gestural interfaces because it broadens the range of distinguishable

gestures.

Sensors like the Microsoft Kinect for Xbox One and the Leap Motion are capable of capturing depth data. The depth information is further processed on their Support Development Kits (SDKs) to extract the skeleton joints of the user body and hand, respectively. These are low cost, commercial sensors capable of performing tracking in real time with enough precision for general purpose applications (WANG et al., 2015; WEICHERT et al., 2013). These currently available tracking solutions demonstrate it is possible to acquire precise and real-time information of the user skeleton, opening space for advancing to the recognition step.

There are benefits of entirely separating the tracking and recognition problems. In our work, we assume that a skeleton will be provided as input. One of the advantages related to this choice is that by basing the recognition method on the skeleton, the semantics of the gesture is more reachable; the skeleton representation reveals all relative and absolute positioning of each tracked body part. For instance, this allows a direct textual transcription of the gesture to be used as code for recognition, which is one of the pillars of our recognition tool (Prepose). Moreover, the skeleton data is a good representation of the tracking output. On this representation, the required amount of data is small enough to be transmitted in real time through networks or even to store entire datasets on personal computers. The skeleton can also be treated as a common structure for both, body and hand description, or even face if landmarks such as eyes, eyebrows and mouth corners are considered as joints; thus, the same recognizer structure can be extended to support other skeletons. Also, the developed method can be further extended to support new devices and trackers, considering the only assumption that they output a skeleton.

In this Chapter, we explore the gesture recognition challenge focused on the development of solutions for in-air gestural recognition, and consider as input a skeleton of 3D joints. Additionally, we aim to enable developers and non-developers with techniques and tools to build gestural interfaces, targeting fast high-fidelity prototyping alternatives. Concerns are not only directed to real-time and accurate recognition but also to the required time effort and expertise to build the recognizer, the easiness of its use, and its reconfigurability (FAUGEROUX et al., 2014; IBAÑEZ et al., 2014; ZHAO et al., 2014). Another issue while creating recognizers is to unintentionally allow two or more gestures to be executed at once, which can be phrased as a *gesture conflict*. In general, *conflicts* are only detected at runtime, while experimenting the developed gestures. Detecting *conflicts*, before experimentation, would save development time. Our specific goals for the recognition solution are:

- Recognition: develop gesture recognizers using a joint skeleton as input.
- Invariance: be robust to orientations, positions, and scales of the skeleton [3].

[3] Although invariant tracking solutions aim to retrieve user joints regardless of user position and orientation, the extracted data is usually presented as 3D positions using the sensor center and orientation as the definition of the coordinate system for these points. This way, these tracking solutions are committed to providing reliable skeleton data, but additional transformations must be applied to make the recognition also invariant to rotation, translation, and scale of the skeleton data.

- Continuous output: provide continuous real-time *Flow*[4] on all interactions.
- Partial gestures: recognize gestures that require only part of the skeleton.
- Prototyping: develop tools to allow fast high-fidelity prototyping.
- Easiness of use: conceive and develop recognizers that are easy to understand and use by non-developers.

4.2.1 Scope and research directions

In-air gestures recognition can be defined as the task to interpret the user posture and motion. The gesture can be executed using the whole body or any part of it. Regarding to its *Form*[5], on dynamic gestures, in addition to the shape of the gesture (analyzed, for example, through the performed trajectory), its dynamics can also be measured. The dynamics concerns about the behavior of the user over time, analyzing variables such as velocity, acceleration, and rhythm. Our work aims to recognize both static and dynamic gestures, focused mainly on the shape aspect; the dynamic aspect is later discussed as one of the future work directions for this thesis.

Regarding the input skeleton data and supported sensors, the first version of Prepose provides support to the Microsoft Kinect V2 (2nd version of the sensor launched along with the console Microsoft Xbox One). We chose this sensor considering the following factors as detailed below:

- It is one of the most (arguably the most) popular depth sensor available in the market.
- It is an improved version of its 1st version (Microsoft Kinect V1), which was launched back in 2010 being the first low-cost depth sensor available.
- The V2 SDK 2.0 showed to be similar to the V1 SDK 1.8, which facilitates the applicability of the new sensor on the same applications handled by its 1st version.
- Its SDK provides real-time tracking of 25 body parts, including head, hips, elbows and feet. By tracking the user body, it is possible to relate the positioning of arms and hands to other joints, recognizing hand gestures that would not be supported by other devices such as the Leap Motion, that tracks hands only.
- Additionally the V2 sensor and SDK provided the tracking of new joints for additional body parts (e.g. left and right thumbs).
- Although it does not provide full tracking of user hands and fingers, a research publication from Microsoft Research showed impressive results of hand tracking

[4]Detailed on the Section 3.2.2 of Chapter 3.

[5]Detailed on Section 3.2.2 on the Chapter 3. There we classify the gestures as either static or dynamic. Static gestures are those described by a single posture (or pose), and dynamic gestures, by its turn, require motion and can be defined as the transition between postures.

using the sensor (SHOTTON et al., 2013), which may indicate future support of its SDK for retrieving full hand skeletons in addition to the body skeleton.

Therefore, the Microsoft Kinect V2 was our first adopted sensing device. Once we are proposing the use of the retrieved skeleton data as input, extending Prepose by adding new sensors (which also retrieve skeleton data) should require low implementation cost. Additionally, the choice of the used device also implies the selection of the used platform. The Kinect V2 is fully supported only on Windows OS, versions 8 or higher.

Regarding the gesture recognition, we consider two approaches as complementary: rule-based and machine learning methods. Once the intent is to recognize previously defined gestures, supervised machine learning methods are applicable. By giving as previous input executions of the target gesture, classifiers can be trained to recognize further performances of the same gesture. Rule-based approaches, on the counterpart, can potentially create recognizers without any input gesture, using previous knowledge of human motion by domain experts (e.g. physiotherapists for therapy gestures) and non-experts (e.g. programmers). Rule-based approaches do not require training, can be readily reviewed and adjusted among other advantages while machine learning methods are well known for good accuracy.

We explore both approaches for gesture recognition from the viewpoint of a fast high-fidelity prototyping solution. The advantages of each approach are better discussed in the further subsections. For each case, a literature review is conducted to identify which of the goals are supported by current related work and which are remaining gaps. The study is followed by the conceptualization and implementation of the recognition methods, which are later analyzed.

4.2.2 Rule-based recognition

Rule-based recognition defines the challenge of recognizing gestures by providing rules about how the gestures should be performed. This approach presents a set of advantages and disadvantages as we discuss on the following sections. We also introduce a rule-based recognizer called Prepose that is later detailed.

4.2.2.1 Hard coded gestures

In general, developers of NUI applications pursue a few different approaches for creating new gesture recognizers. The simpler approach is to write code that explicitly encodes the gesture movements using the received skeleton data. The code is written in a general-purpose programming language such as C++ or C#. It checks properties of the user joints position and then sets a flag if the user moves in a way to perform the gesture. For example, the Code 4.1 shows a simple punch gesture.

Code 4.1: C# hard-coded punch gesture.

```
// Punch Gesture
```

```
if ( vHandPos.z-vShoulderPos.z>fThreshold1 &&
    fVelocityOfHand > fThreshold2 ||
    fVelocityOfElbow > fThreshold3 &&
    DotProduct(vUpperArm, vLowerArm) > fThreshold4)
{
    bDetect = TRUE;
}
```

The code above checks four Boolean expressions which evaluate: if the user's hand is far enough from the shoulder; if the hand is moving sufficiently fast; if the elbow is also moving fast enough; and if the angle between the upper and lower arm is greater than a threshold. If all these checks pass, the code signals that a punching gesture has been detected. Manually written poses require no special tools, data collection, nor training, which makes them easy to start with, for example on the *Prototype* phase of the interface. Unfortunately, they also have significant drawbacks:

- The code is hard to understand because it typically reasons about user movements at a low level (e.g. "DotProduct(vUpperArm, vLowerArm) > fThreshold4").
- Building these gestures requires a trained programmer and maintaining code requires manually tweaking threshold values, which may not work well for a wider range of users.
- It is difficult to analyze automatically this code because it is written in a general-purpose programming language, so *invalid* gestures, *conflicts* or *unsafe* gestures must be detected at runtime through experimentation. *Invalid*, *conflicted* or *unsafe* gestures are detailed further.
- Finally, the manually coded gesture approach requires the application to have access to sensor data for recognizing gestures. This raises one of our privacy concerns: a malicious developer may directly embed some code to capture video stream or skeleton data.

Invalid gestures can also be pointed as impossible or unachievable gestures. While writing gestures, defects can be inserted within the gesture definition, requiring the user to achieve an *unsatisfiable* state. The freedom developers have while coding allows composed expressions such as $X > Y$, $Y > Z$ and $Z > X$. This expression is *unsatisfiable*, since there is no assignment of values to X, Y and Z that can make all three Boolean constraints to be true. While dealing with a few expressions, these cases may be easily detected by human analysis. However, once the set of expressions grows, it may be not practical to revise all the written gestures. Therefore, developers can write *invalid* gestures which will probably be diagnosed only during runtime,

by experimenting and testing the application. This late diagnostic increases the costs in time to produce and test new gestures.

Conflicts represent the cases when two definitions of gestures can be accomplished at the same time. The same freedom that allows developers to build *invalid* gestures also allows writing two gestures coded with different expressions but being both *satisfiable* by the same input skeleton (or skeleton sequences). For example, the expressions $X > Z$ representing the first gesture and $Y > Z$ representing the second can be *satisfied* by a sign gesture in which both X and Y are greater than Z. In this case, the *conflict* is also usually detected at runtime, while testing the gestural interface. There are *conflict* cases that can be chosen to ignore because designers defined that the actions of both gesture definitions can be triggered simultaneously by the same movement. Nevertheless, there are also cases in which *conflicts* are not desired (e.g. a game gesture mistakenly activating the home function on Xbox One), and finding these *conflicts* sooner can reduce the time of the development cycle.

Unsafe gestures represent gesture definitions that may require the user to perform uncomfortable, harmful or even impossible gestures. For example, if a tutorial or feedback is provided based on such gesture definition, it can lead the user to reach a particular pose (e.g. "bend yourself 90 degrees towards your back") which may not be suitable for all users. There may be required gestures that can be impossible, not by definition but related to limitations of human biomechanics. Developers can write such codes aiming to ask users to excel themselves, but also by accident, writing an unfortunate combination of expressions which when added together result to be harmful or impossible. These *safety* issues are usually detected at runtime.

4.2.2.2 Domain-Specific Language (DSL) for gesture descriptions

As an alternative, there are also rule-based gesture recognizers, which can be defined as abstractions and automation of manually written codes, thus allowing faster and easier generation of the recognizers. Rule-based approaches create a structure for gesture instantiation, so the written code is narrowed and more controlled, depending on the underlying purpose.

For example, Da Gama and colleagues defined body gestures for physiotherapy using as primitives three elements: the body part (limb, or bone); the plane (frontal, sagittal and horizontal); and the target angle (GAMA et al., 2012). In this case, the *rules* are specified to attend the therapy needs, allowing configuration by non-programmers, especially the therapist. The same *rules* are also limited, being not suitable for more complex out of the plane gestures. Zhao and colleagues also introduced a rule-based recognizer (ZHAO et al., 2014). It exposes *rules* as a eXtensible Markup Language (XML) considering joints positions and the corresponding bones orientations. Unfortunately, the *hip abduction* gesture is demonstrated in 48 lines of XML code that are not easily understood by non-experts. The work quotes "For sophisticated gestures, it may be difficult to define the *rules* precisely. Fortunately, rehabilitation exercises usually

involve simple body movements that are easy to define", evidencing that the language is better suited to a specific domain.

Hachaj and Ogiela presented a Domain-Specific Language (DSL) for gesture description (HACHAJ; OGIELA, 2014). Rules are written using the language, and are defined by cause (specific body conditions) and effect (resulting Boolean value). The *rules* can be either logical or numerical, allowing the developer to set thresholds for specific conditions. Rules can be connected by using the effect of previous *rules* as input. It is also supported the use *rules* that relate prior and current poses, and in addition it can specify a required waiting time for poses transition in order to consider the *rule* as valid. In fact, the freedom while writing the *rules* with the proposed DSL is similar to coding with general purpose languages, resulting in low-level written gestures as shown in Figure 4.5. As a result, the final text looks much more like code than natural language. At last, invariance to sensor position, orientation, and the skeleton scale is dealt within the *rule* description, which makes the developer code to be error prone.

A.
```
RULE RightElbow.x[0] > Torso.x[0] & RightHand.x[0] > Torso.x[0]
& RightHand.y[0] > RightElbow.y[0] & abs(RightHand.x[0] - RightElbow.x[0]) < 50
& abs(RightShoulder.y[0] - RightElbow.y[0]) < 50 THEN RightHandPsi
RULE LeftElbow.x[0] < Torso.x[0] & LeftHand.x[0] < Torso.x[0]
& LeftHand.y[0] > LeftElbow.y[0] & abs(LeftHand.x[0] - LeftElbow.x[0]) < 50
& abs(LeftShoulder.y[0] - LeftElbow.y[0]) < 50 THEN LeftHandPsi
RULE RightHandPsi & LeftHandPsi THEN Psi
```

B.
```
RULE sqrt((torso.x[0] - torso.x[1])^2 + (torso.y[0] - torso.y[1])^2 + (torso.z[0] -
torso.z[1])^2) > 10 THEN Moving
RULE Distance(torso.xyz[0], torso.xyz[1]) > 10 THEN Moving
```

C.
```
RULE distance(RightHand.xyz[0], LeftHand.xyz[0]) < 100 THEN HandsTogether
RULE distance(RightHand.xyz[0], LeftHand.xyz[0]) >= 100 THEN HandsSeparate
RULE sequenceexists("[HandsSeparate,0.5][HandsTogether,0.5][HandsSeparate,0.5]") THEN Clapping
```

Figure 4.5: Rule-based DSL presented by Hachaj and Ogiela (HACHAJ; OGIELA, 2014). On part A, two *rules* are created to be further used on a third *rule*. On part B, low-level *rules* are created considering the current '[0]' and the past '[1]' versions of a specific joint. On part C, the time requisite is specified (0.5) as required time for transition of states on dynamic gestures.

Bedregal and colleagues presented a rule-based recognizer and a corresponding DSL for hand gestures, using as input joints and separations between fingers tracked by a data glove (BEDREGAL; COSTA; DIMURO, 2006). The DSL terms for finger joints are STRAIGHT, CURVED or BENT referring to the angles for each joint, and the terms for finger separations are CROSSED, CLOSED, SEMI-OPEN and OPEN also referring to lateral angles. Despite these angles descriptions being represented at a high-level discrete class, by using Fuzzy logic each joint or separation can partially belong to more than one class. A short notation is attributed to each finger joint (F1-5), separation (S12-45) and the corresponding state terms (St, Cv, and Bt for joints and Cr, Cd, SOp and Op for separations), as shown in Code 4.2. Gestures are defined by a sequence of hand configurations (hand postures/poses), and each state changing is considered a new *pose*. All joints and separations are always considered as relevant, which probably will result in false-negatives for more loose gestures that do not impose restrictions over all parts of the input skeleton. The condensation of proposed DSL and the required full description of each

pose also turn the resulting code to be less readable and harder to be understood by non-experts.

Code 4.2: Code used by Bedregal and colleagues to describe a hand *pose* (BEDREGAL; COSTA; DIMURO, 2006).

```
If F1 is BtBtSt and S12 is Cd and F2 is BtBtSt and S23 is Cd and
F3 is BtBtSt and S34 is Cd and F4 is StStSt and S45 is Cd and
F5 is StStSt
Then HC is [G]
```

4.2.2.3 Prepose language and runtime

Considering the presented context of background works, we introduce Prepose as a language and a runtime solution for authoring gesture-based applications. Prepose concept is created as a complementary approach for enhancing the gesture writing to speed up the early *Prototype* phase of gestural interfaces. Prepose is an open source tool, and it is currently stored and available in a GitHub repository at www.github.com/lsfcin/prepose.

The Prepose DSL is a Natural Programming Language. Therefore, it is based on natural language descriptions of gestures. For illustration, Code 4.3 supported by our system is shown below.

Code 4.3: Sample gesture using Prepose language.

```
GESTURE crossover-left-arm-stretch:
POSE relax-arms:
point your left arm down,
point your right arm down.
POSE stretch:
rotate your left arm 90 degrees counter clockwise on the frontal
   plane,
touch your left elbow with your right hand.
EXECUTION:
relax-arms,
slowly stretch and hold for 30 seconds.
```

As implied by the code, Prepose supports partial gestures, allowing the gesture writer to impose constraints on specific body parts while the user can freely move the remaining ones. Moreover, Prepose exposes the different viewpoint of dealing with the gesture recognition challenge as a constraint-solving problem. Furthermore, it highlights the advantages of this path such as the identification of *invalid*, *conflicted* and *unsafe* gestures. According to our knowledge, other works regarding rule-based recognizers have not tackled the *validity* and *conflicts* issues yet. Neither the *safety* of gestures has been addressed.

At the heart of Prepose is the idea of compiling gesture descriptions to formulas for a

Satisfiable Modulo Theories (SMT) solver; we use the Z3 solver (DE MOURA; BJØRNER, 2008). These formulas capture the semantics of the gestures, enabling precise analyses that boil down to *satisfiability* queries to the SMT solver. Without using the solver, it would not be possible to search and find *invalid*, *conflicted* and *unsafe* gestures automatically. The Prepose language has been designed to be both expressive enough to support standard gestures yet restrictive enough to ensure that fundamental properties remain decidable. Prepose focuses on the ability to analyze gesture-based programs statically:

- Ensures that gestures are internally valid, i.e. do not require the user to both keep his arms up and down.
- Tests whether a gesture conflicts with a reserved system-wide gesture, such as the Kinect attention gesture.
- Finds potential conflicts within a set of gestures, such as two gestures that would both be recognized from the same user movements.
- Validates that gestures have a basic measure of safety, i.e. they do not require the user to overextend herself physically in ways that may be dangerous.

4.2.2.4 System overview

Prepose defines a DSL for writing gesture recognizers. The definition of the DSL is based on gestural descriptions, from general examples such as the ones shown by Fothergill and colleagues (FOTHERGILL et al., 2012) to technical examples such as the exercises provided by the American Academy of Orthopedic Surgeons (AAOS) available at `http://orthoinfo.aaos.org/topic.cfm?topic=A00672`. A set of textual descriptions was gathered, analyzed and converted to an underlying structure of natural language descriptors.

Skeleton The skeleton representation is the first concern while defining our system. The challenges of using the raw skeleton data include variance to position and orientation of the user (from the sensor viewpoint), the scale of the user body (e.g. size difference between children and adults) and different scales within the same skeleton (e.g. variation of the relation between arm and leg bones). Bigdelou and colleagues used distances and displacements between joints and the spine as skeleton representations. As an alternative hierarchical representation, they also explored the displacement between each joint and its parent joint. At last, they also investigated the normalization of the input data (BIGDELOU et al., 2012). Vieira and colleagues stored the distances between all joints are stored in matrices as a representation. Faugeroux and colleagues used spherical angles focusing on the direction from joint to joint (FAUGEROUX et al., 2014).

To achieve invariance to the position, and orientation of the provided skeleton we perform prior transformations on this data. We store the skeleton data as a set of vectors representing the

directions of each body part. This way, for example, the hand vector is computed by subtracting the elbow joint 3D position from the hand joint. All the vectors are then transformed into a new coordinate system. The new coordinate system is based on three normalized and orthogonal vectors:

1. The first vector of this new system is defined by the user's spine; this vector specifies the up and down directions.

2. The second vector is based on the user hips (defined by $righthip - lefthip$), and represents the right and left directions.

3. The third vector is calculated by the cross product between the last two vectors and describes the front and back directions.

The origin of the new coordinate system is the *spine base* point. Therefore, this point does not become transformed into a vector. To convert all body vectors to the new coordinate system, we apply the change of basis transformation on all vectors except for the *spine base*. In this representation, each body part individually represents only its direction regarding the *spine-hips* coordinate system. The calculation of the position of the body part is performed by summing up the vectors that precede it on the hierarchy until the spine base point is reached. For example, in order to find the *right ankle* position it is required to sum up the vectors of the *right ankle*, *right knee*, and *right hip*. There is no need to sum the spine base point because it is the origin of the new coordinate system and therefore it represents the $(0,0,0)$ point. Once all vectors are transformed to this new system, the position and orientation of the user are no longer relevant for further recognition steps.

Actions *Actions* are the atomic definition of Prepose gestures. *Actions* must be represented by a verb that anticipates the description of what the user should do to fulfill the desired interaction. The language breaks down the descriptions of the following *actions*:

- *Point*, asks the user to point a body part to a specific direction.
- *Rotate*, rotate a part to a direction by a minimum or specific amount of degrees.
- *Touch*, the user should use one of his hands to touch another body part.
- *Put*, asks the user to place a part in a comparative position to another part (e.g. above).
- *Align*, asks the user to align two body parts.

Each *action* can take a set of corresponding parameters, as specific joints (e.g. *left wrist, left elbow*), compound body parts (*left arm* or *arms*), directions (e.g. *up, down, left, right, front, back*), or numeric parameters (*90 degrees*). We created this initial set of *actions* to handle different possibilities for descriptions. For instance, the combination of *Point* and *Rotate actions* can precisely describe a body posture. If the description asks the user to "point her left arm to the left and rotate 30 degrees up" a precise posture is defined. On the other hand, the *Put*, *Touch* and

Align actions can allow the user to accomplish a gesture by several different ways. For example, with the intent to recognize the clapping gesture if a description asks the user to “touch your right hand with your left hand”, it allows the user to clap hands below her chest or above her head. Besides, all *actions* define gestures based solely on the user body, skipping definitions such as “put your hand 2 meters above the ground”. This way gestures described using Prepose are invariant to scale, which means that a gesture that can be performed by an adult can also be accomplished by a child.

Grammar We created a Backus-Naur Form (BNF) context-free grammar to define Prepose DSL (Figure 4.6). The grammar describes how Prepose applications can be composed out of gestures; gestures formed out of *poses* and *execution* blocks; *execution* blocks made out of *execution steps*. The implemented *actions* (translated as transformations and restrictions) are also detailed at (FIGUEIREDO et al., 2014). The grammar is extensible: if one wishes to support other kinds of *actions*, one needs to extend the Prepose grammar, regenerate the parser, and provide runtime support for the added *action*. Note also that the Prepose grammar lends itself naturally to the creation of developer tools such as context-sensitive auto-complete in an Integrated Development Environment (IDE) or text editor.

Figure 4.6: BNF free grammar that defines Prepose DSL.

In its current version, as shown by Figure 4.6, Prepose language supports the description of body postures and gestures from a single person. Both the allowed actions and body parts do not mention interaction between two or more users. That said, the grammar is extensible and could be modified to, for example, support the replacement of the "your" word by a codename for each user such as “user 1” and “user 2.”

Structure The basic unit of the Prepose language is the *action*. By its turn, a *pose* can contain one or more *actions*. These *actions* can be either transformations or restrictions, being separated by commas. Each *pose* represents a state of the given skeleton in time, meaning all its *actions* must be satisfied all together at once. On Prepose, the coded *pose* refers to few specific postures which are capable of defining the gesture as the called *key pose* by Miranda and colleagues (2012; 2014), *salient posture* by Vieira and colleagues (2012), or *configuration* by Bedregal and colleagues (2006). Both static and dynamic gestures can be described by *poses*. The static gesture consists of a single *pose*, and the dynamic gesture is represented by the transition through two or more *poses*. Gestures include *pose declarations*, followed by an *execution sequence*. For example, a gesture for doing the "ola", which is a reference to a wave staging by the crowd of fans on a stadium, contains the *execution* shown in Code 4.4.

Code 4.4: Prepose *execution* of the "ola" gesture.

```
EXECUTION:
point-hands-up,
point-hands-forward,
point-hands-down.
```

That is, to perform the "ola", the user needs to execute three pre-declared *poses* in a particular order. The three *poses* are "point-hands-up"; "point-hands-forward"; and "point-hands-down". Each of these *poses* can be as simple or as complex as wanted. The *execution* sequence of a gesture can use any *pose* defined by any loaded gesture, which allows developers to build libraries of *poses* that can be shared by different gestures. This feature can be particularly useful when considering dynamic gestures that use similar bases at the start, middle or end of the motion.

The gesture *execution* also supports the *repeat* keyword allowing the description of repetitions of repetitions of the sequence of *execution steps*. This way, it is possible to define a gesture *execution* as "repeat 2 times the following steps: pose1, pose2." meaning the user must perform in sequence the executions of pose1, pose2 and then pose1 and pose2 again.

Interruptions The user must match each *pose* in the sequence, using the same *order* of *poses* from the *execution steps* on the gesture description. The gesture matching succeeds only when the last *pose* in the sequence matches. The *transition* between *poses* must be *valid* as well. The *transition* is *valid* if on the way to the next *pose* the user continuously approximates his posture to *satisfy* the required *actions* of the next *pose*. If the user's movements do not respect either the *order* of *poses* or the *transition* criteria the gesture recognition interrupts the current performance, braking it and resetting the current target *pose* to the first *pose* of the *execution steps*. This approach dismisses the need for a time window[6] to be used for recognizing dynamic

[6]Time windows are usually applied for the recognition of dynamic gestures, giving the user a specific amount of time to perform the entire gesture from start to end. A time window starts from the current frame and considers a set

gestures. In Prepose the user can take as long as needed to complete a gesture, it only requires her movements to be always employed to reach the next *pose*. Therefore, Prepose seamlessly supports gestures with different durations.

Execution Underneath the execution, the transformations act as a function that takes as input a skeleton, modifies it and returns the updated skeleton. Currently, transformations in Prepose include *Point* and *Rotate*, as in the Prepose language shown in Code 4.5. The first line is a *Point* transformation, which takes as arguments the name of a user skeleton joint (*right wrist*) and a direction (*up*). In the second line, the transformation *Rotate* takes as arguments the name of the user skeleton joint (*left wrist*), the amount of rotation (*30 degrees*), and the direction of rotation (*front*). The third line is similar to the second one, targeting the right wrist instead of the left.

Code 4.5: Transformations coded on Prepose.

```
POSE sample_pose:
point your right wrist up,
rotate your left wrist 30 degrees to the front,
rotate your right wrist 30 degrees to the front.
```

When applied to a skeleton position, the effect of all three transformations are applied using the same order presented in the code, producing a single new target skeleton for the user. On the target skeleton, the right wrist will be mostly directed upwards, but rotated 30 degrees to the user front, while the left wrist will be rotated 30 degrees to the user front in comparison with the user skeleton state at the moment the sample_pose was targeted by the current *execution step*.

A restriction is a function that takes as input a Kinect skeleton, checks if the skeleton falls within a range of allowed positions determined by a Boolean expression, which then returns true or false. If it returns false, it also returns a value between 0 and 1 expressing how close the input skeleton is to fulfill that expression. This feedback allows the use of continuous *Flow*. To make it available we coded for each expression its completion percentage. For example, for the *Align* restriction the percentage is defined by the following expression $percentage = degrees/threshold$, in which *percentage* represents the result, *degrees* determine the current angle (in degrees) between the two body parts, and *threshold* defines the maximum angular distance accepted as an alignment (currently 20, which we defined through basic experimentation). After calculating the percentage, we crop the result value to fit the interval between 0 and 1. The *pose* matches if all restrictions are *satisfied* and the user body position is close to the target state.

Analysis In Prepose we use static analysis to search for the issues regarding gestures internal *validity, conflicts, and safety*. The use of static analysis allows us to check for these matters before runtime, i.e. before a tester or user has to experiment the gesture recognizer. For instance,

of previous frames sufficient to fit the specified time window. Generally, at the arrival of each new frame, a previous frame is discarded from the window, restricting the point in time the gesture start could be considered.

static analysis as a post-built event, just after writing and compiling the gesture. Therefore, we built Prepose to compile programs written in the Prepose language to formulas in Z3, a state-of-the-art Satisfiable Modulo Theories (SMT) solver (DE MOURA; BJØRNER, 2008). The Code 4.6 shows the stages of runtime processing in Prepose. A high-level Prepose statement is compiled into C# for matching and to a Z3 formula for analysis.

Code 4.6: Prepose code translated to C# and later to Z3.

```
PREPOSE
point your arms down

C#
public static BodyTransform ArmsDownTransform() {
return new BodyTransform()
.Compose(JointType.ElbowLeft, new Direction(0, -1, 0))
.Compose(JointType.WristLeft, new Direction(0, -1, 0))
.Compose(JointType.ElbowRight, new Direction(0, -1, 0))
.Compose(JointType.WristRight, new Direction(0, -1, 0));}

Z3
joints['elbow left'].Y > -1 &
joints['elbow left'].X = 0 &
joints['elbow left'].Z = 0
```

By using Z3 it is possible to define a generic skeleton data, with no defined positions regarding its joints, and ask its solver to find, for example, if there is a solution for that skeleton that makes it fit a given gesture (checking *pose* by *pose*). If the solver returns that this requisition is *unsatisfiable*, that means there are no possible set of 3D vectors for the given skeleton which could *satisfy* that gesture description, therefore showing that the gesture is *invalid*. For *conflicts* we ask Z3 if two gestures (*pose* per *pose*, $O(n^2)$ complexity) can be triggered at the same time; if so, Z3 will return a sample skeleton which *satisfies* the query. For *safety*, we coded a set of *safety* restrictions (such as "do not bend your neck by more than 90 degrees") and asked Z3 if it was possible to satisfy the given gesture without compromising any of the defined restrictions. This set of *safety* restrictions can be later extended, and personalized by the user if desired. These analyses and their background motivations are better detailed on the following subsection.

Safety, conflicts, and validity By of building the recognizers on top of a SMT solver, we can apply static analysis to the Prepose program. The first analysis employed is for gesture *safety*. A gesture description may induce people to overextend their limbs, make an obscene motion, or otherwise potentially harm the user. To prevent a threatening gesture from being delivered to an application we first define *safety* restrictions. Safety restrictions are common Prepose restrictions

but expressly encoded as SMT formulas that specify disallowed positions for Kinect skeleton joints. The goal of these restrictions is to make sure we do not break any bones by asking the user to follow this gesture. We define a collection of *safety* restrictions about the head, spine, shoulders, elbows, hips, and legs. Each gesture is checked to see if it is only *satisfied* by an *unsafe pose*.

We say that a pair of gestures *conflicts* if the user movements match both gestures simultaneously. Gesture *conflicts* can happen accidentally because one or more developers can write new gestures without fully considering the descriptions of the ones that were written previously. On machine learning approaches an example of avoiding *conflicts* while training gestures are presented by Faugeroux and colleagues (2014). During our review of rule-based approaches, no work was found with this capability. In Prepose, because all gestures have semantics in terms of SMT formulas, we can ask a solver if there exists a sequence of body positions that matches both gestures. If the solver completes, then either it certifies that there is no such sequence or gives an example that satisfies such conditions.

It is possible in Prepose to write a gesture that can never be matched. We also want to ensure that our gesture is not inherently contradictory (*invalid*), i.e. all sequences of body positions will fail to match. As example of a gesture that is *invalid*, consider the use of two restrictions within the same *pose "put your arms up; put your arms down;"*. Both of these requirements cannot be satisfied at once. To check for *validity*, we ask Z3 if the imposed conditions are satisfiable.

Additionally, several NUI systems include so-called system attention positions that users invoke to get privileged access to the system. These are the gesture-based equivalents of an *Alt* + *Tab* on a Windows system. For example, the Kinect on Xbox has a Kinect activation gesture that brings up a unique system menu no matter which game is currently being played. For Google Glass, a similar utterance is "Okay Glass". On Google Now for Android phones, the utterance is "Okay Google". We want to make sure that Prepose gestures do not attempt to redefine system attention positions. Thus, a particular case of *conflict* detection is detecting overlap with reserved gestures. For example, the Xbox One Kinect has a singular attention gesture that opens the Xbox OS menu even if another game or program is running. Checking *conflicts* with reserved gestures is important because applications should not be able to "cast a shadow over" the system attention gesture with its gestures.

Distance approximation We detail in this subsection a particular issue regarding the challenge of transcribing Prepose gestures into Z3 formulas. The use of Z3 brings clear advantages regarding the possibilities of static analysis. However, there are also some drawbacks. Unfortunately, Z3 has a conceptual limitation, being able to solve only linear expressions.

The distance between joints is required by definition from some restrictions such as *Touch*, used to define two joints that are in contact with each other. This limitation implies that calculations of pair exponents (e.g. X^2), for example, are not entirely supported. Given this

limitation, the use of Euclidean distance calculation is not supported. Z3 may even give an answer, but with some significant processing time efforts (losing its advantage of being a fast SMT solver), and not guaranteeing its correctness.

Therefore, we searched for other options to calculate the distance between two 3D points that did not require the use of any exponents. As an alternative, the Manhattan distance can be employed, but it is inaccurate, especially in cases where two coordinates have similar lengths. Being $v1 = (1,1,1)$ a vector which represents the subtraction between two points (displacement vector) and its norm represents the distance between these points, using the Manhattan method this distance would be 3 instead of $\sqrt{3}$, value 73% higher than expected.

To be more accurate regarding the distance measurement of two 3D points we managed to develop an approximation function for that task. Our estimate considers only the use of *MIN, MAX* and *ABS* functions that are fully supported by Z3. Our calculation uses similar concepts as the Manhattan distance does; we consider adding up the three coordinate values. However, we propose a different addition process. The central idea of our proposed approximation is to apply weights on the *common lengths* on each subset of the three coordinates (X,Y,Z), before adding it to obtain the final distance. For instance, the common length on the vector $(1,0.7,0.3)$ are 0.3 between all three coordinates and 0.7 between the X and Y coordinates. The applied weights are precisely the square roots of the number of coordinates on the particular common length. On the example showed above, the weight of the 0.3 length would be $\sqrt{3}$ and the weight of the 0.7 length $\sqrt{2}$. Therefore, the weights are meant to boost the final calculated length. On the counter part, common lengths are computed once, which differs from the Manhattan distance calculation. The following C++ Code 4.7 calculates the approximate distance developed in this work.

Code 4.7: Distance approximation function written in C++.

```
static const double sqrt2 = 1.41421356237309504880168872420 97;
static const double sqrt3 = 1.73205080756887729352744634150 59;
double approximated(double x, double y, double z)
{
const double absX = abs(x);
const double absY = abs(y);
const double absZ = abs(z);

const double firstCommonLength = std::min({ absX, absY, absZ });
const double secondCommonLength = (std::max({
std::min(absX, absY),
std::min(absX, absZ),
std::min(absY, absZ) }) -
firstCommonPart);

const double exclusiveLength = std::max({ absX, absY, absZ }) -
```

```
secondCommonLength -
firstCommonLength;

return firstCommonLength * sqrt3 +
secondCommonLength * sqrt2 +
exclusivePart;
}
```

The approximation concept can be explained by treating the problem of distance calculation as the problem of computing the norm of a vector, being the vector the result of the subtraction between two points. A sample vector $(3,4)$ in the 2D space is shown on part A.1 of Figure 4.7. The correct norm of this vector is 5, calculated by the Euclidean distance. By its turn, the norm calculation can be approached as the decomposition of the input vector into other vectors, which are measurable by the used method. Using this viewpoint, the Manhattan method can only measure single coordinate vectors, and then it decomposes the $(3,4)$ vector into two other measurable vectors which are $(3,0)$ and $(0,4)$ as shown in Figure 4.7 A.2. The Manhattan method adds the measured values of the decomposed vectors, which are 3 and 4 to obtain the norm of the input vector 7.

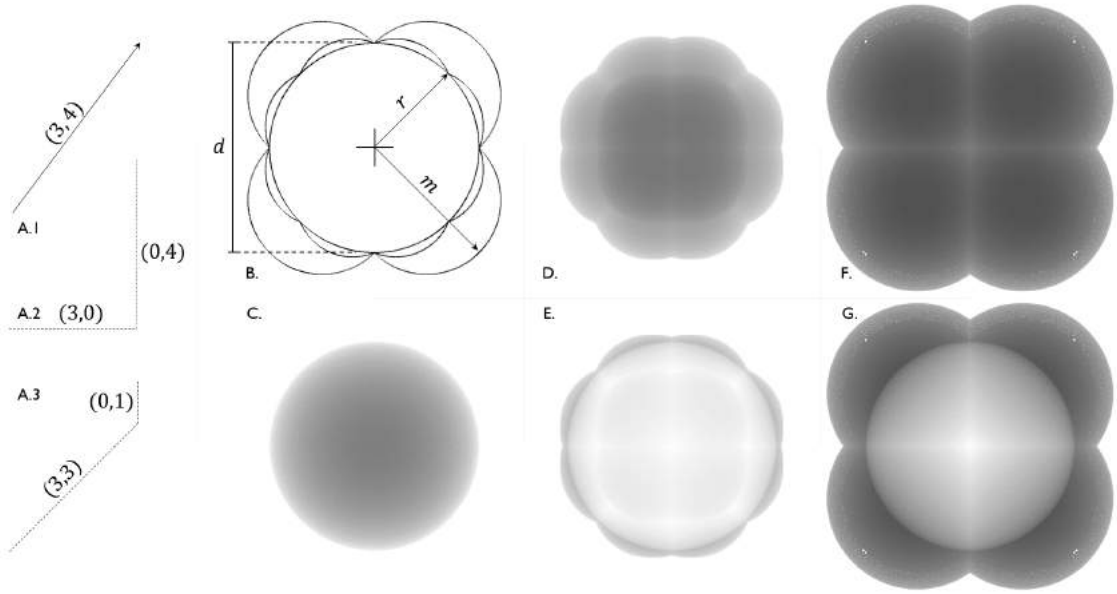

Figure 4.7: Visualization of approximated and Manhattan distances' functions by rendering several sampled vectors on a sphere and redefining the norm of each vector based on the calculated distance by each function.

The less the decomposed vectors deviate from the original orientation of the entry vector, the smaller will be the error. This way, the developed approximation adds to the list of measurable vectors the ones which have equal length on any pair of coordinates, e.g. $(3,3)$ in Figure 4.7 A.3. The measurement of these vectors is calculated by multiplying the length of one of the sides (e.g. 3) by a *constant*, which for 2D vectors is $\sqrt{2}$. The approximation then uses the best set

Table 4.1: Error measures of the Manhattan distance and the developed method.

Method	Average Error	Standard Deviation	Minimum	Maximum
Manhattan	54.6%	14.19%	0%	73.2%
Proposed method	8.15%	2.97%	0%	12.8%

of decomposed vectors, in this case, $(3,3)$ and $(0,1)$, to calculate the norm of the input vector. The resulting norm for the vector $(3,4)$ using the approximation approach is 5.24. In the 3D space there is also the case when all three coordinates have common lengths, for example on the vector $(3,2,1)$, the vector $(1,1,1)$ is one of the results of the decomposition. In these cases, the *constant* is $\sqrt{3}$ as shown in code 4.7.

Part B of Figure 4.7 renders a circle of radius r and diameter d as the result of plotting 2D points of sampled vectors in all directions with norm r. To illustrate the error resulting from the measurement of the norm by the Manhattan method each one of the vectors norms are set to the corresponding measure (i.e. the m vector), which results in the shape of four big arcs. The same procedure is applied using the developed approximation method leading to the shape of eight small arcs. Since the nearer the resulting shape is to the circle, the smaller is the measurement error, it is possible to see that the approximation achieves a better result if compared to the Manhattan method.

To visualize the error in the 3D space the same procedure is performed rendering the Z coordinate with different gray tones; the nearer the Z is to 0 the brighter is the pixel. For reference, a sphere is rendered in Figure 4.7 part C, the approximation is shown in part D and the Manhattan on part F. Similarly to the 2D case, the nearer the rendered shape is to the sphere the smaller is the error. To better illustrate this error, Figure 4.7 part E shows the difference between the sphere and the shape obtained by our developed approximation, and part G illustrates the difference between the sphere and the shape achieved by using the Manhattan method. When the difference is 0, the rendered pixel is white. Table 4.1 shows the errors of both the Manhattan and the proposed approximation methods for distance calculation.

4.2.2.5 Experiment and results

To test our tool, we experimented writing gestures for three application domains that involve different styles of gestures: physical therapy, ballet, and tai-chi. Here we aimed to evaluate Prepose capabilities on a broader scenario, also considering full-body gestures and monitoring applications. Given the natural syntax of Prepose and a flat learning curve, other applications can be easily added to the system. For each of these gestures, we then performed the series of analyses enabled by Prepose, including *conflict* detection and *safety* checks. The written gestures are listed on Table 4.2.

We then measure runtime performance and static analysis performance of Prepose. We used the Kinect Studio tool, which ships with the Kinect for Windows SDK, to record depth and video traces. We recorded a trace performing two gestures. Each trace was about 20 seconds in

Table 4.2: Written gestures using Prepose. For each of the three domains the number of gestures, poses, lines of code and an URL containing the final file.

Application	Gestures	Poses	LOC	URL
Therapy	12	28	225	http://pastebin.com/ARndNHdu
Ballet	11	16	156	http://pastebin.com/c9nz6NP8
Tai-chi	5	32	314	http://pastebin.com/VwTcTYrW

length and consisted of about 20,000 frames. On the first frame, we observed matching times between 78 ms and 155 ms, but for all subsequent frames matching time dropped substantially. For these frames, the average matching time was 4 ms with a standard deviation of 1.08 ms, which is suitable for real-time recognition at 60 frames per second. For *pose* transition time, we observed an average duration of 89 ms, with a standard deviation of 36.5 ms. While this leads to a skipped frame each time we needed to create a new *pose*, this is still fast enough to avoid interrupting the user movements.

The right part of Figure 4.8 shows a near-linear dependence between the number of steps in a gesture and the time to check against *safety* restrictions. The average checking time is only 2 ms. On the center another near-linear dependence between the number of steps in a gesture and the time to check if the gesture is internally *valid*. The average checking time is 188.63 ms. We also performed pairwise *conflict* checking between 111 pairs of gestures from our domains. The part C of Figure 4.8 shows the Cumulative Distribution Function (CDF) of *conflict* checking times, with the x-axis in logarithmic scale. The CDF allows quick visualization of which percentage of the data is below a certain threshold. For 90% of the cases, the checking time is below 0.170 seconds, while 97% of the cases took less than 5 seconds and 99% less than 15 seconds. Only one query out of the 111 took longer than 15 seconds. As a result, with a timeout of 15 seconds, only one query would need attention from a human auditor.

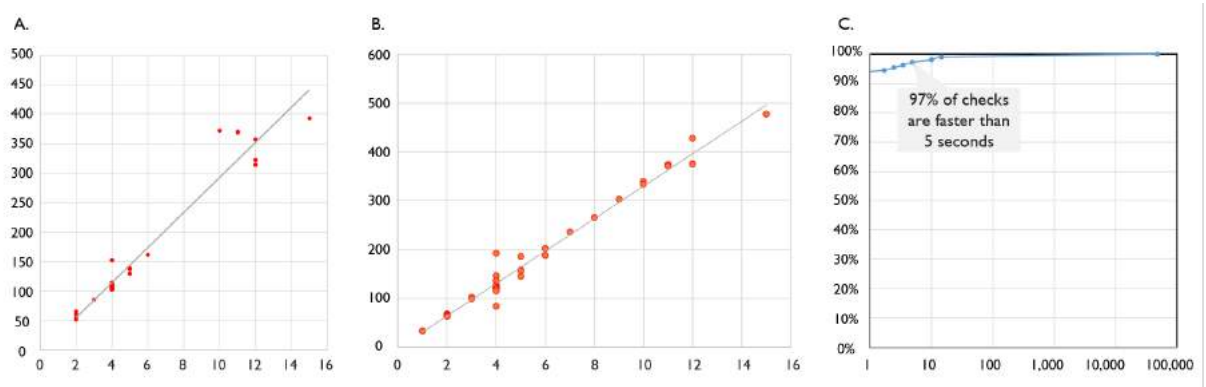

Figure 4.8: On part A time to check for *safety*, in milliseconds, as a function of the number of steps in the underlying gesture. On part B time to check internal *validity*, in milliseconds, as a function of the number of steps in the underlying gesture. On part C time to check *conflicts* for a pair of gestures presented as a CDF. The *x* axis is seconds plotted on a logarithmic scale.

4.2.2.6 Prepose IDE

Coding We also built a development environment for writing, compiling and executing Prepose gestures, which we call Prepose IDE. Figure 4.9 shows a screenshot of our tool. On the left, a text entry box allows a developer to write Prepose code. We also added a syntax highlighting, coloring reserved words to improve the writing and reading experience. Furthermore, we introduced the auto-complete feature, which helps writers to describe gestures faster and with the precise phrasing. Our environment also supports comments, to enhance the readability of the written code. On the right part, the tool shows the current user position in green as feedback of what the sensor is tracking on real-time.

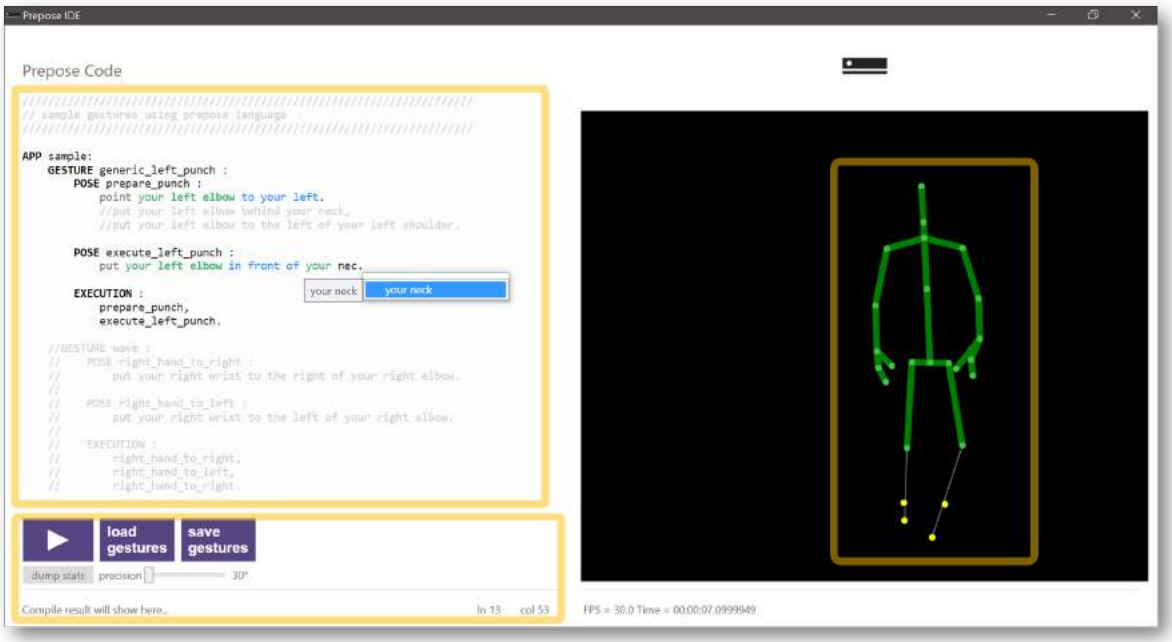

Figure 4.9: Screenshot of Prepose IDE coding view.

On the bottom part, it offers the option to run (play button) the currently written gestures and to load or save gestures. The *dump stats* button runs all the analyses and stores it on a spreadsheet (.csv) file for consultation. The precision slider allows the configuration of the minimum required angle to consider a *pose* as a match. The precision can be tuned depending on the application goal, for example, on physiotherapy the therapist may need the user to reach as well as possible a particular posture, while on games, in general the precision is of low concern and the users should have more freedom to execute the gestures. At last, on the very bottom, the interface shows the compilation status, showing all the occurred errors and specifying the line and column where the error was found.

Recognizing Once the play button is hit, the code is compiled, and if it was successfully compiled the Prepose IDE switches to the recognition view as shown by Figure 4.10. Here we display the user in white. On the right, the tool gives feedback about each written gesture.

Prepose can run several gestures simultaneously; therefore, we allow the test of gesture sets as discussed on our Wizard of Oz experiment. For each gesture, the interface shows a green progress bar illustrating how close the user is to complete the gesture. This bar value is presented as an exact division of the full bar length by the number of *poses* contained on the gesture. A gesture with two *poses*, on which the user completed the first *pose* and is seeking to complete the second would show at least half of the bar filled. Depending on the second *pose* percentage, a respective percentage of the second half of the bar is filled as well. A *pose* percentage is defined by the lowest percentage of its containing *actions* (transformations or restrictions).

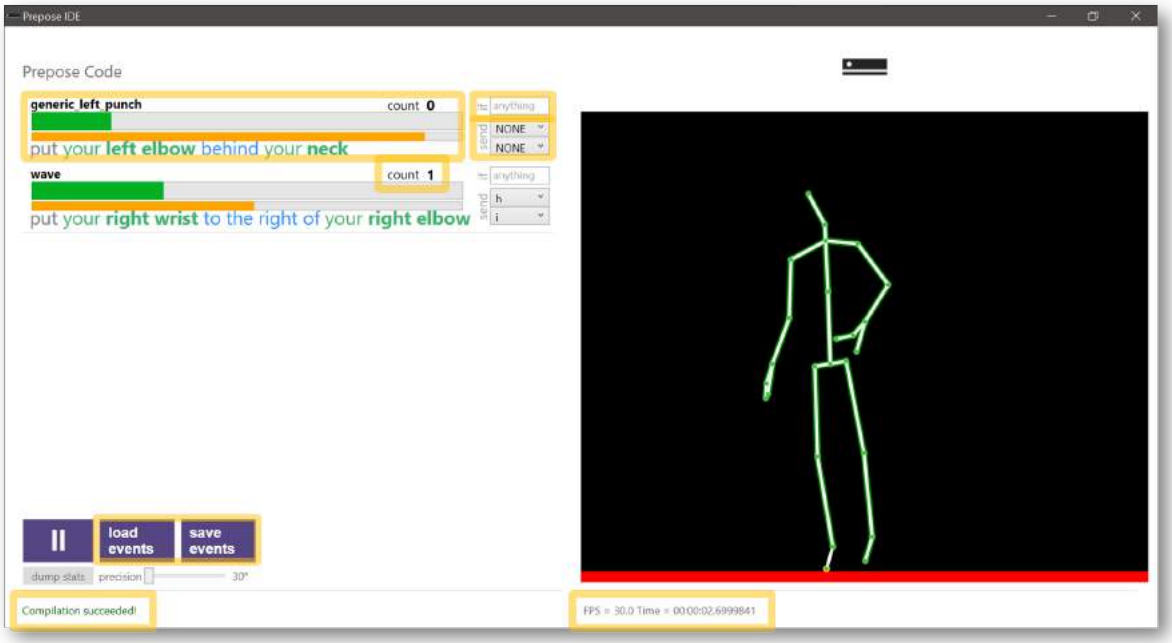

Figure 4.10: Screenshot of Prepose IDE recognition view.

We also included a yellow and thinner progress bar illustrating how close the gesture is to be interrupted. As previously discussed, a gesture is interrupted (i.e. reset to its first *pose*) once the user moves in the opposite direction of the next *pose*. Moreover, we present as textual feedback the *action* with the lowest percentage, which is the *action* the user is revealing to be more distant to accomplishing at that exact moment. We show on top of the progress bars a count of completed gestures.

Output Prepose can be used as a managed C# library, giving developers full access to the code of each gesture status on real-time, and this way, producing the desired output for their prototypes. Also, we aim to provide output options for non-developers as well. On the left of each gesture progress feedback, we present a mini-menu allowing the execution of the gesture to trigger some events as output.

For example, the *generic_left_punch* gesture triggers no action because its event options are set as *NONE*. The *wave* gesture by its turn sends two events, the letters “h” and “i” which

will make each execution of the gesture type the "hi' word using keyboard events of Windows OS. We also added support to special keys such as *Alt, Shift, Tab, Enter, F1, F2, F3, F4, Esc, Left arrow, Page down* and so on. This way, while prototyping using Prepose IDE it is possible to use this resource to perform Windows commands such as "Alt + Tab", "Alt + F4", "Shift + Delete", "Ctrl + Esc" and so on. Therefore, this resource allows the prototyping of some tasks like *Home, Switch, Previous, Next*, and more, on some Windows applications.

We also added support to mouse events, so a gesture can trigger a mouse click (left, right and middle buttons), wheel events (scroll up and down) and positioning events. The position events allow the application to set the mouse position on the screen. For triggering these events, instead of sending it only once each gesture is completed, we used the current gesture completion percentage. This way, it is possible to assign a gesture like "point your right hand to the right" as the control for the X axis of the mouse position. If the right hand is 100% pointing to the right, the completion percentage is 100% and the mouse hits the right border of the screen, if the hand points 100% to the left, the percentage is 0%, and we assigned the 0% to position the mouse 100% to the left of the screen.

At last, we added an option to use one gesture completion as a condition to trigger the events of another gesture. This option is given on the *"if anything"* text box. By replacing the "anything" with the name of the required gesture, the Prepose IDE starts using its completion as a condition to trigger that gesture. This way, it is possible to assign a previous gesture, which allows the user to take control of the second gesture. For example, the mouse controlling gesture may produce an undesired effect known as the "Midas touch" problem, which refers to unintentionally triggered events. If the user guides the mouse with the direction his right-hand vector (from the skeleton joints), the mouse will be controlled through the entire interaction, even when the user forgets about it, the mouse will move. By using the *"if"* option, designers can assign a "take control" gesture and the mouse controlling will be only triggered if the "take control gesture" is active.

Events can also be saved and loaded. For each saved gesture, if there is an events file with the same name (different extensions), the events file will be automatically loaded once that gesture is loaded. We designed these output options to enable easy ways of prototyping gestures. At the same time, we recognize that mouse and keyboard outputs may not suffice the prototype needs. For example, to control a TV, it may be required to manipulate a remote control interface. On such cases, our tool does not provide a proper output, and the designers must search for alternatives. We would like to point also the possibility that designers can use Prepose IDE to trigger mouse and keyboard events, and then capture these events and transform them into the desired output, for example, network messages, robot controlling events through specific APIs, and so on.

4.2.3 Training-based recognition

On the counterpart, describing complex gestures may require a considerable amount of time, even in natural language. A complex gesture may be defined by more than ten *poses*, or use the relation between several body parts, and writing each relationship can be unfeasible considering the development cycle, as well as lead to human error. This way, we acknowledge that writing gestures may be not practical for some scenarios, and therefore, we propose the use of training-based recognition as a complementary approach. We incorporate in Prepose the option of recognizing gestures based on a one-shot learning algorithm.

4.2.3.1 Machine learning related work

Machine learning has been extensively used for the task of gesture recognition, targeting both static and dynamic gestures. Examples include Naïve Bayes (NB), Decision Tree (DT) classifiers (PREIS et al., 2012; PATSADU; NUKOOLKIT; WATANAPA, 2012; BHATTACHARYA; CZEJDO; PEREZ, 2012) and Random Forests (RFs) classifiers (SCHMIDT et al., 2014; FOTHERGILL et al., 2012). There are also works using the Nearest Neighbors (NNs) and Decision Forests (DFs) approaches (LAI; KONRAD; ISHWAR, 2012; MIRANDA et al., 2012, 2014). Particularly, Support Vector Machines (SVM) (PATSADU; NUKOOLKIT; WATANAPA, 2012; ZHANG et al., 2014; VEMULAPALLI; ARRATE; CHELLAPPA, 2014; MIRANDA et al., 2012, 2014; CHAUDHRY et al., 2013; MARIN; DOMINIO; ZANUTTIGH, 2014; SCHMIDT et al., 2014; BHATTACHARYA; CZEJDO; PEREZ, 2012) and Hidden Markov Models (HMM) (GU et al., 2012; SONG et al., 2013; VAMSIKRISHNA; DOGRA; DESARKAR, 2015; WANG et al., 2014; FOK et al., 2015; WU; CHENG, 2014) are largely applied to solve gesture recognition as a classification problem.

The Dynamic Time Warping (DTW) algorithm is particularly useful for dynamic gestures. Due to its property of correlating time series, the DTW has also being used to align the chosen features over time, which usually are trajectories of skeleton joints. It can be used as a preprocessing step to deliver a more cohesive data for a further machine learning technique (VEMULAPALLI; ARRATE; CHELLAPPA, 2014). Moreover, it can also be utilized as a recognition technique itself by using the trajectories of dynamic gestures aligned over time to define the ideal trajectory and its boundaries (IBAÑEZ et al., 2014). Weighted DTW can also be applied to reduce the impact of taking into account unnecessary joints, e.g. considering the leg joints movement on an arm swing gesture or considering the thumb position on a forefinger pointing gesture (CELEBI et al., 2013; ARICI et al., 2014).

All these approaches tackle the gesture recognition from a supervised learning viewpoint, by gathering a set of previously executed gestures to train the classifiers on an offline phase, and later use this classifier to identify if an input gesture belongs to any of the trained classes. These approaches proved to achieve high rates of recognition once a set of previously labeled training gestures is provided. It is usually recommended to gather the training data considering possible

variations, such as environment occlusions and self-occlusions, different sensor positions and different users. These variations allow the recognition technique to achieve invariance to these variation conditions, being more robust.

4.2.3.2 Background work: Checkpoints

At the same time, by requiring an extensive set of training data, the use of these techniques also requires a considerable amount of time to collect and tag that data. This required time may be critical on fast prototyping cycles, while ideas are still being validated and refined. However, we understand that while developing a gestural interface, the *Prototype* phase can be useful to comprehend the impact of a particular gesture choice on the interaction. The prototyping process can benefit from the fast creation of gesture recognizers, as shown by Faugeroux and colleagues (2014), this way allowing new gestures to experiment with a small amount of time. Thus, we focus on a one-shot[7] gesture learner, by developing a fast and straightforward technique to recognize human movements.

We introduce a tool called Checkpoints, which we use to validate our concept of one-shot learning solution for the gesture recognition problem. Checkpoints is a separate tool from Prepose. We detail the Checkpoints system to expose the background work used as the base for the Prepose learning strategy and solution. Checkpoints system for recording a movement as well as for its recognition is summarized in Figure 4.11.

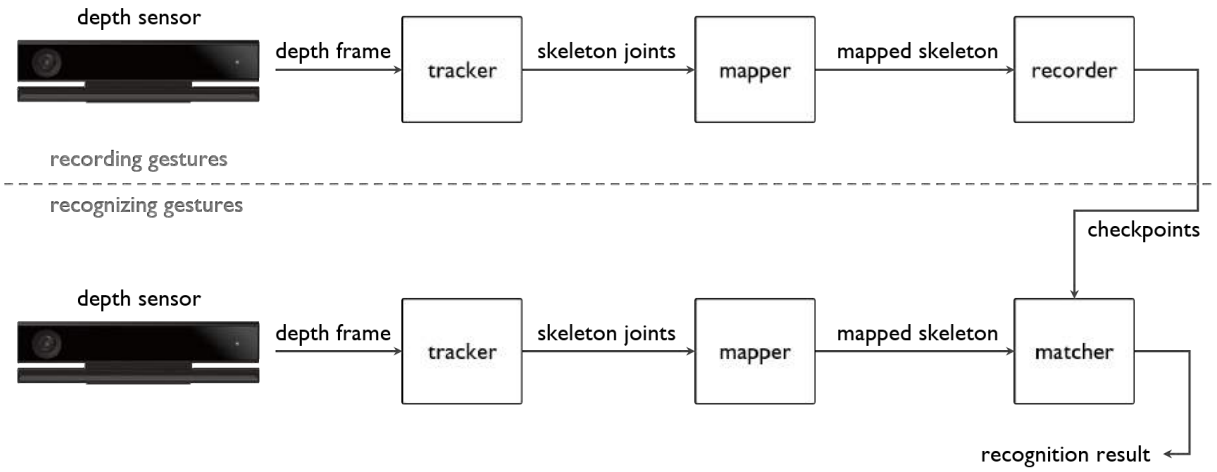

Figure 4.11: System overview for gesture recording and recognition.

Skeleton Similarly as detailed on the Prepose system overview (subsection 4.2.2.4), we also apply a change of representation of the skeleton data as an initial step on the Checkpoints system. The new representation describes each body segment (e.g. from elbow to wrist) as normalized direction vectors, therefore the length of the vectors does not matter, only their

[7] One-shot learning solutions require a training set containing a single sample of the necessary data. Therefore, the efforts to collect and tag the training data are considerably reduced.

direction. Also, for each vector, we apply a change of basis to be invariant to rotations. The same new representation (shown as the *mapped skeleton* on Figure 4.11) is applied for both training (recording), and recognizing (matcher). More details of the used skeleton representation can be found in the paper entitled "Human body motion and gestures recognition based on checkpoints" (CHAVES et al., 2012).

Training The new skeleton representation is sent to the recorder, which evaluates *key states* (or *poses*) of the gesture and registers it in a structure called *checkpoint*. The *checkpoint* represents the same concept as the *pose* does in Prepose language, and are the core unit of the Checkpoints system. A *checkpoint* contains the set of vectors from the new skeleton representation. Each set of vectors represents the state of the movement at a certain point in time.

For the recording procedure, we specify which body segments must be tracked for the recognition. This way, on Checkpoints we allow partial gestures, considering only the body parts relevant to the specified gesture. The first *checkpoint* is recorded at the start of the recording session, and from that point, if the direction of any vector from the *selected body parts* reaches an angular distance threshold (e.g. 30 degrees), a new *checkpoint* is added for that state. Therefore, a movement is defined by a sequence of states, or *checkpoints*. With a single execution of the specified gesture, we generate a descriptor of the gesture through a set of *checkpoints*. This way, we provide a *one-shot* learning strategy for our tool.

Recognizing The *checkpoints* saved from the recording step are then used for the recognition. The matching procedure analyzes the *selected body parts* of the input skeleton (on the new representation), comparing it with the vectors of each *checkpoint*. To validate if the user posture fits within the gesture space we perform the following procedure. First, we link the *checkpoints* by lines. Then we select the point on these lines representing the point of shortest distance to the input vector, for this calculation we use the input vector as a point. We consider the selected point as a vector and then calculate the angular distance between it and the input vector as shown in Figure 4.12. This interchange between points and vectors is possible because the input and checkpoints' vectors are all normalized, and therefore can be interpreted as representing points on a sphere. If the minimum angular distance between the input vector and the *checkpoint* lines falls within the threshold, the current state is considered valid for this gesture. All selected skeleton segments must be within the required range[8] (e.g. 30 degrees). It is important to notice that, despite tracking the exact part of the gesture is being executed by the user at the moment, the Checkpoints solution does not interrupt or reset the gesture progress if they are performed in the opposite direction. The Checkpoints define only the gesture subspace, and the user can move back and forth between the recorded *checkpoints*.

[8]The required range can be tuned by changing the angular distance threshold.

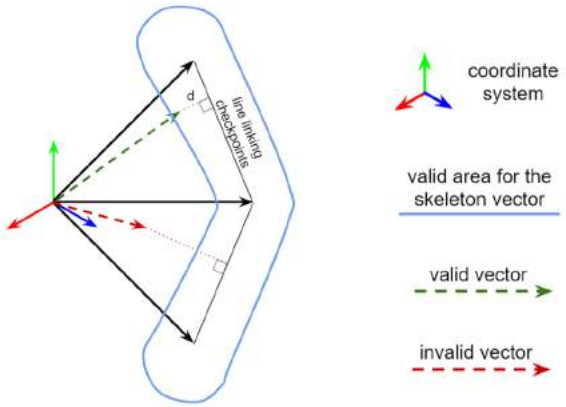

Figure 4.12: Checkpoints matching approach to validate a particular input vector (bone or skeleton segment).

Testing We tested the Checkpoints system considering variations of duration (1 second for fast and 3 seconds for slow gestures), precision (15° and 30° for the angular distance threshold), and two different gestures (one planar with a continuous unilateral movement, and another more complex) as shown in Figure 4.13. The figure also shows the test's scenario with the user performing a movement about two meters away from the Kinect sensor. The test consists of registering a set of moves on the training phase, one at a time, and then checking if the movement reproduced is validated by the system. This set is composed of movements made varying the parameters listed below, and the results are shown in Table 4.3.

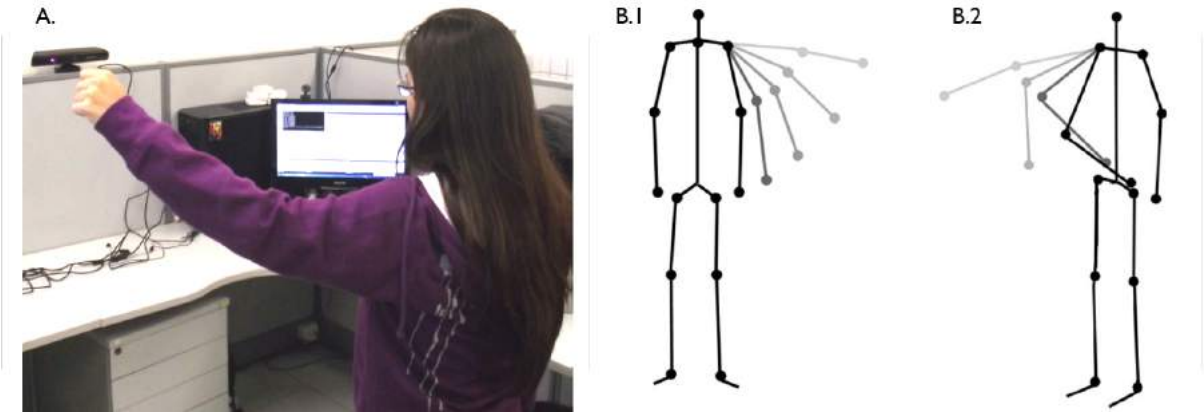

Figure 4.13: On part A, tests scenario. On part B, the gestures used for testing, an arm lateral elevation (part B.1) and a "drawing the sword" metaphor (part B.2).

The required time to validate the current state of the user skeleton was 1 millisecond. The results show a high rate of recognition, in most cases above 95%. The required precision and speed showed influence on the recognition rate of the complex gesture, presenting the worst rate of 64.49%. With the range of 30°, almost all movements were validated. On the other hand, by loosening the distance angle threshold, the recognition may accept some invalid executions as false-positives. To check false-positives, a test was conducted targeting wrong executions of the gestures. The tests were made varying the precision and the execution fidelity, introducing

Table 4.3: Results showing the amount of movements recognized as correct during its execution.

Range (°)	Duration (sec)	Complexity	Recognition(%)
15	1	Simple	95.07
15	1	Complex	64.49
15	3	Simple	98.14
15	3	Complex	77.27
30	1	Simple	100
30	1	Complex	94.93
30	3	Simple	100
30	3	Complex	99.76

Table 4.4: Tests to evaluate false-positives by incorrectly executing gestures.

Range (°)	Error Intensity	Discarded (%)
15	Gross	84.74
15	Soft	49.76
30	Gross	58.33
30	Soft	23.63

soft and gross errors. A soft error means a slight deviation from the gesture, while a gross error means to leave the track of movement completely. The results are shown in Table 4.4.

Most of the gross errors are discarded when a 15° of precision is set. The number of discarded soft errors is near to 50% with 15° of precision, but with a loose precision (30°), it falls to 23.63%. These data suggest that a trade-off should be found according to the application and the used gesture while configuring the required range, to discard a significant number of false-positives while maintaining a high rate of true-positives.

4.2.3.3 Prepose automatic gestures writing

As a complementary approach to writing gesture recognizers using natural language, we introduce a training-based approach integrated on Prepose. We use a similar strategy as the one presented on the Checkpoints system. We present the Prepose Recorder Tool as a tool to handle the goal of providing a one-shot gesture recognizer (requiring a single sample of the gesture execution as training data) for the high-fidelity prototyping step. Prepose Recorder Tool is an open source project and is available in a GitHub repository at `www.github.com/lsfcin/prepose`.

In some cases, writing down an entire gesture description may be troublesome, filled with intricate relationships between *joints* and *action*, and therefore requiring an extensive amount of time and being more susceptible to human errors. Prepose Recorder Tool can be used in these cases to write the entire gesture description automatically. While recording the sample execution, our tool automatically writes the observed performance, in real-time, using Prepose language. The resulting gesture description can be used then as input to recognize the gesture on

the Prepose IDE.

By following this approach, also, to provide a training-based solution, we maintain the advantages as mentioned earlier, such as the static analysis; it's clear semantics (since it is written in the natural language); and the capability of editing it by reviewing the gesture description. This is possible because the training phase mainly outputs *rules*, which are later used in the same way it would occur if humans wrote the gestures.

Skeleton The training for generating the gesture description follows a similar procedure as the one shown on the Checkpoints system. First, the input skeleton is remapped on the new skeleton representation (described on the Section 4.2.2.4 of Prepose system overview). Once the skeleton is represented in the new coordinate system, it is continuously evaluated by Prepose.

Training The tool considers only the set of selected joints (body parts relevant for the gesture) and a set of selected *actions* (e.g. *Put*, or *Align*). The selection of joints allows the gesture description to consider only the important joints of the gesture, writing no information about the unselected ones. This way, the written gesture description leaves the user free to move and position the unselected body parts as desired, supporting partial gestures. The selection of *actions* allows the designer who is using the Prepose Recorder Tool to control the specificity of the description for the gesture.

For example, if the selected *actions* are *Point*, and *Rotate*, the resulting description will be very accurate regarding the allowed postures on that gesture. These *actions* are transformations and specify rigorous body positions. If the selected *action* is *Put*, then the resulting description will likely give more freedom for the users while performing the gesture. The *Put action* is a restriction and can be *satisfied* by different body postures. This choice is directly related to the concepts of coverage and correctness discussed by Fothergill and colleagues (2012). This way, while aiming correctness (increasing the true-positives, and accepting the positive inputs) on the gesture description of Prepose, designers should also take care to not extend in excess the coverage so undesired motions will also be triggered as positive (increasing the false-positives).

After selecting *joints* and *actions*, the training session can start. The tool continuously evaluates (frame by frame) the selected joints of the input skeleton regarding the chosen *actions*. Each evaluation also generates a description of the current *pose*. In case the described *pose* is not yet found in the complete gesture description, the tool adds it to the gesture. This way, on the first frame at the start of the training session, a *pose* is instantly attached to the gesture. If the user does not move (e.g. while training a static gesture), no other *pose* will be added to the gesture. Once the user moves, a different description of the *pose* may be generated for the current frame, and then it will be added to the gesture.

For instance, if one of the selected *actions* is *Rotate*, it is likely that the *pose* will change every frame. These changes occur because the *Rotate action* describes rotations using directions (e.g. front, or up) and angles as numeric parameters, which present small variations even if the

user intends to stand still. For this *action*, we added an angular threshold (e.g. 30°) to ignore variations within that range (similarly as the training on the Checkpoints solution).

Prepose code defines *poses* as descriptions of the states of the user skeleton in time. Therefore, in a similar way as the Checkpoints, the *poses* (and the way from one *pose* to the other) represent the subspace of the gesture. However, on the Checkpoints system there are no concerns about the gesture direction, i.e. there are no consequences if the user starts from the 2nd *checkpoint*, then goes to the 3rd and finishes on the 1st.

On Prepose, in addition to defining this subspace (by specifying the *poses*), we also define the required direction and order of *poses* the user should perform to complete the gesture. The *execution steps* define this direction, as shown on Code 4.4. This way, while training (and writing) the gesture, each time a new *pose* is performed, an *execution step* is added to the gesture description. Therefore, each new *pose* added to the gesture automatically generates a new *execution step*. Moreover, if the user performs an existing *pose* again (e.g. the user went back and forth in a movement), the tool adds an *execution step* for that *pose*.

Interface Developers can use the Prepose Recorder Tool as a managed C# library, which provides full access to all its features. Besides, we provide a visual interface to facilitate the use of the tool by developers and non-developers as shown in Figure 4.14. The interface shows a settings tab which allows the configuration of the selected body parts and the chosen *actions*. For example, in the case shown in Figure 4.14 the selected body parts are the user *right elbow* and *right wrist*, and the chosen *action* is the *Put action*, which describes spatial relationships between joints. The interface also gives the option to set a time interval for the recording session (10 seconds in this case). During our previous recording experiments with the Checkpoints tests, we found value in adding an initial countdown to start the session (3 seconds in this case), this way a single person can use the interface and then move to be in front of the sensor to perform the gesture.

Once the recording button (circle on the bottom-left corner of the screen) is hit, the countdown starts and the tab container switches to the *Code* tab as shown in Figure 4.15. While the user is performing the gesture, the tool simultaneously writes the gesture description on the code tab. In the example of Figure 4.15, the tool describes the *Put action* relationships between the user right elbow and right wrist. Each direction of the space (axis X, Y, and Z) are used in each *pose* description, specifying for that *pose* if the *wrist* was above or below, in front or behind, and on the right or the left of the *elbow*.

Each time a new configuration is found by the tool, it writes a new *pose*, and therefore a new *execution step*. The five *execution steps* on Figure 4.15 show that the user first performed the “pose_1”, then the “pose_2”, “pose_3”, and after she repeated the “pose_1” and “pose_2” again. If the user performed a single *execution* of the gesture, this means the gesture may have a cyclic aspect. Alternatively, the user can perform several repetitions of the same gesture, and the respective *execution steps* will be presented. Reproducing the gesture more than once can be

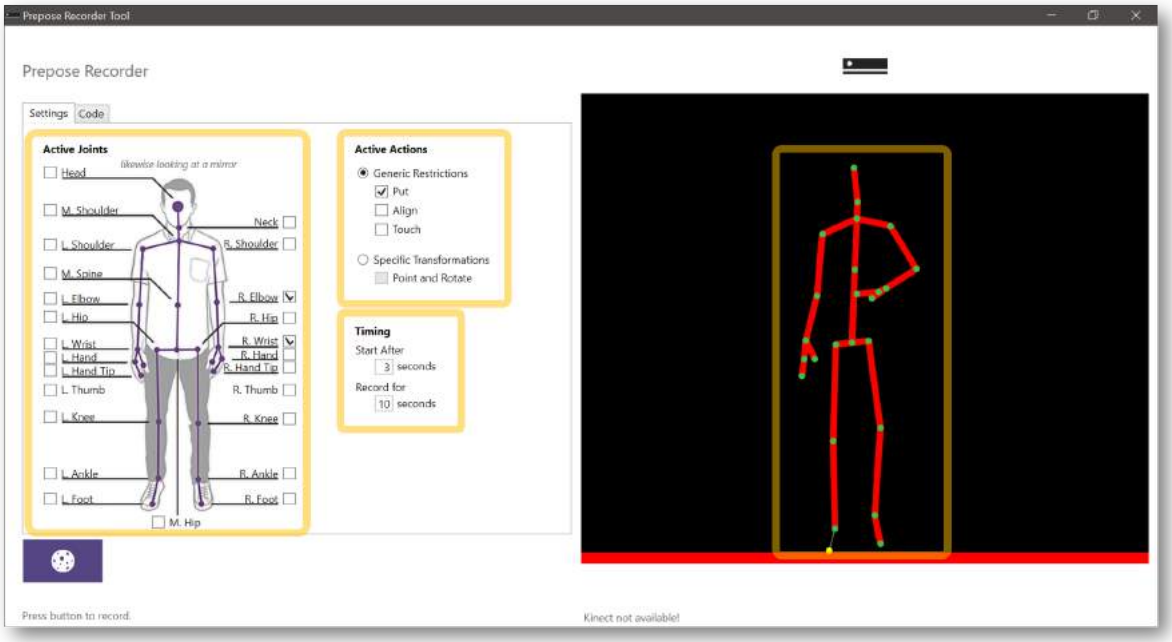

Figure 4.14: Visual interface of the settings tab on Prepose Recorder Tool.

useful for complex gestures. Complex gestures with a large set of selected *joints* and *actions* will likely generate a pack of *poses*. If desired, the number of poses can be reduced while analyzing the recorded *execution steps* and filtering the most repeated *poses*. This procedure may produce a more compact and manageable version of the gesture description.

4.3 Conclusion

In this Chapter we discussed and proposed options for the steps of low and high-fidelity prototyping of our toolset.

4.3.1 Low-fidelity prototyping

Regarding the low-fidelity prototyping, we proposed and evaluated the use of the Wizard of Oz technique. According to our knowledge, there is no work on the literature which evaluated the technique extent regarding its applicability for sustained gestures (static gestures requiring continuous *Flow*). We also compared the response time and precision of gestures recognized by Wizards versus a basic recognition solution using the Microsoft Kinect device. For precision, the rate of false-positives was nearly zero (0.007), and zero false negatives occurred. Regarding the achieved response times, at the end of a training procedure, the Wizards performed the recognition on an average latency of 93 milliseconds. In general, 100 milliseconds of latency is acceptable for interactive systems that are meant to make the user feel in direct control of the system. Therefore, we reinforce the potential of the technique for low-fidelity prototyping of in-air gestural interfaces.

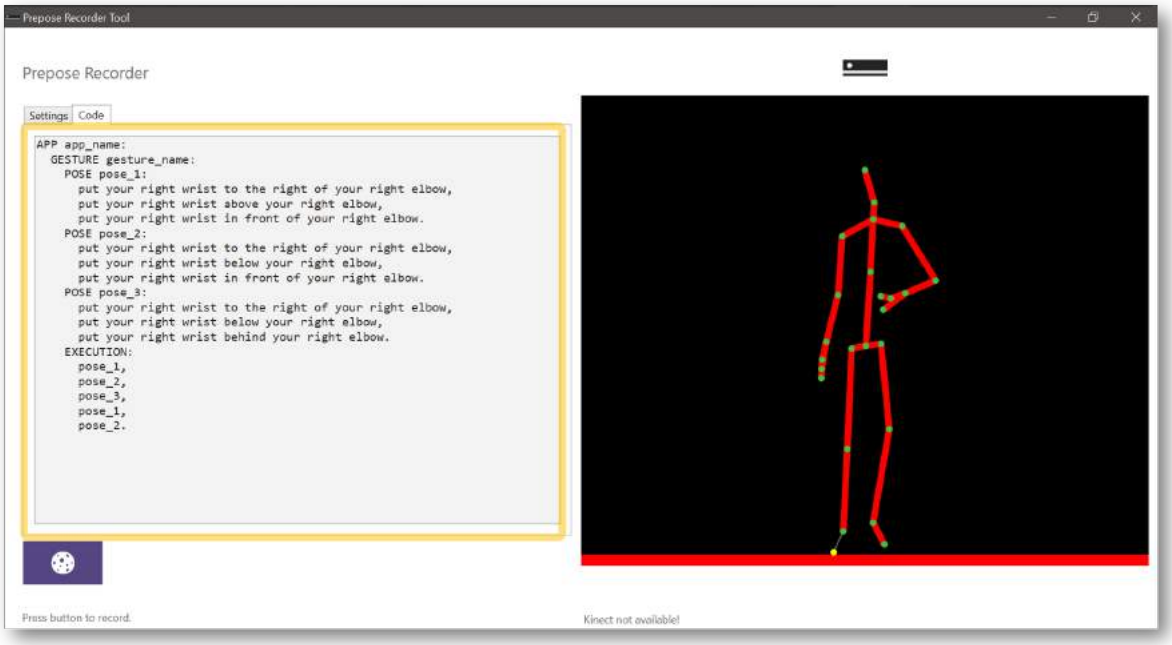

Figure 4.15: Visual interface of the code tab on Prepose Recorder Tool.

One significant finding is that assigning more than one task to a single Wizard may be cumbersome, stressing her cognitive load and undermining her performance. On the counterpart, Wizards could handle a pair of commands intrinsically related, such as rotate left and rotate right. At last, it is important to notice that our experiments did not explore dynamic gestures with continuous *Flow*. This particular case treats gestures that represent movements through space (dynamic) and receive continuous feedback for each state of motion (continuous *Flow*). Therefore, in this case, the output is not Boolean and requires a degree of refinement that was not explored yet.

On a more fluid interaction, the user should be able to define intensity levels for each command through gestures; there are also several other tasks which require a more precise recognition (e.g., pointing on a 2D screen) with continuous feedback. In these cases the Wizard of Oz, as it was used, may not work. For instance, Lee and colleagues (2013) explored the pointing task using the Wizard of Oz technique, and the user's response showed frustration on the following quotes: "the fidelity of the feedback was low and unreliable. Thus, the task performance could not be handled" and "It was impossible to point at a specific position on the screen in the experiment." Once a performed gesture allows intensity control, the intensity level becomes ambiguous for the Wizard requiring the use of additional tools to track this level in real-time. Further detail about our Wizard of Oz experiment can be found in the paper entitled "In-place natural and effortless navigation for large industrial scenarios" (FIGUEIREDO et al., 2014).

4.3.2 High-fidelity prototyping

For high-fidelity prototyping, we introduce a tool called Prepose. Prepose is a tool that brings together the concepts of rule-based authorial recognizers with training-based recognizers. Prepose answers not only if the gesture is recognized but also in which state of the execution is the user, returning a continuous value between 0 and 1. This feature allows applications to use gestures to control fine tuning tasks as well as anticipating which gestures are ongoing allowing faster feedback, this way, covering both discrete and continuous *Flow* as discussed on Section 3.2.2. Prepose also allows partial gestures, described for a particular part of the skeleton, which allows the user to move the other parts of his body freely or even execute another gesture with them.

Prepose can be presented as a Domain-Specific Language (DSL) to write gestures, and a runtime recognizer to interpret the same written gestures. Towards the goals of easiness of use and prototyping, Prepose allows developers to write high-level natural language gesture descriptions, which have semantics translated in terms of SMT formulas. According to our knowledge, Prepose is the first rule-based solution with a base on a constraint solver, capable of translating gestures into expressions which can be later statically analyzed to check for *safety, validity, and conflicts*.

Required processing times for both runtime matching in Prepose, as well as static *conflict* checking, are applicable in real scenarios. Our Z3-based approach has more than acceptable performance. Pose matching in Prepose averages 4 ms. Synthesizing target *poses* (performed once a *pose* is reached, and a new *pose* is required) ranges between 78 and 108 ms. Safety checking is under 0.5 seconds per gesture. The average *validity* checking time is 188.63 ms. For 90% of the cases, the *conflict* detection time is below 0.170 seconds, and 97% of cases less than 5 seconds, with only one query taking more than 15 seconds.

The training based option on Prepose is available by the Prepose Recorder Tool. Following the easiness of use and prototyping goals, it is also based on a one-shot learning procedure, meaning a single execution is required for training. Thus, presenting the advantage to be less time consuming and to require less effort on the training process, being suitable for rapid high-fidelity prototyping of gestural interfaces. The tool allows the recording of gestures using a single sample execution. As output, the tool automatically writes the performed gesture description using Prepose language. Therefore, all the analysis and recognition properties available for human-created gestures on Prepose are also available to its training-based solution. Additional detail of Prepose, including a particular viewpoint of its value to the *Privacy and Security* field, can be found in the paper entitled “Prepose: Privacy, Security, and Reliability for Gesture-Based Programming” (FIGUEIREDO et al., 2016).

5

Pilot

On the previous Chapters, we introduced our proposed toolset, detailing the techniques and tools for each step of the design process. In this Chapter, we present a pilot study. In this pilot, we generate a high-fidelity prototype using our toolset, by requiring as input a set of target tasks. Our pilot study has two goals. The first is to demonstrate the utilization of the toolset on a straightforward and plausible scenario of development of in-air gestural interfaces. The second is to validate and evaluate the toolset as a solution to produce in-air gestural interfaces, analyzing its flaws and potentials.

As we discussed in the Chapter 2, the process of delivering a new solution can be diverse. On several processes descriptions, the initial phases are defined by terms such as Analysis, Understanding, Empathy and Definition. These early phases usually produce as output a defined problem that will be tackled. Similarly, our toolset requires a defined problem, which is written as the *challenge of producing high-fidelity prototypes for a particular target task*. For this pilot study, we performed these initial phases for the definition of the target activity (and tasks) through discussion and with little concern regarding the analysis structure. After considering a few alternatives and arguing about the advantages and disadvantages of each option regarding the pilot purposes, we defined our target activity and task.

5.1 Activity

On our pilot, we chose the activity of *presenting slides* as the target activity for the application of the toolset. Slides presentation are an everyday activity in several environments, ranging from TED talks (available at `www.ted.com`) to business meetings presentations, and lessons on colleges and universities.

Presentations are usually managed by using specific input interfaces. Examples are keyboard, e.g. for advancing slides; wired mouse, which also allows pointing and drawing on the slides; wireless mouse or presenter, that give freedom to the user while presenting on stage; and laser pointer, to call attention to specific parts of the slides. There are also cases in which the user does not have the direct control of the presentation. On these cases, the user asks a second

person to operate mouse and keyboard. Then, the operator should wait and perform the presenter commands for tasks such as advancing the slides. These commands are usually given through voice or gestures.

This way, an in-air gestural interface may present some advantages for the user. While using gestures, the user has no need to stay near to a fixed keyboard or computer, or else to leave and repeatedly return to that position for each time she desires to give a command. Through gestures, while using vision-based solutions, the user can be free of holding or having any attached devices. It would also release the need for an additional person as the operator. Therefore, we consider the use of in-air gestures control as a potential promising option for the activity of presenting slides, and this way, a proper choice for the pilot study.

5.2 Task

Regardless of the type of presentation, there are some common tasks we surveyed for this activity. Being based on the set of tasks shown on our proposed tasks classification (see Figure 3.3), there are generic tasks previously identified and summarized, while some of these tasks are too specific and yet not classified:

- Generic tasks:
  - Next, to advance to the next slide.
  - Previous, to go back to the previous slide.
  - Home, to go or return to the cover slide.
  - Play/Resume, to start a video playback or a chain of animations.
  - Pause, to pause a video playback.
  - Stop/Reset, to stop a video playback and return the slides.
- Specific tasks:
  - Point (simulated laser pointer), to call attention to a part of the slide.
  - Draw (pen), to leave comments or stress, circulate a part of the slide.
  - Highlight (highlighter), to highlight part of the slide.
  - Erase (eraser), to erase a particular drawing or highlight on the slide.
  - Erase all ink, to collect such feedback properly all drawings and highlights.
  - Zoom in, to a simple in a particular part of the slide.
  - Zoom out, to zoom out seeing the entire slide once again.

For our pilot, we select the *Next*, and *Previous* tasks, since they are the core of slides presentations. By using these tasks it is possible to run through the entire presentation, from start to end. Also, playing videos and animations can also be alternatively performed by these tasks.

For instance, the Microsoft PowerPoint presentation tool allows the use of the same command for both advance to the next slide, and play a video.

On this initial step of surveying and analyzing the activity tasks we identified a first potential limitation of the toolset, which can be circumvented by readjusting it. Some of the presented tasks, such as *Point*, and *Draw*, require a precise spatial 2D positioning of a cursor on the screen. For these tasks, the low-fidelity prototyping step may be unfeasible. These are tasks with continuous *Flow* tasks, which for this case will likely require a dynamic gesture.

In general, the Wizards of the Wizard of Oz technique are not fit for such challenge, because the fine-tuning for this task is essential and it is hard for the Wizards to perform the fine-tuning visually, just by subjectively evaluating the intensity of the user gestures. Moreover, according to our knowledge, there is no low-fidelity prototyping technique able to handle such challenge. This way, the use of the toolset in its original form is not suitable for those tasks. Therefore, if such tasks represent the core of the target activity, or designers decide they want to deliver prototypes for the task, we suggest an adaptation of our toolset.

To handle the fine-tuning of continuous *Flow* interactions through dynamic gestures, we propose the designers add a high-fi prototype recognition solution for retrieving the gesture intensity. The high-fi prototype can be used to replace the Wizards entirely, or partially. Meaning, while the fine-tuning is accomplished by the recognition solution, the Wizards can still perform their role for the remaining tasks and gestures.

5.3 Conception

Given the target tasks, we conducted the *Conception* phase. On this phase, we considered as input sources the designers (in this case ourselves), the literature of in-air gestural interaction, and the Sci-Fi titles.

5.3.1 Designers as source

In total, a group of four collaborators participated in a brainstorming session to generate interaction concepts for the given tasks. All the collaborators were researchers on the HCI field and had background work exploring in-air gestural interfaces. For the sake of simplicity, on the *Conception* phase, we consider these collaborators as designers from this point.

On the brainstorming session, designers were asked to generate a coupled set of gestures for the tasks of advancing (*Next*) and going back (*Previous*) on the presentation. This way both actions are coupled, and it is more likely that they will share the same metaphor, providing a more intuitive interaction with the user. Additionally, designers were encouraged to demonstrate their suggested gestures, bringing to the brainstorming session elements from the bodystorming technique (HANINGTON; MARTIN, 2012).

Each gesture was described at the moment of the suggestion. The description was then

read aloud, and if any of the designers disagreed or did not understand part of it, they suggested alterations until a clear description was presented. As a result, designers proposed 22 sets of gestures (one for each task), which we added to the 1st Pool of gestures, containing the descriptions of the gesture. We stored the *1st Pool* on a collaborative spreadsheet (from Google Spreadsheets) so each designer could access the material for consultation.

5.3.2 Literature and Sci-Fi as source

We also collected a set of gestures from the literature using our literature catalog (available at `http://goo.gl/frpBpA`). On the filters section of the catalog, we selected the tasks *Next* and *Previous*. As a result, the catalog returned 49 entries. For each entry, we copied the gesture description from the catalog, producing the same output for entry from the brainstorming session, ending with a full description of its execution. Also, we reviewed each entry and coupled it in pairs in case two entries were from the same work and presented the similar metaphor for the target tasks (e.g. "Thumb up" triggers *Next*, and "Thumb down" triggers *Previous*). The result consisted of 25 sets of gestures, which we also added to the *1st Pool*.

We performed the same procedure on the Sci-Fi catalog (available at `http://goo.gl/XSX5fn`). On this catalog, we collected 4 entries which were grouped in 2 sets of gestures. We added both sets to the *1st Pool*, totalizing 49 sets of gestures.

5.3.2.1 Duplicates

Before selecting a subset of concepts for the prototype, we removed duplicates from the *1st Pool* of gestures. Duplicates can be found once two different entries, from the same or different sources, can describe the same gesture for the same task. For instance, we found 17 duplicates; the most typical case was the swipe gesture, which we found instances of from all the three sources (designers, literature, and Sci-Fi). After the removal, the *1st Pool* contained in total 32 sets of gestures for the given tasks.

5.3.3 Selection

Considering our previous experience using the Wizard of Oz technique (detailed on the low-fidelity Section of Chapter 4), we found to be not practical to prototype all 32 gesture sets. On our previous experiment for navigating in 3D virtual environments, each session with users took more than 40 minutes to end. On that experiment, we tested six different gesture sets, testing at a time three gestures per user. Therefore, for our pilot, we decided to repeat the same strategy.

This way we reduced the *1st Pool* to the six selected gesture sets. To perform the selection we used the Weighted Matrix technique (HANINGTON; MARTIN, 2012). We defined four criteria as relevant points to be desired on such interaction:

- Intuitiveness: intuitiveness is related to the gesture memorability if the user can use and reuse the gesture without much cognitive load.
- Effort: the effort describes if the continued use of the gesture through a long presentation (e.g. 1 hour) would bring fatigue.
- Social: the social aspect represents our notion of how much social discomfort the user would feel while using the proposed gestures on a real presentation with an audience. We considered the social aspect relevant because the activity of talking to an audience is directly related to the social relationship between the presenter and the audience.
- Ambiguity: our measure of ambiguity was if an observer[1] could distinguish well one gesture from another, as well as to identify and differentiate the controlling gestures from common speaker gestures, which are employed to give rhythm and emphasis to the presentation. We acknowledged ambiguity as a relevant class because we considered that the presenter should, as long as possible, feel free to gesticulate while presenting without worrying if any commands will be accidentally triggered. These accidents could occur either through recognition failures (false-positives), but also because the employed control gesture is ambiguous, and can be used in both cases, as a resource for rhetoric, but also as a control feature.

We assigned weights for each criterion as follows: Intuitiveness 8; Effort 6; Social 9; and Ambiguity 7. Then, each designer assigned scores (from 1 to 3, being three better than 1) for each criterion on each gesture set. We calculated the average rating for each category and then multiplied it by the established weight, adding the result to the total score. We then used the total score to rank the gesture sets, later selecting the six gesture sets on the top of the list as shown in Table 5.1, which represents the *1st Pool* of gestures produced by using our toolset.

5.3.3.1 Pools

During the *Conception* phase, we applied a sharp reduction of the *1st Pool*. We performed such procedure to select a reasonable amount of gestures for the *Prototype* phase. However, we perceived we were discarding a considerable amount of selected and analyzed data, which could be useful in a second iteration of the design process or future consults regarding the same tasks. Therefore, we stored a copy of a previous version of the *1st Pool*, from before the completion of the *Select* step, and while it still contained all the evaluation data and ranking. This way, storing this version of the *1st Pool* is one of the planned modifications for a future version of the toolset.

[1] In this case the observer can be depicted as an operator that is assigned to pass the slides for the presenter.

Table 5.1: *1st Pool* of gestures produced on the *Conception* phase.

Set ID	Gestures Description
1	Rotate clockwise 360° palm/wrist with index finger stretched pointing sidewards
1	Rotate counter-clockwise 360° palm/wrist with index finger stretched pointing sidewards
2	Swipe of open right palm to the left
2	Swipe of open right palm to the right
3	Touch mid phalange of index finger with thumb
3	Touch last phalange of index finger with thumb
4	Like scrolling on smartphones moving the right thumb up over the right index finger
4	Like scrolling on smartphones moving the right thumb down over the right index finger
5	Grab-n-drag slides to the left
5	Grab-n-drag slides to the right
6	With only the two fingers (index and middle) stretched, move the hand to the left
6	With only the two fingers (index and middle) stretched, move the hand to the right

5.4 Prototyping

5.4.1 Low-fidelity prototyping

Using the *1st Pool* of gestures as input we started the *Prototype* phase, commencing with the Wizard of Oz technique. As our first step, once we confirmed a user would be part of our experiment, we asked her to bring a Microsoft PowerPoint presentation containing at least 20 slides. Users either copied the presentation on a USB drive or emailed us with it attached. With this request, we intended to bring more realism to the experiment, as well as asking users to present a familiar content to them, instead of a generic and unknown content.

5.4.1.1 Setup

Figure 5.1 shows the setup of the experiment sessions. We used a projector to display the presentation, the Wizard was placed on the side of the presentation and used a notebook to both run the presentation and perform the commands. The user was free to position herself and face any desired direction.

5.4.1.2 Procedure

We started each session introducing the user to the purpose of the experiment and explained to them the role of the Wizard. Each user explored three different (and randomized) set of gestures from our *1st Pool*. We explained a set of gestures at a time, and on the explanation, we read the gestures description and demonstrated its execution. For each set, we asked users to present the current slides on a soft version of the original presentation. After some time of presentation, we then asked users to return a few slides to explain better a previous topic, so they would have to trigger the *Previous* task as well. The presentation time was about 3 to 5 minutes for each gesture set.

Figure 5.1: User performing a gesture to advance a slide.

After each performance users answered the SUS questionnaire (BROOKE et al., 1996), detailed on the the Section 2.4.2.4 of Chapter 2. In addition to the ten items presented on the SUS questionnaire, we introduced four items to assess the specific concerns about social discomfort and fatigue. We added these items as an experimental modification of the SUS questionnaire. Therefore, we also used the rule of balancing positive and negative items. Our introduced questions are:

- I think audience will naturally accept the use of these gestures to present.
- I think I'd feel strange using these gestures to make public presentations.
- I think that presenting using this gestural control is physically comfortable.
- I think I would feel fatigue by performing a full presentation using these gestures.

Additionally, users also had an open field on the questionnaire to leave any desired comments. Although we did not explicitly apply the Think-aloud Protocol (HANINGTON; MARTIN, 2012), we allowed users jump out of the role-play and share their experience during the presentations. We also recorded all sessions for registration and future video analysis.

5.4.1.3 Results

In total 16 users (13 male and 3 female) participated in our experiment, aged between 21 and 32 years old. Therefore, eight different users tested each gesture set. The test sessions had an average duration of 30 minutes. The most time-consuming part of the experiment was the answering of the questionnaires. Figure 5.2 shows the users comments on each gesture. We highlighted (in bold) some of the key issues and suggestions.

ID	Gesture	Comments
1	**Rotate** clockwise (next) or counter-clockwise (previous) 360° palm/wrist with index finger stretched pointing sidewards	The level of **physical comfort** with this option depends on the **precision of the recognizer**; Keeping the **arm up** can be a **nuisance**: This gesture was **less unnoticeable than expected**. I think it is more **invasive** and **harder to learn** if compared to other options.
2	**Swipe** of open right palm to the left (next) or to the right (previous)	I think that there should exist na option to use the **left hand**; For those who use primarily the **left hand**, this gesture is kind of **confusing**, in some moments it is hard to **remeber which gesture is used to advance**, and which is used to go back; ... Can be **confused** when you eventually needs to gesticulate or **deflect a mosquito** :)
3	**Touch** mid (next) or last (previous) phalange of index finger with thumb	I found this to be the **best gesture so far**. Maybe it is so small that it **may cause ambiguites** on the time that the system has to differentiate one gesture from another; Gesture **too small and complex for a detector** to perform the detection correctly. It is **comfortable and acceptable**, but it can be **hard to be recognized**; If it worked correctly it would be **the best**... on the presentation **the system failed** on several times.
4	Like **scrolling** on smartphones moving the right thumb up (next) or down (previous) over the right index finger	This gesture would be comfortable if I was **holding something** in my hand to **better slip**; Very **confusing**, there should be an option to **use the left hand**. In fact I found the gesture to be **confusing** regardless of which hand was being used. In several moments I **confused** the gesture to advance with the gesture to go back.
5	**Grab-n-drag** slides to the left (next) or to the right (previous)	This gesture presents variable usability **depending on which hand** is being used; This was the **worst gesture**, without a doubt. Very "**gross**". By a viewpoint it may be even better to make na **explicit gesture** so the **audience does not find it awkward**. But it is far **worse** regarding the caused **fatigue**.
6	With only the **two fingers** (index and middle) stretched, move the hand to the left (next) or to the right (previous)	The movement of the forearm is **unecessarily complicated** given that the same can be performed through a movement on the **wrist**; Sometimes it is **hard to remember** to use only **two fingers**, the tendency is to **use the open hand** to perform the gesture.

Figure 5.2: Some users comments for each gesture set.

Figure 5.3 shows a summary of users responses both to the SUS questionnaire and to our extended items for social and fatigue measurement. The averages presented on each item (e.g. Q1, Q2, Q3) are already pre-converted to be added to the final score. This way, each item ranges from 0 to 4, being 4 the more positive value. We then calculate the score by summing up the results from Q1 to Q10 and multiplying the result by 2.5. This calculation represents the official score proposed to be used in the SUS questionnaire.

We also use the adjectives associated with each score (explained on Table 2.2 on the Toolset Chapter) to add semantics to the resulting scores. The colored arrow on the bottom of Figure 5.3 represents all possible scores, from left to right, ranging from 0 to 100. Each adjective is placed on its respective position (as showed on Table 2.2). Then we plot the ID of each gesture on the same scale to infer on sight how well each gesture is ranked according to the SUS questionnaire.

ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Score	Q11	Q12	Q13	Q14	Extended
1	3	3	3.4	3.8	3.5	3.9	3.4	3	3.4	3.9	**85.3**	2.9	2.1	3	3.1	80.8
2	2.5	3.1	3.3	4.0	3	3.1	3.4	3.5	2.8	4.0	**81.6**	3.1	2.8	3.3	2.8	79.5
3	3.1	3.3	3	3.8	3.5	3.4	2.9	3.5	2.6	3.8	**81.9**	3.8	2.9	3.1	3.5	82.1
4	1.6	2.8	3	3.6	2.4	2.1	3.0	2.9	2.3	4.0	**69.1**	2.9	2.6	2.1	2.9	68.1
5	2.1	2.3	2.9	3.5	3.3	3.1	2.9	2.9	2.4	3.5	**71.9**	2.3	1.9	2.1	2.4	66.7
6	3.1	2.6	3.6	3.9	3.5	2.9	3.4	3.1	2.6	4.0	**81.9**	3.3	3.1	3.1	3.1	81

Figure 5.3: Average scores users gave for each gesture section on each questionnaire item.

At last, we also computed a similar extended score considering our added items. We

present the score on the column entitled “Extended” on Figure 5.3. The main change for computing our score is to, instead of multiplying the sum by 2.5 (used to make the maximum of 40 from the ten items scale up to 100), we multiply the sum by 1.78 (to make the maximum of 56 from the fourteen items scale up to 100). This modification was experimental and deviates in part from the original purpose of the SUS questionnaire. The SUS questionnaire does not claim to provide a diagnostic of the system; it rather provides a general measure of the user perception of usability. In fact, our added items are specific and do explicitly mention direct concerns about the target activity, which is presenting slides. Therefore, these items are not generic, and not necessarily desired as common goals of good usability. Nevertheless, we added this extended version to gather more insight and experiment on an extended version of the score.

5.4.1.4 Analysis

The 1st gesture set presented the best official score (85.3) on the experiment (Figure 5.3). In addition to being the best, which is a relative measure to the other options, it is also positioned as “Excellent” on the scale of adjectives. Therefore, we considered this gesture set as a valid option for the *Hi-fi* step. However, it is also important to notice that once we consider our added items concerning the social discomfort and fatigue, the score of the 1st gesture set decreased 5 points.

At last, on their comments some users revealed that concerns regarding the physical comfort on the gesture are linked to the recognition precision (Figure 5.3). Our understanding about this comment is that the gesture set has a potential to be physically comfortable if the recognizer can detect its execution even on subtle and smooth rotations. Users also reported the gesture as a physical *nuisance*, *less unnoticeable than expected*, *invasive* and comparatively *harder to learn*. Part of these comments is directly related to our concerns about the social and physical effort aspects of the gesture set, which may explain the low scores this option received in items Q11, Q12, Q13, and Q14.

The 2nd gesture set represents the swipe motion and one of the most found examples, on both literature and Sci-Fi, to handle the tasks of going to the *Next* or *Previous* content. The SUS score for this set was 81.6, being classified between “Good” and “Excellent” on the score scale. On their comments, users mentioned *“confusion”* in two different contexts. The first is about which hand and which direction to use for each task. The second is related to the ambiguity of the gesture, which can be falsely recognized simply because the user is using his body language as a communication resource while presenting, or “deflecting a mosquito”.

The 3rd gesture set received the score of 81.9, being the second best score of our experiment. This gesture set was also the only case that showed improvement on its score considering the extended items about social discomfort and fatigue. A user mentioned it as being the best gesture. There were also comments concerned about the difficulty of recognizing such gesture, and one user pointed out that the Wizard failed to recognize it several times. This gesture is

considered a micro-gesture since it requires only small motions of the palm and fingers, and this fact may explain the user's concerns and the Wizard failures. Therefore, we understand that the recognition failures impacted negatively on the official SUS score (which is directed to the system usability).

The 4th gesture set received the score of 69.1, which fits between "OK" and "Good" on the scores scale. On their comments, users pointed out this gesture as being confusing, mainly regarding the direction of the gesture. Similarly to the previous gesture set, this was also a micro-gesture. However, no comments were found regarding the recognition failures. Despite being a micro-gesture, its use is more explicit and visible; it requires motion (dynamic) and the 3rd gesture set required only a particular position (static).

The 5th gesture set scored 71.9 (which can be translated to "Good") on the SUS official questionnaire. Once added our extended items, the score decreased to 66.7, being the lowest score on our extended version of the questionnaire. Users mentioned this as being the "worst gesture", too "gross," and "worse regarding the caused fatigue."

The 6th gesture set scored 81.9, which is the second best score. Its score did not present significant decrease when considering our added items on the questionnaire. However, users mentioned it as a *unnecessarily complicated* gesture and suggested to use only the wrist movement instead of the entire forearm. The requirement of using only two fingers was also pointed out as an issue regarding its intuitiveness. We see the value of specifically requiring the use of two fingers because it reduces the gesture ambiguity, but as showed by users this also brings a drawback related to the gesture memorability and intuitiveness.

Decision Considering the exposed results and analysis we decided to include two gesture sets on the *2nd Pool*, being the 1st and 3rd gesture sets. We added the 1st because it was the most well evaluated gesture on the SUS questionnaire, and despite users concerns and its lowered score on our modified questionnaire, it remained with a good score (80.8). We added the 3rd gesture set because it was well evaluated despite the observed issues on the recognition by the Wizards. Once this issue can be tackled on the next step of high-fidelity prototyping, we considered this option as a promising one. All the remaining gestures achieved positive results, being classified as at least "OK" or "Good" solutions, and could be revisited if the selected gestures are not feasible to implement on high-fidelity prototypes.

5.4.1.5 Lessons

On this second experience of using the Wizard of Oz technique we acquired and matured some lessons for its use on our toolset.

Failures Users pointed recognition failures for the thumb-based gesture set (1st gesture set). We attribute this issue to the *Form* of the gesture. This gesture set requires only the use of the fingers on a static manner, and the remaining parts are unused. On our proposed code for describing forms (explained in Figure 3.4 on the Chapter 3) this gesture would be described by the following code **s–<O>—**. Thus, this is a very subtle gesture, which requires minimal motion, which the Wizard was not capable of fully recognize to a point users would not notice the failures.

This way, we understand it is not possible, despite our initial intent, to fully ignore technical limitations on the *Lo-fi* step, because the Wizards also present "technical limitations" as recognition solutions. Therefore, it is important for users to notice and point out such issues, so we can diagnose these technical difficulties and make decisions considering it. For instance, if the central concern of an explored option is a recognition limitation from the Wizards, it may be possible to overcome such limitation on a high-fidelity prototype, and therefore, that option should not be excluded based only that argument.

We also missed additional feedback from the Wizard. Currently, our toolset does not provide a means to collect such feedback. By understanding the Wizard difficulty, we could take additional measures. For example, the session setup could be tweaked, so the Wizard gets a better viewing of the performed gestures, adjusting the Wizard position or adding mirrors are some options. We can also add a second Wizard to perform the same task, this way it would take only one of the Wizards to trigger the command, and if both trigger the same command, we can ignore the second input by adding a *cooldown*[2] mechanism that ignores all inputs within a small amount of time (e.g. 0.5 seconds) after a successfully triggered input.

Audience While analyzing the recorded data, we perceived that the absence of an audience for the presenters triggered some unrealistic behaviors. In some cases, we found the users presenting to the wall on the opposite side of the slides as they were role-playing and pretended to face an audience. The most common cases were to present looking at the slides, or else looking at the Wizard. We understand that these behaviors draw back some of the intended realism of the experiment which is negative. Therefore, we consider on a next case to employ additional efforts to make the low-fidelity prototyping a more activity-focused experience, for example, by adding an audience (one or two people).

Focus Although users did not explicitly express that the Wizard divided their attention during the experiment while analyzing the recorded content we observed that some users repeatedly looked at the Wizards to signalize they were about to perform a gesture. This behavior gradually

[2]Cooldown is a concept usually applied on computer games. As described on `https://en.wiktionary.org/wiki/cooldown` *cooldown* is the "The minimum length of time that the player needs to wait after using an ability or item before it can be used again." Although the concept is directly applied to computer games, it can be extended to general interactions. For instance, by assuming that two Wizards represent a single *"player"* and their command triggers an *"ability"*, we can adapt the *cooldown* definition from computer games to our case.

faded out during the session duration. We understand this change as a result of users increasingly acquiring trust on the Wizard as a gesture recognition solution.

We did not observe this behavior in our first experiment for navigating in 3D scenarios. There, users had a goal to be achieved in the least possible time and had to keep focused on the scene due to the continuous *Flow* of the task.

Structure and time Despite providing an easy way of prototyping different gestures, our structure of combining Wizard of Oz followed by questionnaires on each gesture set showed to be time-consuming. We consider that the required time for each session plus the gap between one session and the other was not optimized. For reducing the gap between sessions, a possible solution is to have a member of the designer's team encumbered of scheduling the users.

For the time of each session, we noticed the most consuming part were the questionnaires. There are advantages but also some drawbacks against asking the users to stop experimenting and then answering the questionnaire (set of 14 closed items plus the open question). The main benefit was to gather feedback when the experience was still fresh in the user's mind since they just used that gesture. On the other hand, it also introduced a break in the session *Flow* and consuming time.

At last, our use of the Wizard of Oz technique restrained it in a particular manner. By fixing the set of gestures from the start, we prune its potential of rapidly modifying and adapting the solutions. During the experiment, some users provided insight of modifications on our proposed gestures, but the way we structured the sessions did not allow the test of such modifications. If we decided to use that feedback, while respecting the current toolset structure, the only option would be to save the suggested change for a second iteration of the process.

5.4.2 High-fidelity prototyping

After the low-fidelity prototyping and test we added the 1st and 3rd gesture sets to the *2nd Pool*, of gestures. Using these gestures, we started the *Hi-fi* step.

5.4.2.1 Available sensors

Our first concern was to explore and evaluate which sensors we had available to create the high-fidelity prototypes. The sensors we had in hand were the Microsoft Kinect V2, the Leap Motion, and the Intel RealSense F200. All three sensors are supported by official SDKs, which provide skeleton data about some body parts. The Kinect SDK retrieves 25 joints of the user body in real-time, it is a *mid-range* sensor that works at its best when the user is at least half a meter away from it, and at most 4.5 meters. The Leap Motion SDK is a *short-range* sensor, capable of tracking both user hands and forearms once they are within a hemispherical volume of, at most, 60 centimeters of distance from the sensor. The Intel RealSense F200 is

also a *short-range* sensor, capable of retrieving a skeleton of the user's hands from at most 60 centimeters away from the sensor.

This way, the supported range of each sensor already gives us a good idea of which are easier to be applied for the activity of presenting slides. Nevertheless, we wanted to explore the tracking of the available sensors on some gesture simulations within their allowed range. The Kinect does not entirely track user's fingers, therefore, in case we would find out that one of the short-range sensors (Leap or F200) could handle well the user gestures we could employ our efforts on adapting the presentation setup so the short-range sensor could be used.

However, on our pre-test the Microsoft Kinect V2 showed to be considerably more robust, maintaining the tracking even after rapid motions and rapidly recovering after self-occlusions (part of the user body becomes hidden by another part, e.g. arm behind chest). On the other sensors (Leap or F200) we tested our thumb-based gesture set (3rd gesture set) to check if they could provide a robust tracking for it. Both Leap and F200 commonly lost the tracking and were not always able to recover it in an available time, this behavior on the presentation could affect the entire performance of the user.

Therefore, despite its limitations of not entirely tracking the user's hands, we chose the Kinect as the sensor for the high-fidelity prototypes. This choice also automatically discards the *thumb-based* gesture set, given the Kinect cannot retrieve accurate tracking information about the user's thumb and index finger (parts used on the gesture).

5.4.2.2 Prepose

As a result of the pre-test of our available sensors, the 1st gesture set was the remaining gesture pair to be prototyped using Prepose. This gesture is described as follows: "Rotate clockwise (next) or counter-clockwise (previous) 360º palm/wrist with index finger stretched pointing sidewards". Considering the gesture definition, we firstly observed that despite Prepose offers a *Rotate action*, it does not natively handle full circles as the one described for the gesture. Therefore, we would have to write the gesture in terms Prepose already understands, using the available *actions* to write one *pose* for some different stages of the circle.

Training This way, our first step to build the recognizer was to run the Prepose (shown in figures 4.14 and 4.15 of the Chapter 4) recorder to automatically write an initial *pose* for the *Next* gesture, and the *Previous* gesture. We selected three different *actions* (*Put*, *Align* and *Touch*) and six joints (*spine mid, right shoulder, right elbow, right wrist, right hand* and *right hip*) for the recorder. After one execution, the recorder generated the *pose* called *"wrist_arc_down"*, shown in Figure 5.4.

While observing our recordings from the Wizard of Oz experiment and after simulating ourselves the use of the gesture, we noticed that the start point (or *pose*) for the gesture to go back a slide (*Previous*) is slightly different from the start point to advance a slide (*Next*). This

```
GESTURE next:
    POSE wrist_arc_down:
    put your right elbow to the right of your spine mid,
    put your right elbow to the right of your right shoulder
    put your right elbow below your right shoulder,
    put your right wrist to the right of your spine mid,
    put your right wrist below your spine mid,
    put your right wrist in front of your spine mid,
    put your right wrist below your right shoulder,
    put your right wrist in front of your right shoulder,
    put your right wrist to the left of your right elbow,
    put your right wrist below your right elbow,
    put your right wrist in front of your right elbow,
    put your right hand below your spine mid,
    put your right hand in front of your spine mid,
    put your right hand below your right shoulder,
    put your right hand in front of your right shoulder,
    put your right hand to the left of your right elbow,
    put your right hand below your right elbow,
    put your right hand in front of your right elbow,
    put your right hand to the left of your right wrist,
    put your right hand below your right wrist,
    put your right hip below your spine mid,
    put your right hip to the left of your right shoulder,
    put your right hip below your right shoulder,
    put your right hip to the left of your right elbow,
    put your right hip below your right elbow,
    put your right hip below your right wrist,
    put your right hip behind your right wrist,
    put your right hip below your right hand,
    put your right hip behind your right hand,
    don't align your right shoulder and your spine mid,
    don't align your right elbow and your spine mid,
    don't align your right elbow and your right shoulder,
    don't align your right wrist and your spine mid,
    don't align your right wrist and your right shoulder,
    don't align your right wrist and your right elbow,
    don't align your right hand and your spine mid,
    don't align your right hand and your right shoulder,
    don't align your right hand and your right elbow,
    don't align your right hip and your spine mid,
    don't align your right hip and your right elbow,
    don't align your right hip and your right wrist,
    don't align your right hip and your right hand,
    don't touch your spine mid with your left hand,
    don't touch your spine mid with your right hand,
    don't touch your right shoulder with your left hand,
    don't touch your right shoulder with your right hand,
    don't touch your right elbow with your left hand,
    don't touch your right wrist with your left hand,
    don't touch your right hand with your left hand,
    don't touch your right hip with your left hand.

    POSE to_up:
    rotate your right wrist 15 degrees up.

    POSE to_front:
    rotate your right wrist 15 degrees to your front.

    EXECUTION:
    wrist_arc_down,
    to_up,
    to_front,
    wrist_arc_down.
```

Figure 5.4: Resulting gesture description on Prepose for the *Next* task.

way, we also used the Prepose Recorder Tool to automatically write the first *pose* of the gesture for the *Previous*, which is called *"wrist_down"*, as shown in Figure 5.5.

```
GESTURE previous:
    POSE wrist_down:
    put your right elbow to the right of your spine mid,
    put your right elbow to the right of your right shoulder,
    put your right elbow below your right shoulder,
    put your right wrist to the right of your spine mid,
    put your right wrist below your spine mid,
    put your right wrist in front of your spine mid,
    put your right wrist below your right shoulder,
    put your right wrist in front of your right shoulder,
    put your right wrist to the left of your right elbow,
    put your right wrist below your right elbow,
    put your right wrist in front of your right elbow,
    put your right hand below your spine mid,
    put your right hand in front of your spine mid,
    put your right hand below your right shoulder,
    put your right hand in front of your right shoulder,
    put your right hand to the left of your right elbow,
    put your right hand below your right elbow,
    put your right hand in front of your right elbow,
    put your right hand to the left of your right wrist,
    put your right hand below your right wrist,
    put your right hip below your spine mid,
    put your right hip to the left of your right shoulder,
    put your right hip below your right shoulder,
    put your right hip to the left of your right elbow,
    put your right hip below your right elbow,
    put your right hip below your right wrist,
    put your right hip behind your right wrist,
    put your right hip below your right hand,
    put your right hip behind your right hand.

    POSE slightly_front:
    rotate your right wrist 5 degrees to your front.

    POSE to_back:
    rotate your right wrist 10 degrees to your back.

    EXECUTION:
    slightly_front,
    to_up,
    to_back,
    wrist_down.
```

Figure 5.5: Resulting gesture description on Prepose for the *Previous* task.

Presentation Despite the facilities of handling the descriptions of the gesture in natural language, the amount of information automatically written (in particular on its current presentation), showed to be difficult to be tuned manually. We noticed that a considerable part of descriptions could be compressed through a better writing. For example, instead of using three different lines for saying that the elbow, the wrist, and the hand are below the shoulder, we can compress this

information in a single line. In fact, Prepose language already supports such compression, but our recorder does not use it to its full capacity yet. Therefore, to keep the complete gesture description as a manageable amount of text, we maintained the start *pose* as the one fully described by the automatic writer and wrote ourselves the remaining *poses*.

Writing The start *pose* gives us a solid set of restrictions. This set guarantees the gesture will not be triggered randomly. This way, we wrote the remaining *poses* as simple rotations (e.g. "to_up" and "to_front"), in a way that with a few *poses* we would require the user to start from a *pose*, define a rotation direction and then return to the start *pose*.

Refining After a few experimentations we noticed that the start *pose* of the gesture *Previous* could come from different positions, so we removed the trained start *pose* from the first *execution step* and maintained it as the last step for the gesture completion. We also tuned the rotation angles for the transition *poses* "to_up", "to_front" and "to_back". We also added the *pose* "slightly_front" on the *Previous* gesture because we observed that while preparing to perform the rotation users usually open a little of space rotating their arm to their front.

Events Additionally, we assigned one event for each gesture using the Prepose IDE. The *Next* gesture triggers the *Page Down* key event, and the *Previous* gesture triggers the *Page Up* event. This way our prototype not only recognizes the gesture but also allows the user to advance and go back on the slides using in-air gestures. We saved the gestures description file, and the events file using the same name, this way, once the user opens one of these files the other is automatically loaded and can be played to run the high-fidelity prototype.

5.4.2.3 Results

To present our recognition results, we recorded a small set of repetitions of the selected gestures. In total, we recorded 19 repetitions of the *Next* gesture and 7 repetitions of the *Previous* gesture. On each repetition, we introduced small variations of the relative position and orientation between user and sensor, and varied the speed of the gestures execution. As preliminary results, we obtained 88% of correctly classified inputs, and with 0 false-positives, which means it is unlikely that the gesture will be triggered without the user intention and control.

Given the recognition results, we consider that our high-fidelity prototype is a positive output of our toolset and can be used as a start point to deploying and refining the final solution. We also added the 1st gesture set to the *3rd Pool*, storing it as a proposed solution that went through the entire process and was implemented as a high-fidelity prototype.

5.4.2.4 Lessons

While using Prepose to produce the high-fidelity prototypes, we noticed spaces for improvement on the tool.

Actions As the first point, we missed a specific *action* to handle cyclic gestures (as the one we prototyped). A *Spin action* could specify how many cycles the user performed with a specific joint would solve the needs of our interaction with a more compact and precise solution. The *Spin* is just one example of *action* to be introduced. We think that the current Prepose language has space to absorb a considerable larger amount of *actions*, increasing its power of description.

Compression The produced text by the Prepose Recorder Tool is useful, but also unnecessarily large. Once a single *pose* hits more than 20 lines of *actions*, it becomes hard to understand what exactly that *pose* is specifying. A proper text compression, using the full capacities of Prepose language, would significantly improve the experience of reading and editing recorded gestures.

Animation The textual description Prepose provides allows an easy way to read what each *pose* and gesture represent. However, in some cases visual cues are critical. Expressly for complex gestures, understanding the full gesture just by reading its description is a difficult task. Therefore, a useful improvement would be to have rendered skeletons for each *pose*, which could come from recordings but also inferred from the *pose* description. Associating each *pose* to a rendered skeleton would facilitate the editing process. This way, by changing one line of the description the skeleton can visually respond to the consequences of that change, and in the edition process, we would be able to anticipate if such change improves or not the gesture description.

Intelligence At last, we also noticed that the gestures descriptions can be automatically tuned through recording repetitions of the same gesture. Prepose Recorder Tool should be able to automatically take advantage of repeated gestures, and enhance its description by using the similarities between each repetition.

5.5 Conclusion

In this Chapter, we presented a pilot study using our toolset. On this pilot, we focused on producing high-fidelity prototypes for the activity of presenting slides. As one of its purposes, this pilot can be used for guidance on how to apply the toolset to future works. Moreover, we approached and discussed several key points of the toolset that were revealed on this pilot and may deserve attention for future work as they represent room for improvement.

6

Conclusion

This thesis addresses the process and the implied challenges of building in-air gestural interfaces. We tackle these challenges by creating an assembled set of tools and techniques (toolset) for building such interfaces while using the design process (detailed in Chapter 2). We detail the use of each technique and tools for each step of the design process, concatenating inputs and outputs for a fluid execution while using the toolset. As input, our toolset requires a target task, such as "apply zoom on a map" or "turn on the TV." Once at some point of an ongoing process, designers[1] encumbered of developing a solution reach the conclusion they want to control such tasks with in-air gestures, they can use our toolset to do so.

Our toolset has two main goals. The first is to avoid the full implementation of bad choices (in usability terms) of gestures to control the target task. The second is to speed up the implementation of valid choices. We tackle these goals by proposing and providing a set of techniques and tools for the phases of conception and prototyping of the new gestural interface.

For the *Conception* phase, we proposed the use of different inspirational sources for the generation of the concepts. This way, we increase the diversity of the concepts favoring a larger exploration of possible solutions (detailed in Chapter 3). For this, we facilitate the secondary research for designers. We mapped and analyzed research (from the HCI field) and fictional (Sci-Fi) examples of gestural interactions, providing accessible catalogs as inspirational and representative content for the creation of new applications.

For the *Prototype* phase, we proposed the use of both low-fidelity and high-fidelity prototypes (detailed in Chapter 4). This way, designers can rapidly explore a larger set of concepts on the *Lo-fi* step, and then narrow down the possible solutions on the *Hi-fi* step. As an option for low-fidelity prototyping, we demonstrated the use of the Wizard of Oz technique, as a fast alternative to validate the defined gesture set. For the high-fidelity prototyping, we introduced Prepose as a solution for writing, training and recognizing gestures given an input skeleton. At last, we demonstrated the use of the toolset in our pilot study (in Chapter 5). We state each of our contributions on the Section 1.3 on the Introduction Chapter.

[1] For the sake of simplicity in this Chapter we refer to these stakeholders as *designers*, relating their application of the toolset as a role from the Design field.

6.1 Publications

Part of the work we describe in this thesis has been already shared with the research community, and with the general public for consultation and collaboration. Here we list all the published content related to this thesis:

- Published resources on web pages:
  - Summary of the proposed toolset for consultation and guidance: http://cin.ufpe.br/~lsf/thesis/deliverables/summary.pdf.
  - Public spreadsheet containing the mapping of literature papers: https://goo.gl/VY9uBY.
  - Web catalog for easy access and filtering of the gestures found in literature: http://cin.ufpe.br/~gesturescatalog/classification.
  - Public spreadsheet containing the mapping of Sci-Fi gestural interactions: https://goo.gl/tozbCS.
  - Web catalog for easy access and filtering of the gestures found in Sci-Fi titles: http://cin.ufpe.br/~gesturescatalog.
  - Open source repository containing the source code of the web catalogs: https://github.com/voxarlabs/gesturescatalog.
  - Open source repository containing the source code of underlying library used to translate the spreadsheet contents into filterable cards: https://github.com/voxarlabs/catalogs-api.
  - Open source repository containing the latest stable version of Prepose source code: https://github.com/Microsoft/prepose.
  - Open source repository containing the latest version of Prepose source code: https://github.com/lsfcin/prepose.
- Published work on research venues:
  - Work in progress published on the ACM CHI conference about the Sci-Fi catalog (FIGUEIREDO et al., 2015).
  - Full paper published on the INTERACT conference about the Sci-Fi catalog (FIGUEIREDO et al., 2015).
  - Full paper published on the HCII conference about the Wizard of Oz experiment for navigation (FIGUEIREDO et al., 2014).
  - Full paper published on the SVR conference about the Checkpoints (CHAVES et al., 2012).
  - Technical report published about the Prepose tool (FIGUEIREDO et al., 2014).

- Full paper published on the IEEE S&P conference about the Prepose (FIGUEIREDO et al., 2016).
- Full paper to be published on the IEEE S&P journal about the Prepose (FIGUEIREDO et al., 2017).

6.2 Derived results and additional uses

Along with the presented toolset, part of our work can be extended for additional uses that are unrelated to the goal of building in-air gestural interfaces. For instance, the mappings of the use of in-air gestures can be utilized for both a separate and relational analysis of these interactions on HCI research and Sci-Fi industry. We start this discussion on the paper entitled "Sci-Fi Gestures Catalog" (FIGUEIREDO et al., 2015).

We also address additional concerns of Prepose regarding the research field of Security and Privacy (FIGUEIREDO et al., 2016). From that viewpoint, we propose that Prepose can be used as a trust barrier, preventing malicious applications from acquiring streamed data (e.g. color, depth or skeletons) from always-on sensors commonly placed on places which potentially may reveal sensitive information (e.g. Microsoft Kinect devices placed on living rooms). This way, applications would only use gesture descriptions as requisition to Prepose, which in addition to performing the static analysis (to detect potential *safety* issues) would be capable of output solely the gesture recognition result. A full presentation regarding this topic is available at `https://youtu.be/i1TuE4UG9pM`.

There are also additional uses to Prepose, in addition to recognizing gestures for active control, it can also detect and analyze the user movements and posture. Prepose has the bonus of providing semantics about the missed points of the user performance on a described gesture. Therefore it can be employed for an analysis of what a user is doing wrong while performing a particular gesture. It also supports full body gestures, extending its use to recognize body postures from domains such as physiotherapy and martial arts. We are currently developing a reviewed version of Prepose called Corpora, which shows some of these capabilities (viewable in `https://www.youtube.com/watch?v=MyWQkJR-tv0`).

6.3 Future work

6.3.1 Workshop

Our first future work is to conduct a comprehensive evaluation regarding the use of our toolset by designers. To perform such evaluation, we intend to promote a design workshop (detailed by Martin and Hanington (HANINGTON; MARTIN, 2012)), providing several design teams with a challenge of producing high-fidelity prototypes of in-air gestural interfaces, but also exploring different alternatives to deliver a quality prototype. Some of the teams would receive

no instructions on how to conduct the challenge. For other teams, we will ask the designers to use an already available tool to build the high-fidelity prototypes (e.g. the Microsoft Visual Gesture Builder for the Kinect V2). The remaining teams will use our toolset. As objective goals, we will evaluate the success of each team resultant interface regarding the perceived usability by users, among other additional criteria which the team considered as relevant for their concept. In addition to video recordings, we will collect feedback through observation, and questionnaires and interviews. Such evaluation may provide us with additional insights about our toolset strengths and weaknesses.

6.3.2 Adaptability

In its current version, our toolset allows designers to explore different sources for conception, and then to narrow down the solutions through prototyping. However, as an already noticed drawback, we observed that our toolset (and the way we fit it within the design process) does not allow the adaptability and evolution of the interaction concepts as solutions. Currently, the only way a solution can be adapted is by rewriting it as a new entry on a second iteration of the process. Therefore, we envision a second version of the toolset to handle the adaptability of the initially conceived solutions, mainly by using the user's feedback.

In addition to the always forward flow of processes, Hugh Dubberly (2005) points the feedback option, adding a cyclic nature to the flow. As described by Dubberly "some of the output signals is split off and 'fed back' into the input signal." The feedback property allows iterations to continuously refine a solution. Following this principle, we propose the modification of our toolset to make it support fast and small cycles of conception, prototyping, and test.

This concept is backed by a technique Rapid Iterative Test and Evaluation (RITE) (HANINGTON; MARTIN, 2012). RITE is a technique that brings developers close to the usability experiments, allowing them to see the system failures and issues rapidly, and then correct or tune it between one user and another. In our case, having developers in the team is not a requirement, because both of our proposed prototyping strategies, Wizard of Oz and Prepose, do not demand programming skills.

Additionally, we propose the use of Participatory Desing (HANINGTON; MARTIN, 2012), including users as a part of the team and asking them to help on the conception of the system actively. For this, we propose the use of the Think Aloud Protocol on the prototyping tests, and at each new insight, the team of designers and users can modify the gesture. Given that adapting a solution for the Wizards or for Prepose requires a small amount of time, it is likely that the same user that made the suggestions in the first place can explore the changes on the system. On RITE definition, these changes are made between the sessions from one user to the other. This way, we would bring together the concepts of *prototyping* and *gesture elicitation* (or *user-defined gestures*).

At last, as an initial step to bring together the experiment setup and to remove major

issues, we propose that the designers perform an initial set of iterations to evaluate their solutions. This step represents the application of a technique called Heuristic Evaluation (HANINGTON; MARTIN, 2012), for which Martin and colleagues state that “An agreed-upon set of usability best practices can help detect usability problems before actual users are brought in to further evaluate an interface”. This way, by detecting gross failures in advance, designers do not only save time but also increase the chances of exploring better options with users.

6.3.3 Understanding phase

Currently our toolset does not tackle the Understanding phase of the design process, and we assume that designers already executed that phase and found out target tasks to be controlled through in-air gestures. As future work, we aim to add techniques and tools to comprise that phase within our toolset, and then aid designers while empathizing and defining the target problem, including the set of target tasks and success criteria which should be considered for the solution. On the Understanding phase the team should observe the target activity, analyze the environment restrictions and social constraints, and then define the target problem.

6.3.4 Tasks relationships

On the problem definition, it is also important to understand which set of tasks should be tackled and the relation between these tasks. For example, on our pilot we selected two target tasks (Next and Previous), and these tasks were intrinsically linked by opposite metaphors, meaning that one task accomplishes the exact opposite of the other. Another point is that, although the tasks are mirrored one from another regarding the expected outcome, the frequency of use of the Next task is considerably higher than the use of the Previous task, this occurs because ideally slides presentation do only need to move forward. Relationships such as linked metaphors and comparison between the frequency of use between the selected target tasks should be considered in advance in a practical manner on the Understanding and Conception phases. The relationship between tasks should be observed and synthesized on the Understanding phase and taken into consideration on the Conception phase while proposing gesture sets in a way that the intuitiveness and memorability of the used gestures does not become compromised once they are coupled together. This way we point as future work the analysis and synthesis of related tasks and therefore linked gestures or sequential gestures in order to explore gesture sets with a higher chance of success on the Prototyping phase.

6.3.5 Descriptions catalog

To boost the prototyping of already cataloged gestures (from HCI and Sci-Fi), we have the goal as future work to translate the already provided gesture description on each entry into Prepose code. The first impact of this exercise would be to find gaps in the power of

expressiveness of the Prepose language. This way we would be able to extend the DSL based on already used gestures on literature and Sci-Fi. Secondly, once the descriptions are translated, the gesture recognizers for each cataloged entry would be available to be instantly applied on high-fidelity prototypes.

6.3.6 Extended language

We also point as future work the goal of extending Prepose language in two aspects. The first aspect is to support, in addition to the shape, the recognition of the performance dynamics. This way, the recognizer would not only evaluate the user posture in time, but also how the posture changes regarding aspects such as velocity, acceleration, stillness, fluidity, smoothness, and so on. Current Prepose grammar (shown in Figure 4.6) already supports the description of some dynamic aspects. For instance, the symbol *motionConstraint* allows the specification of the velocity of the gesture. The other symbol *holdConstraint* specifies the amount of time a user must hold a particular *pose* once she reaches it. However, the current Prepose runtime does not translate these symbols to C# code yet, which gives no compilation errors but also represents no change in the recognition and analysis of the gesture.

Secondly, we propose as a future extension to add on Prepose language and runtime the support for hierarchical descriptions of gestures. This way, a gesture could be built on top of other two pre-written smaller gestures. This capability would allow the description of more complex gestures, sequences of gestures (e.g. exercise sequences), or even activities (e.g. the activity of watching movies can be written as the combination of being sited and looking forward). In addition, by allowing the combination of smaller gestures within other gestures we can provide different types of connectors. Gestures can be connected by the *AND* keyword meaning both gestures must be satisfied at once, or else by the *OR* connector which allows the user to choose different ways to accomplish the same gesture.

6.3.7 Visual feedback

Despite providing, on both Prepose IDE and Prepose Recorder Tool, a real-time visual feedback of the user skeleton, we do not provide visual feedback about the gesture description. This way, the only information the user and designer have about the gesture are its textual description. On simple gestures, textual descriptions may be even easier to be interpreted than the visual feedback of the user skeleton. However, for complex gestures, it is not trivial to interpret all the required aspects of the gesture description.

Therefore, we consider that providing both, textual and visual feedbacks of the gesture would enhance the understanding of both users and designers about what the gesture specifies. Given that Prepose uses Z3 underneath its written *rules*, it is capable of retrieving a sample skeleton that satisfies each *pose* of a gesture. This way, we intend to use that sample to provide the visual feedback. Then, after each change on the gesture description, the output skeleton can

be recalculated. We also intend to add a playback feature so that designers can go back and forth on the gesture *poses* and understand the required motion by watching the visual feedback of the gesture skeleton as an animation.

6.3.8 Improved training

At last, the automatic training provided by the Prepose Recorder Tool can be enhanced by making use of additionally recorded samples. Once a user or designer provide more than one sample of the same gesture, the recorder should be able to infer which are the similarities between these samples. This way, it could remove specific *actions* of each sample and maintain its similarities. This way, the automatic training would increase its *correctness* by opening its *coverage* (as discussed by Fothergill and colleagues (2012)).

On the other hand, by increasing the coverage, the recognizer can also be subject to an increased number of false-positives. This way, we also propose the option of adding negative examples of the gesture execution. Thus, the training tool can identify which *poses* should not trigger the gesture and correctly define the coverage boundaries.

References

AIGNER, R. et al. Understanding mid-air hand gestures: a study of human preferences in usage of gesture types for hci. **Microsoft Research TechReport MSR-TR-2012-111**, 2012.

ARICI, T. et al. Robust gesture recognition using feature pre-processing and weighted dynamic time warping. **Multimedia Tools and Applications**, volume 72, number 3, pages 3045–3062, 2014.

ASUS. **Xtion PRO LIVE**. URL: https://www.asus.com/3D-sensor/.

ASYMCO. **IBM and Apple**: catharsis. URL: http://www.asymco.com/2014/07/15/catharsis/.

AUMI, M. T. I.; KRATZ, S. AirAuth: evaluating in-air hand gestures for authentication. In: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION WITH MOBILE DEVICES & SERVICES. 2014. pages 309–318.

BANGOR, A.; KORTUM, P.; MILLER, J. Determining what individual SUS scores mean: adding an adjective rating scale. **Journal of usability studies**, volume 4, number 3, pages 114–123, 2009.

BEDREGAL, B. C.; COSTA, A. C.; DIMURO, G. P. Fuzzy rule-based hand gesture recognition. In: **Artificial Intelligence in Theory and Practice**. Springer, 2006. pages 285–294.

BHATTACHARYA, S.; CZEJDO, B.; PEREZ, N. Gesture classification with machine learning using kinect sensor data. In: EMERGING APPLICATIONS OF INFORMATION TECHNOLOGY (EAIT), 2012 THIRD INTERNATIONAL CONFERENCE ON. 2012. pages 348–351.

BIGDELOU, A. et al. Simultaneous categorical and spatio-temporal 3d gestures using kinect. In: D USER INTERFACES (3DUI), 2012 IEEE SYMPOSIUM ON, 3. 2012. pages 53–60.

BRAGDON, A. et al. Code space: touch+ air gesture hybrid interactions for supporting developer meetings. In: PROCEEDINGS OF THE ACM INTERNATIONAL CONFERENCE ON INTERACTIVE TABLETOPS AND SURFACES. 2011. pages 212–221.

BROOKE, J. et al. SUS-A quick and dirty usability scale. **Usability evaluation in industry**, volume 189, number 194, pages 4–7, 1996.

CALEGARIO, F. et al. A Method and Toolkit for Digital Musical Instruments: generating ideas and prototypes. **IEEE MultiMedia**, volume 24, number 1, pages 63–71, 2017.

CAUCHARD, J. R. et al. Drone & me: an exploration into natural human-drone interaction. In: PROCEEDINGS OF THE 2015 ACM INTERNATIONAL JOINT CONFERENCE ON PERVASIVE AND UBIQUITOUS COMPUTING. 2015. pages 361–365.

CELEBI, S. et al. Gesture Recognition using Skeleton Data with Weighted Dynamic Time Warping. In: VISAPP (1). 2013. pages 620–625.

CHAUDHRY, R. et al. Bio-inspired dynamic 3d discriminative skeletal features for human action recognition. In: COMPUTER VISION AND PATTERN RECOGNITION WORKSHOPS (CVPRW), 2013 IEEE CONFERENCE ON. 2013. pages 471–478.

CHAVES, T. et al. Human body motion and gestures recognition based on checkpoints. In: VIRTUAL AND AUGMENTED REALITY (SVR), 2012 14TH SYMPOSIUM ON. 2012. pages 271–278.

COMPUTING MACHINERY (ACM), A. for. **Computing Classification System (CCS)**. 2012.

DE MOURA, L.; BJØRNER, N. Z3: an efficient smt solver. In: **Tools and Algorithms for the Construction and Analysis of Systems**. Springer, 2008. pages 337–340.

DONG, H. et al. An Elicitation Study on Gesture Preferences and Memorability Toward a Practical Hand-Gesture Vocabulary for Smart Televisions. **IEEE Access**, volume 3, pages 543–555, 2015.

DUBBERLY, H. How do you design? **Dubberly Design Office**, 2005. URL: http://www.dubberly.com/articles/how-do-you-design.html.

ELIZABETH BEAR GREG BEAR, D. B.; KRESS, N. **Future Visions**: original science fiction inspired by microsoft. URL: http://www.amazon.com/Future-Visions-Original-Inspired-Microsoft-ebook/dp/B0182NCTWS/.

FAUGEROUX, R. et al. Simplified training for gesture recognition. In: GRAPHICS, PATTERNS AND IMAGES (SIBGRAPI), 2014 27TH SIBGRAPI CONFERENCE ON. 2014. pages 133–140.

FIGUEIREDO, L. S. et al. In-place natural and effortless navigation for large industrial scenarios. In: **Design, User Experience, and Usability. User Experience Design for Diverse Interaction Platforms and Environments**. Springer, 2014. pages 550–561.

FIGUEIREDO, L. S. et al. **PrePose**: security and privacy for gesture-based programming. 2014. (MSR-TR-2014-146).

FIGUEIREDO, L. S. et al. Sci-Fi Gestures Catalog. In: **Human-Computer Interaction–INTERACT 2015**. Springer, 2015. pages 395–411.

FIGUEIREDO, L. S. et al. An Open Catalog of Hand Gestures from Sci-Fi Movies. In: PROCEEDINGS OF THE 33RD ANNUAL ACM CONFERENCE EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS. 2015. pages 1319–1324.

FIGUEIREDO, L. S. et al. Prepose: privacy, security, and reliability for gesture-based programming. In: IEEE SYMPOSIUM ON SECURITY AND PRIVACY (SP), 2016. 2016. pages 122–137.

FIGUEIREDO, L. S. et al. Prepose: privacy, security, and reliability for gesture-based programming. **IEEE Security & Privacy Magazine**, 2017. Forthcoming.

FOK, K.-Y. et al. A Real-Time ASL Recognition System Using Leap Motion Sensors. In: CYBER-ENABLED DISTRIBUTED COMPUTING AND KNOWLEDGE DISCOVERY (CYBERC), 2015 INTERNATIONAL CONFERENCE ON. 2015. pages 411–414.

FOTHERGILL, S. et al. Instructing people for training gestural interactive systems. In: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. 2012. pages 1737–1746.

GAMA, A. D. et al. Guidance and movement correction based on therapeutics movements for motor rehabilitation support systems. In: VIRTUAL AND AUGMENTED REALITY (SVR), 2012 14TH SYMPOSIUM ON. 2012. pages 191–200.

GE, W.; COLLINS, R. T.; RUBACK, R. B. Vision-based analysis of small groups in pedestrian crowds. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, volume 34, number 5, pages 1003–1016, 2012.

GOOGLE. **Project Soli**. URL: https://www.google.com/atap/project-soli/.

GROUP, N. **How to Make a Cheap Multitouch Pad**. URL: http://nuigroup.com/forums/viewthread/1731/.

GROUP, N. **NUI Group Community Main Page**. URL: http://wiki.nuigroup.com/.

GU, Y. et al. Human gesture recognition through a kinect sensor. In: ROBOTICS AND BIOMIMETICS (ROBIO), 2012 IEEE INTERNATIONAL CONFERENCE ON. 2012. pages 1379–1384.

GUPTA, S. et al. Soundwave: using the doppler effect to sense gestures. In: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. 2012. pages 1911–1914.

HACHAJ, T.; OGIELA, M. R. Rule-based approach to recognizing human body poses and gestures in real time. **Multimedia Systems**, volume 20, number 1, pages 81–99, 2014.

HANINGTON, B.; MARTIN, B. **Universal methods of design**: 100 ways to research complex problems, develop innovative ideas, and design effective solutions. Rockport Publishers, 2012.

HENZE, N. et al. Free-hand gestures for music playback: deriving gestures with a user-centred process. In: PROCEEDINGS OF THE 9TH INTERNATIONAL CONFERENCE ON MOBILE AND UBIQUITOUS MULTIMEDIA. 2010. pages 16.

HÖYSNIEMI, J.; HÄMÄLÄINEN, P.; TURKKI, L. Wizard of Oz prototyping of computer vision based action games for children. In: PROCEEDINGS OF THE 2004 CONFERENCE ON INTERACTION DESIGN AND CHILDREN: BUILDING A COMMUNITY. 2004. pages 27–34.

IBAÑEZ, R. et al. Easy gesture recognition for Kinect. **Advances in Engineering Software**, volume 76, pages 171–180, 2014.

IGN. **Fighters Uncaged Review**. URL: http://www.ign.com/articles/2010/11/09/fighters-uncaged-review.

INTEL. **Intel® RealSense Camera (F200)**. URL: https://software.intel.com/en-us/blogs/2015/01/26/can-your-webcam-do-this.

INTEL. **SDK Intel® RealSense™**. URL: https://software.intel.com/en-us/intel-realsense-sdk/download.

JAKOB, N. Usability engineering. **Fremont, California: Morgan**, 1993.

KARLESKY, M.; MELCER, E.; ISBISTER, K. Open sesame: re-envisioning the design of a gesture-based access control system. In: CHI'13 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS. 2013. pages 1167–1172.

LAI, K.; KONRAD, J.; ISHWAR, P. A gesture-driven computer interface using Kinect. In: IMAGE ANALYSIS AND INTERPRETATION (SSIAI), 2012 IEEE SOUTHWEST SYMPOSIUM ON. 2012. pages 185–188.

LEAP. **Leap Motion SDK**. URL: https://developer.leapmotion.com/.

LEE, S.-S. et al. Towards more natural digital content manipulation via user freehand gestural interaction in a living room. In: PROCEEDINGS OF THE 2013 ACM INTERNATIONAL JOINT CONFERENCE ON PERVASIVE AND UBIQUITOUS COMPUTING. 2013. pages 617–626.

LLC, S. C. E. A. **Playstation Vita**. URL: https://www.playstation.com/en-us/explore/psvita/.

MARCUS, A. The Past 100 Years of the Future: hci and user-experience design in science-fiction movies and television. In: SIGGRAPH ASIA 2015 COURSES, New York, NY, USA. ACM, 2015. pages 15:1–15:26. (SA '15).

MARIN, G.; DOMINIO, F.; ZANUTTIGH, P. Hand gesture recognition with leap motion and kinect devices. In: IMAGE PROCESSING (ICIP), 2014 IEEE INTERNATIONAL CONFERENCE ON. 2014. pages 1565–1569.

MATTHEW ELLINGSEN EMILIE FETSCHER, E. S. **Design Thinking in a Day**. URL: http://designthinking.co.nz/design-thinking-in-a-day/.

MCNEILL, D. **Hand and mind**: what gestures reveal about thought. University of Chicago Press, 1992.

MICHIGAN 3D LAB, U. of. **Kinect - Navigation in a Virtual Reality CAVE**. URL: https://www.youtube.com/watch?v=XMl4Q2smpPk.

MICROSOFT, C. **Kinect for Xbox 360**. URL: http://www.xbox.com/en-US/xbox-360/accessories/kinect.

MICROSOFT, C. **Kinect for Xbox One**. URL: http://www.xbox.com/en-US/Xbox-One/accessories/kinect-for-xbox-one.

MICROSOFT, C. **Kinect for Windows SDK 2.0**. URL: https://www.microsoft.com/en-us/download/details.aspx?id=44561.

MIGNOT, C.; VALOT, C.; CARBONELL, N. An experimental study of future "natural" multimodal human-computer interaction. In: INTERACT'93 AND CHI'93 CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS. 1993. pages 67–68.

MIRANDA, L. et al. Real-time gesture recognition from depth data through key poses learning and decision forests. In: SIBGRAPI 2012 (XXV CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES), Ouro Preto, MG. IEEE, 2012. pages 268–275.

MIRANDA, L. et al. Online gesture recognition from pose kernel learning and decision forests. **Pattern Recognition Letters**, volume 39, pages 65–73, april 2014.

MOTION, L. **Leap Motion**. URL: https://www.leapmotion.com/.

NILSSON, N. C. et al. Tapping-In-Place: increasing the naturalness of immersive walking-in-place locomotion through novel gestural input. In: IEEE SYMPOSIUM ON 3D USER INTERFACES (3DUI), 2013. 2013. pages 31–38.

NILSSON, N. C. et al. Tapping-in-place: increasing the naturalness of immersive walking-in-place locomotion through novel gestural input. In: D USER INTERFACES (3DUI), 2013 IEEE SYMPOSIUM ON, 3. 2013. pages 31–38.

OTHMAN, N. Z. S. et al. Creating 3D/Mid-air gestures. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATICS: CONCEPTS, THEORY AND APPLICATION (ICAICTA), 2016. 2016. pages 1–6.

PAOLA ANTONELLI, N. B.; FINLAY, V. et al. **Twelve Tomorrows**. URL: http://www.technologyreview.com/twelvetomorrows/16/.

PATSADU, O.; NUKOOLKIT, C.; WATANAPA, B. Human gesture recognition using Kinect camera. In: COMPUTER SCIENCE AND SOFTWARE ENGINEERING (JCSSE), 2012 INTERNATIONAL JOINT CONFERENCE ON. 2012. pages 28–32.

PERCOLATE. **50 Charts on the Future of Marketing and Technology**. URL: http://pt.slideshare.net/HuyHungHng/50-chartsfuturemarketingtechnology.

PERUSQUÍA-HERNÁNDEZ, M. et al. User-centered design of a lamp customization tool. In: PROCEEDINGS OF THE 5TH AUGMENTED HUMAN INTERNATIONAL CONFERENCE. 2014. pages 36.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: 12TH INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING. 2008. volume 17, number 1, pages 1–10.

PITSIKALIS, V. et al. Multimodal Gesture Recognition via Multiple Hypotheses Rescoring. **Journal of Machine Learning Research**, volume 16, pages 255–284, 2015.

PIUMSOMBOON, T. et al. User-Defined Gestures for Augmented Reality. In: HUMAN-COMPUTER INTERACTION – INTERACT 2013: 14TH IFIP TC 13 INTERNATIONAL CONFERENCE, CAPE TOWN, SOUTH AFRICA, SEPTEMBER 2-6, 2013, PROCEEDINGS, PART II, Berlin, Heidelberg. Springer Berlin Heidelberg, 2013. pages 282–299.

PREIS, J. et al. Gait recognition with kinect. In: 1ST INTERNATIONAL WORKSHOP ON KINECT IN PERVASIVE COMPUTING. 2012.

PU, Q. et al. Whole-home gesture recognition using wireless signals. In: PROCEEDINGS OF THE 19TH ANNUAL INTERNATIONAL CONFERENCE ON MOBILE COMPUTING & NETWORKING. 2013. pages 27–38.

SCHEBESCH, A. Reading the body language of mankind's oldest figurines: an experimental approach. In: TRACING GESTURES: THE ART AND ARCHAEOLOGY OF BODILY COMMUNICATION. 2014.

SCHMIDT, T. et al. Real-Time Hand Gesture Recognition Based on Sparse Positional Data. In: X WORKSHOP DE VISãO COMPUTACIONAL (WVC). 2014. pages 243–248.

SCHMITZ, M.; ENDRES, C.; BUTZ, A. A survey of human-computer interaction design in science fiction movies. In: PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON INTELLIGENT TECHNOLOGIES FOR INTERACTIVE ENTERTAINMENT. 2008. pages 7.

SHARP, T. et al. Accurate, Robust, and Flexible Real-time Hand Tracking. In: CHI. 2015. volume 8.

SHEDROFF, N.; NOESSEL, C. **Make it so**: interaction design lessons from science fiction. rosenfeld media. 2012.

SHOTTON, J. et al. Real-time human pose recognition in parts from single depth images. **Communications of the ACM**, volume 56, number 1, pages 116–124, 2013.

SOMA, J. **TableTop**. URL: http://www.builtbybalance.com/Tabletop/.

SONG, Y. et al. A Kinect based gesture recognition algorithm using GMM and HMM. In: BIOMEDICAL ENGINEERING AND INFORMATICS (BMEI), 2013 6TH INTERNATIONAL CONFERENCE ON. 2013. pages 750–754.

SUAREZ, J.; MURPHY, R. R. Hand gesture recognition with depth images: a review. In: IEEE RO-MAN: THE 21ST IEEE INTERNATIONAL SYMPOSIUM ON ROBOT AND HUMAN INTERACTIVE COMMUNICATION, 2012. 2012. pages 411–417.

TULLOCH, A. I. et al. To boldly go where no volunteer has gone before: predicting volunteer activity to prioritize surveys at the landscape scale. **Diversity and Distributions**, volume 19, number 4, pages 465–480, 2013.

UBISOFT. **Fighters Uncaged**. URL: http://fighters-uncaged.uk.ubi.com/.

VALLI, A. **Notes on Natural Interaction**. 2005.

VAMSIKRISHNA, K.; DOGRA, D. P.; DESARKAR, M. S. Computer Vision Assisted Palm Rehabilitation With Supervised Learning. **IEEE Transactions on Biomedical Engineering**, 2015.

VATAVU, R.-D. User-defined Gestures for Free-hand TV Control. In: PROCEEDINGS OF THE 10TH EUROPEAN CONFERENCE ON INTERACTIVE TV AND VIDEO, New York, NY, USA. ACM, 2012. pages 45–48. (EuroiTV '12).

VEMULAPALLI, R.; ARRATE, F.; CHELLAPPA, R. Human action recognition by representing 3d skeletons as points in a lie group. In: COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2014 IEEE CONFERENCE ON. 2014. pages 588–595.

VIEIRA, A. W. et al. Distance matrices as invariant features for classifying MoCap data. In: ICPR 2012 (21ST INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION), Tsukuba Science City, Japan. IEEE, 2012. pages 2934–2937.

WALTER, R. et al. Cuenesics: using mid-air gestures to select items on interactive public displays. In: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION WITH MOBILE DEVICES & SERVICES. 2014. pages 299–308.

WANG, Q. et al. Dynamic gesture recognition using 3D trajectory. In: INFORMATION SCIENCE AND TECHNOLOGY (ICIST), 2014 4TH IEEE INTERNATIONAL CONFERENCE ON. 2014. pages 598–601.

WANG, Q. et al. Evaluation of pose tracking accuracy in the first and second generations of Microsoft Kinect. In: HEALTHCARE INFORMATICS (ICHI), 2015 INTERNATIONAL CONFERENCE ON. 2015. pages 380–389.

WEICHERT, F. et al. Analysis of the accuracy and robustness of the leap motion controller. **Sensors**, volume 13, number 5, pages 6380–6393, 2013.

WIGDOR, D.; WIXON, D. **Brave NUI world**: designing natural user interfaces for touch and gesture. Elsevier, 2011.

WOBBROCK, J. O.; MORRIS, M. R.; WILSON, A. D. User-defined gestures for surface computing. In: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS. 2009. pages 1083–1092.

WU, H.; WANG, J.; ZHANG, X. L. User-centered gesture development in TV viewing environment. **Multimedia Tools and Applications**, pages 1–28, 2014.

WU, J.; CHENG, J. Bayesian co-boosting for multi-modal gesture recognition. **The Journal of Machine Learning Research**, volume 15, number 1, pages 3013–3036, 2014.

ZHANG, Z. et al. A novel method for user-defined human posture recognition using Kinect. In: IMAGE AND SIGNAL PROCESSING (CISP), 2014 7TH INTERNATIONAL CONGRESS ON. 2014. pages 736–740.

ZHAO, W. et al. Rule based realtime motion assessment for rehabilitation exercises. In: COMPUTATIONAL INTELLIGENCE IN HEALTHCARE AND E-HEALTH (CICARE), 2014 IEEE SYMPOSIUM ON. 2014. pages 133–140.