DynaTouch: A Remote Ray-Casting Based Interface with Tactile Feedback

BY

ANTONIO FARALDO
B.S., Computer Engineering, Politecnico di Milano, Milan, 2009
M.S., Computer Engineering, Politecnico di Milano, Milan, 2012

THESIS

Submitted as partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Chicago, 2012

Chicago, Illinois

Defense Committee:

Prof. Robert V. Kenyon, Chair and Advisor
Prof. Jason Leigh
Prof. Pier Luca Lanzi, Politecnico di Milano
To my family,

who supported me each step of the way.
ACKNOWLEDGMENTS

I want to thank my thesis advisor, Prof. Robert V. Kenyon for all his advices and suggestions, supporting me during the preparation of this manuscript. He accompanied me from the very start of this invention with valuable opinions and discussions, indispensable for the successful completion of this work.

Additionally, I express my appreciation for Prof. Pier Luca Lanzi’s help regarding the organization and coordination of this work at Politecnico di Milano.

I also want to thank the Electronic Visualization Laboratory (EVL) for the space and the instrumentation I had access to, without which I would not have been able to conduct my experiments. In particular, I express my sincere gratitude to Lance Long for all the technical support and the assistance he provided me in building the initial set-up for this interface.

Furthermore, without my family, giving me courage and supporting my decisions all the way ever, I would never have been able to achieve such an accomplishment. Therefore, I am more than just grateful to them.

But I was not only supported by my advisor or my family. My friends were also a great help to me, always supporting me when I got stuck with a problem, encouraging me, when I could not find a solution or just stood by my side and helping me wherever they could. For all the support and help they provided, and the great times we had together, I want to express my heartfelt thanks.

AF
TABLE OF CONTENTS

CHAPTER        PAGE

1 INTRODUCTION ........................................... 1
1.1 Problem Statement .................................... 1
1.2 Summary ............................................. 6

2 RELATED WORKS .......................................... 8
2.1 Hand-Held ............................................ 9
2.2 Eye-Driven .......................................... 18
2.3 Body-Driven ......................................... 26
2.4 Ray-Casting .......................................... 37

3 PROPOSAL ................................................ 47
3.1 Component Description ............................... 52
3.2 Software Architecture ............................... 56

4 METHODOLOGY ........................................... 60
4.1 Models Identification ................................. 62
4.1.1 MSE Data Fitting ................................... 64
4.1.1.1 Line ............................................. 66
4.1.1.2 Plane ........................................... 68
4.2 Camera ............................................... 72
4.2.1 Depth Linearization ............................... 74
4.2.2 Pin-hole Model ................................... 75
4.2.3 Projection Inverse ................................. 78
4.2.4 Projection On A Plane ............................ 80
4.3 Calibration ........................................... 83
4.3.1 Overview .......................................... 83
4.3.2 Display Identification ............................ 84
4.3.3 Cameras Transformation Estimation ............... 92
4.3.3.1 Common Reference System Method ............. 92
4.3.3.2 Affine 3D Homography Method .................. 100
4.4 Finger Detection ...................................... 106
4.4.1 Overview .......................................... 107
4.4.2 Threshold Map ..................................... 109
4.4.3 Finger-tip Extraction ............................. 113
4.4.4 Dynamic Point Accumulator ....................... 116
4.4.4.1 Data Structure ................................. 117
4.4.4.2 Outliers ....................................... 119
### TABLE OF CONTENTS (continued)

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.3</td>
<td>Adaptive Filtering</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Points Accumulation</td>
</tr>
<tr>
<td>4.5</td>
<td>Head Tracking</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Cyclopean Center of Projection</td>
</tr>
<tr>
<td>4.6</td>
<td>View-port Mapping</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Cursor Super-sampling</td>
</tr>
<tr>
<td>5</td>
<td>PRELIMINARY STUDY</td>
</tr>
<tr>
<td>5.1</td>
<td>Experiment</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Inclusion Criteria</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Exclusion Criteria</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Task</td>
</tr>
<tr>
<td>5.2</td>
<td>Results</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Game Performances</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Movement Time</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Discussions</td>
</tr>
<tr>
<td>6</td>
<td>CONCLUSIONS</td>
</tr>
<tr>
<td>6.1</td>
<td>Future Works</td>
</tr>
</tbody>
</table>

CITED LITERATURE                  159

VITA                                163
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>FILTERED DATA</td>
<td>148</td>
</tr>
<tr>
<td>II</td>
<td>NON-FILTERED DATA</td>
<td>148</td>
</tr>
<tr>
<td>III</td>
<td>CONFIDENCE INTERVAL FOR MEAN - FILTERED DATA</td>
<td>150</td>
</tr>
<tr>
<td>IV</td>
<td>CONFIDENCE INTERVAL FOR MEAN - NON-FILTERED DATA</td>
<td>150</td>
</tr>
<tr>
<td>V</td>
<td>TUKEY-CRAMER CONFIDENCE INTERVALS</td>
<td>151</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>
## LIST OF FIGURES (continued)

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>Corner extension</td>
</tr>
<tr>
<td>40</td>
<td>Checker-board identification (color)</td>
</tr>
<tr>
<td>41</td>
<td>Checker-board identification (depth)</td>
</tr>
<tr>
<td>42</td>
<td>Built reference system</td>
</tr>
<tr>
<td>43</td>
<td>Homography transformation</td>
</tr>
<tr>
<td>44</td>
<td>Scale invariancy compensation</td>
</tr>
<tr>
<td>45</td>
<td>Finger interaction</td>
</tr>
<tr>
<td>46</td>
<td>Blob example</td>
</tr>
<tr>
<td>47</td>
<td>Space layering</td>
</tr>
<tr>
<td>48</td>
<td>Space layering - description</td>
</tr>
<tr>
<td>49</td>
<td>Ortho correction</td>
</tr>
<tr>
<td>50</td>
<td>Raw blob mask</td>
</tr>
<tr>
<td>51</td>
<td>Barycentric coordinates</td>
</tr>
<tr>
<td>52</td>
<td>Contours example</td>
</tr>
<tr>
<td>53</td>
<td>Circular buffer</td>
</tr>
<tr>
<td>54</td>
<td>Locality property</td>
</tr>
<tr>
<td>55</td>
<td>Outliers</td>
</tr>
<tr>
<td>56</td>
<td>Bell graph</td>
</tr>
<tr>
<td>57</td>
<td>Sample mean frequency domain</td>
</tr>
<tr>
<td>58</td>
<td>Linear and sinusoidal interpolation</td>
</tr>
<tr>
<td>59</td>
<td>Eye’s COP estimation</td>
</tr>
<tr>
<td>60</td>
<td>Data consumption</td>
</tr>
<tr>
<td>61</td>
<td>Binary split</td>
</tr>
<tr>
<td>62</td>
<td>Filling process</td>
</tr>
<tr>
<td>63</td>
<td>Game experiment</td>
</tr>
<tr>
<td>64</td>
<td>Performances</td>
</tr>
<tr>
<td>65</td>
<td>Movement time - mean (left) &amp; median (right)</td>
</tr>
<tr>
<td>66</td>
<td>Movement time - variance</td>
</tr>
<tr>
<td>67</td>
<td>Confidence intervals - filtered data</td>
</tr>
<tr>
<td>68</td>
<td>Confidence intervals - non-filtered data</td>
</tr>
</tbody>
</table>
SUMMARY

In this work we present a novel human-computer interface that dynamically maps the user’s view of a distant display screen onto a planar transparent surface that is within arm’s reach.

By mean of a color and depth camera, we are able to transform the planar surface into a touch sensitive working area, used to interact with the distant display. The surface acts as a touch-screen where the contact position corresponds to objects shown on the distant display along the user’s visual line of sight.

After a required calibration with an external display, the camera tracks the finger’s motion on or near the touch-screen, as well as the user’s head position. This combination allows the software-developed system to build the necessary correspondence between the touch-screen, the user’s line of sight and the external display, so that the user interacts based on the objects he sees through the planar surface.

This interface can be used effectively as a virtual touch surface for any external display, regardless of its size. This reveals to be particularly effective for high-resolution displays where, due to their large size, some areas are often not directly accessible. Besides, with our prototype we prove that it is possible to obtain reasonable results with cheap and widely available components, thus making it accessible also for non-professional users.
CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Advancements in technology are having a pervasive impact on society regardless of age, influencing people’s life styles in a unique and singular way.

The firm grasp that technology holds on our lives is particularly evidenced by the important role that it plays in our daily routines. A smart-phone, for instance, has become an asset of common use and people hardly leave home without carrying it. Likewise, the Internet is made available to more and more consumers that, moreover, cannot renounce to its many services, like the massive amount of free information or being able to shop on-line with just few clicks.

Besides being synonym of "anywhere" and "anytime", today’s technologies are also becoming synonym of "everyone". In fact, their fast growing pace is tightly accompanied by a substantial price lowering that makes them accessible and therefore attractive to an always larger number of people. In addition, they encounter fewer and fewer obstacles compared to the past in entering into people’s lives, especially because the current generations are much more proactive towards them.

In this thriving and rapidly mutating scenario, an important role is being assumed by a particular class of technologies, generally addressed with the name of "collaborative". These
technologies are created by design to embrace novel and often singular interaction paradigms, so to enable a cooperative working model where multiple people can work together.

Although the word "collaborative" does not represent a very common term among non-technical users, their presence can be observed almost everywhere. In fact, important instances of collaborative solutions can be found in both the software and the hardware world, sometimes even combined into one.

Many web solutions are an obvious instance of collaboration-enabled technologies. Online services, like "Google Documents" or "CollabEd", implement cooperative paradigms so to allow several people the sharing and editing of a same document. Many others instead are mostly meant to facilitate a coordinated task in a specific working field. In the programming development context, for instance, this is the case of many versioning and revision tools, like "SVN" or "Git".

Even though it may seem easier implementing a collaborative application from a software point of view, the whole situation changes when leveraged to a physical level. In the task of developing an interface, collaboration becomes synonym of handling the user’s "presence", meaning that a so-called collaborative solution should be able to enable a scalable multi-user physical interaction. Finding such a solution has always been a challenging problem, as it has been shown in (1),(2) and (3)

One of the most significative instance of collaboration-meant device is definitively represented by the "high-resolution display". These displays are defined as a cluster of many smaller displays, usually assembled into a large-sized grid configuration. Such a composition allows to
reach resolutions of even 100 million pixels, otherwise impossible to achieve by mean of a single display.

The availability of a large working area is the key component to enable the collaborative interaction, often implemented turning the display into a touch sensitive surface with the application of special touch-panels. The large size, in fact, allows multiple users to work either simultaneously or in parallel and without any compromise regarding the content’s quality, thanks to the high resolution.

As the literature shows, today high-resolution displays are receiving a lot of attentions from the research community. One of the institutions who has long studied these displays is the Electronic Visualization Laboratory (EVL) of the University of Illinois at Chicago. In their many works they have widely acknowledged this technology’s potential and, furthermore, explored new possible forms of interaction (4) (5). No one had ever proposed an effective solution to fully exploit the tiled displays’ hardware, until they designed and developed a middle-ware component, known by the name of "Scalable Adaptive Graphics Environment" (SAGE) (6). SAGE is a software that turns any display’s combination, driven by a computer’s cluster, into a seamless environment similar to that of a local desktop. Users can then resize and manipulate windows with all the benefits granted by the high-resolution.

All this interest lead to the inevitable consequence that high resolution displays are starting to be referred to as a "pervasive" phenomenon. They are rapidly penetrating in the academic and even beginning to appear in the most technologically advanced offices. Ultimately they will also arrive in the homes. A group of scientists even envision a near future where users
will collaborate and work together in rooms, where the walls will be entirely made of these ultra-resolution displays. As a matter of fact, the scientific community is the first concretely appreciating the benefits brought by high resolution. In fact, as today’s data are collected at a always higher level of quality, only a technology like high resolution displays is capable of performing a fair enough visualization. Furthermore, while increasing in term of size and resolution, they are becoming cheaper and cheaper every year. This factor, combined to the availability of off-the-shelf hardware and reliable open-source code, turns them into a very attractive product.

Of course the growth and popularity of these displays is naturally accompanied by a strong demand for more suitable interaction techniques, since the traditional approaches are showing to be ineffective.

For instance, a mouse device, that is probably the most commonly used nowadays, suffers from a problem called *mouse rowing* when used with high resolution displays (7). Moving a cursor from one edge of the display to the opposite forces the user to perform a series of repeated physical movements and therefore causes him a greater and undesirable hand strain. Although the research is advancing techniques that adopt, for instance, movement prediction to shorten this distance (8), the errors caused by the non-deterministic nature of the prediction are too costly for the users experience. Furthermore, operating on a large working space means a greater difficulty for the user in identifying and keeping track of cursor’s position. Especially in a collaborative scenario, where the display’s area is crowded with multiple mouse pointers, distinguishing a cursor from another becomes challenging.
Similarly, also a only close-in approach shows to be not a sufficient enough solution. Dealing with a large-sized display implies that some areas may not be easy to reach, because the screens height exceeds the average persons height or, in other cases, a direct-touch interaction is completely impossible because the screens location is inaccessible.

A close-in interaction makes sense only when aimed to a detailed level of manipulation of the visualized data. In many situations a user wants only a coarse-grained control over the display area, without losing the benefits granted from the high resolution, as being able to show a content to a large audience (9). Many tasks, like sliding pictures for instance, do not require a fine-grained control that. Moreover, might worsen the users interaction.

In this work we introduce an interface for a coarse-grained interaction with a large-sized display, adopting a distant interaction technique such as remote pointing. Remote pointing is the name given for a class of techniques that allows the user to interact with the display from an appreciable distance. Besides, the solution proposed does not sacrifice the natural and effective experience typical of a touch-screen device, as it will be described in detail in the next chapters.
1.2 Summary

In this paragraph the structure of this work is described. For each chapter a brief description about the content follows.

- **Chapter 2 - Related Works**

  In this chapter a small summary of the currently available solutions for a distant interaction with a high-resolution display is presented. The aim of this section is to show both the benefits and the drawbacks of each presented approach.

- **Chapter 3 - Proposal**

  This chapter provides a high level overview of our work, starting from the idea that is behind its origin. The general functioning is described, together with the whole software architecture.

- **Chapter 4 - Methodology**

  Methodology represents the core chapter of this work. The high level overview given in the previous chapter is here extended to the details. Our approach and techniques are fully described.

- **Chapter 5 - Preliminary Study**

  In this chapter the preliminary study we conducted to understand the suitability of our approach is discussed. 10 subjects performed a task where each participant had to beat a computer controlled cursor in reaching a randomly appearing target. The results of this experiment are described and discussed as well.
Chapter 6 - Conclusions

Finally, this chapter addresses the benefits brought by our approach compared to the traditional techniques. Moreover possible future developments are listed and discussed.
CHAPTER 2

RELATED WORKS

In the literature many solutions were proposed to allow a user to interact with content shown on a display screen. This chapter focuses particularly on those solutions which are or can be applied on a high-resolution large-sized display. Some of the most interesting approaches are summarized in the following sections, with an emphasis on their benefits and drawbacks.

- **Hand-Held**

  Any device which can be held and operated by mean of the hands, with the purpose of manipulating content shown on a display.

- **Eye-Driven**

  Solutions which make use of eye-tracking technology to turn the user’s gaze into a control input.

- **Body-Driven**

  Any approach which translates the natural movements of the user’s body into semantically meaningful actions to interact with a display.

- **Ray-Casting**

  Solutions based on ”ray-casting”, a technique where a virtual ray-line formed in the user’s working space is used to identify a location on the display.
2.1 Hand-Held

As the name suggests, a "hand-held" device is any physical device that can be easily operated using one or both hands with the purpose of interacting with virtual objects shown on a display screen.

Although many interesting solutions were proposed in the literature for a hand-held style interaction, the most popular device today is still the "computer mouse". A mouse is an interface that translates relatively the bi-dimensional position of a cursor pointer when moved on its supporting surface. The combination of a simple design and a fast learning curve for the user are the feats which allowed it to retain till now the title of undisputed ruler among the hand-held devices.

Only recently this unquestioned dominance began to show the first signs of change with the coming of the first high-resolution displays. These displays provide a very large working area, allowing resolutions as high as 100 million pixels. However, when a mouse pointer is used as a control input for such a display, a problem called "mouse rowing" arises. A mouse controller must balance its sensitivity against its gain. Therefore, moving a cursor from one edge of the display to the opposite forces the user to perform a series of repeated physical movements and therefore causes him a greater and undesirable hand strain.

Aware of this issue, many approaches were proposed by the scientific community in order to tackle this problem.

In (7) the authors introduced the idea of a mouse cursor that dynamically changes its size and speed. Accordingly, they named it dynamic size & speed cursor (DSS).
In most of the operating systems the mouse motion adopts a simple and fixed paradigm called "bimodal speed". When the mouse moves on its supporting surface, the physical speed is simply converted into an acceleration of the cursor speed, so to move it n-pixels per movement. Unfortunately, such a choice does not perform well on a high-resolution display, due to the presence of the previously described mouse rowing problem.

To overcome this limitation, the authors proposed an adaptive solution for the mouse cursor. When a faster and farther movement is required, the cursor acceleration is increased since the user needs only a coarse control over the pointer. Conversely, when the task demands a higher precision, the cursor returns back to the speed typical of the bimodal paradigm. In the same manner, the cursor size also changes adaptively. When the cursor speed is high, its size is increased with the purpose of enhancing the visibility and allowing an easier targeting of the displayed objects.

In the experimental evaluation the authors compared five techniques, three of which were different implementations of the DSS cursor.

- **High-Speed Cursor**

  This implementation was based on the idea that a faster and bigger cursor is always more helpful on a large display. Therefore, after several trial and error experiments, they set the speed and size respectively to 6x the normal acceleration and 400x400 pixels.

- **Manual Cursor**
In this implementation of the DSS cursor the authors chose to provide two options for the speed and the size, a fast-mode equivalent to the high-speed cursor and a normal mode equivalent to the control cursor. This approach was called manual because the user is required to press the space-bar on the keyboard to switch from one mode to the other.

- **Automatic Cursor**

  The automatic cursor provides the same dual speed and dual size mode of the manual cursor, but with an automatic switching mechanism. The mouse’s physical speed is sampled every 0.5 seconds to determine if the size and the speed of the cursor should be changed.

- **Standard Mouse Cursor**

  This represents the control element of the experiment. The mouse preserves its normal acceleration and size.

- **Benko’s Warping Cursor**

  In the original Benko’s warping technique (10), the user was allowed to warp the mouse position from one screen to another just by pressing a keyboard button. In this work Benko’s idea was extended allowing a warping also to the upper and lower monitor.

The experiment consisted of three different tasks, performed by 15 participants.

1. **Click Task**

   The subject had to click on a sequence of 24 randomly located squares.

2. **Drag & Drop Task**
The subject had to drag and drop a sequence of 24 randomly located squares, appearing one at the time.

3. Complex Drag & Drop Task

The subject had to drag and drop a sequence of 24 randomly located squares, but the shapes were presented all at once.

The statistical analysis showed that the high-speed cursor had the best time performance. However, most users preferred the manual cursor, since they felt more comfortable with the possibility to manually switch between the fast and slow mode.

Another interesting approach was proposed by P. Baudisch et al. in (11). In particular, they addressed the problem of being able to visually track a fast cursor moving across a large high-resolution display.

Figure 1: Complex drag & drop task
The authors proposed a technique called *high-density cursor*, based on a method named *super-sampling*. Super-sampling is an anti-aliasing technique, widely used in many different fields. For example, in computer graphics the unpleasant jagged effect caused by the rasterization with a low resolution can be reduced with the super-sampling. If extra information is available, it can be used to enhance the rendered result.

![Diagram](image)

**Figure 2: Gaps filling**
The authors noted that the number of cursor images that can be displayed on the screen exceeds the typical mouse sampling frequency. Therefore a super-sampling approach was used to fill the gaps present in the mouse motion trajectory. Specifically they opted for a Bezier curves based interpolation, mainly justified by the computational feasibility. They also discussed simpler approaches, like a linear interpolation. However, the latter was excluded because it would have led to a polygon-shaped appearance of trajectory.

This enhancement led to two major benefits:

- The visual continuity is improved and helps the user to track and extract the cursor position
- The increased cursor "density" makes much easier for a user to visually locate the cursor position

This technique was also compared to another called "continuous motion blur". In the motion blur the mouse cursor leaves a trail on its passage, thus improving the visual continuity. Although this technique is most effective at improving the traceability, the trail left on the mouse trajectory changes excessively the typical appearance of the cursor. Moreover, producing the necessary blur is a very computationally expensive procedure.

Another important feature provided by the high-density cursor is the "dynamic scaling". The cursor size is allowed to change according to the current speed. When the mouse exceeds a certain speed threshold, the size of the cursor is increased to facilitate its visual tracking.

As proof of validity of their approach, the authors performed a user study based on a Fitts’ Law task. The law states that the time required to reach a target from a reference position or a
previous target, is a logarithmic function of the distance when the target dimension is constant
and that it is also a logarithmic function of the target dimension when the distance is constant.

In this experiment they compared four different techniques, three of which were high-density
cursor implementations.

- **HD Conservative**

  This implementation uses a speed threshold of 35 pixels per frame to turn on the high
density mode. The cursor’s size does not change.

- **HD Triple Density**

  The HD Conservative implementation is the same, except for a different on-set threshold
  of 12 pixels per frame.

- **HD Plus Scaling**

  This implementation is the same as the HD Conservative with the addition that during
  a fast movement the cursor size can double.

- **Control Cursor**

  A regular mouse, included as a control element for the evaluation.

The results showed that all the high-density implementations performed faster than the
control cursor. In particular, the implementation with the scaling resulted about 7% faster in
acquiring a target than the others. Another interesting observation the authors pointed out is
that the participants were not able to notice the effect of the cursor interpolation, thus meaning
that this high-density technique could be introduced in the existing systems without any legacy issues.

In another work (12), instead, the authors adopted a different strategy to solve the mouse rowing problem. They made use of the head tracking. Their idea was to use the head position and orientation to "warp" the current mouse location on the monitor where the user turns its attention to and thus considerably reduce the distance required to reach a target.

The first contribution of this work is the development of an algorithm for a robust head tracking. The proposed solution allows a real-time performance of 30 frames per second, with an accuracy of 1 mm for the head position and 1 degree for the head rotation. In an initialization stage the face is detected by mean of the OKAO vision library's face detector. Once the face is identified, a set of feature points is extracted and their 3D location estimated using a stereo camera triangulation. At the end, the final pose is obtained as the result of a particle filter that compared the actual image frame with multiple hypotheses in a 6D space.

To avoid any interaction conflicts, only the head tracking is responsible for moving the cursor between the monitors. Conversely, the mouse is in charge of only moving the cursor within a single monitor area. After the pointer warps on a new monitor, its identification is facilitated by a flashing star around the apparition location. The star flashes five times in one second and a short beep feedback is emitted to indicate the occurred change of position.

An evaluation study was performed to test the performances of their approach. Eight different subjects were chosen to perform a task consisting of rearranging 26 randomly displayed
windows, each containing a letter of the alphabet, from left to right across the monitors. For each participant the task was repeated four times, two with the tracker and two without.

Although the statistical analysis on the mean distance showed a reduction of the 32% when the head tracker was used, the mean time of completion turned out being sensibly worse. To justify this result, the authors blamed the resulting awkward feeling from when the pointer warped between close monitors. Since the cursor does not warp to the expected position, the user feels disoriented and therefore a slight increment of time to reach the target occurs.

![Figure 3: Close monitors warp problem](image)

As a solution to this shortcoming, they suggested the introduction of an algorithm to detect and track the user’s fixations. With this choice it would be possible to warp the cursor directly to the desired location. On the other hand, many participants declared that the use of the proposed technique could improve with the practice and thus return better performances results.
2.2 Eye-Driven

An eye-driven technique represents any approach that uses gaze as a control input. In this section we are especially interested in discussing some studies focused on the control of a cursor on a display by mean of the eyes.

The human computer interaction community has shown a great interest towards these approaches, recognizing their potential as a source for a control input.

Apart from becoming conscious, no special efforts are required from the user, because moving the eyes is a task he already performs naturally. In fact, the user’s eyes already implicitly point to the area where he is currently turning his attention to.

Moreover, when the gaze is the only used input to move a cursor on the display, the user’s body becomes free from any kind of involvement. This feature turns out being very important especially in the case of severely disabled people who cannot use any other part of the body except the eyes.

In (13) L.E. Sibert and R.J.K. Jacob performed an interesting evaluation study to show the effectiveness of the gaze as a control input. Encouraged by the previous studies and by the naturalness typical of an eye-based approach, the authors attempted the use of a hardware eye-tracker to track the eyes’ fixations and subsequently use them to move a mouse pointer. The chosen tracker was an off-the-shelf solution based on the corneal reflection principle. The technique consists of illuminating the pupil and cornea so to produce a reflection that can be used to determine a coordinate location for the user’s visual line of gaze.
In this work they proposed an algorithm to identify the fixation events and store them into a queue data structure. Events like start, continuation, end of a fixation, raw eye position, a failure to locate the eye position are used to select objects shown on the display that are reasonably close to the fixation.

The authors performed two different experiments to prove that an eye-based solution was faster than using a mouse.

- **Normal Selection Task**

  The first experiment, called "circle task", involved each subject in a selection task. After a required calibration of the eye-tracker, every participant was asked to select a circle from a 3x4 grid as fast as possible. The target appeared in the same random sequence for each subject, where the first was used simply to locate the starting position.

- **Complex Selection Task**

  The task in the second experiment was to select a letter from the same grid of circles. For each trial the subject was told which letter to select by means of a pre-recorded audio sample, played through a speaker positioned to its right. If the choice was correct, the next letter was showed, otherwise, after 1250 milliseconds a tone indicating the selection of a wrong letter was played.

Initially 26 people were chosen to perform the experiments, however only with 16 did the tracker work as it was supposed to.
The results of this first experiment showed that performing a selection with the eye gaze was significantly faster than doing it with a normal mouse. With 503.7 milliseconds the average time was in fact faster in respect to the 931 milliseconds of the mouse. Similarly to the first experiment, even the second experiment’s statistical analysis confirmed the dominance of the gaze-based approach. The eye’s mean time was with 338 milliseconds faster, compared to the 1441.0 milliseconds of the mouse time.

While the previous work addressed only the time required to distinguish between objects the user is turning his attention to, many others focused on testing if an eye-driven solution can be also used in tasks where a finer control is required.
For instance, in (14) the authors wanted to compare two different eye-tracking solutions, an open source and a state-of-the-art commercial eye tracker. The main purpose of their work was to show if a low cost solution can be a good substitute to a high-end eye tracker.

A common criticism of eye-tracking technology is that today only very expensive solutions produce adequate results for precise tasks, like for example identifying which word of a text the user is fixating. The usual cost of such a state-of-the-art equipment is ranging between $20000 and $30000 and therefore can only be afforded by the professional industry. Motivated by this, the scientific community is spending its efforts to find low cost alternatives to the high-end solutions.

In their work, the authors chose as low cost alternative an open source tracking software named "ITU Gaze Tracker". This software allows the tracking of the pupil and corneal reflections produced with an infra-red illumination simply using a common web camera and off-the-shelf camcorders.

Already in other studies this solution demonstrated to have the potential of a commercial tracker. However its first implementation was excessively intrusive since it required a head-mouting set-up and the camera had to be very close to the user’s eye. Only after a software update in 2010, did the camera become usable from a distance by changing the lens from a wide angle to a narrow angle of 16 millimeters. Including the IR emitters and the tracker hardware setup, the cost for this open source solution is only $100.
The authors compared the open-source solution with a high-end remote tracker Tobii T60. This choice was driven by the fact that Tobii is the largest commercial eye-tracker manufacturer and the T60 is one of the most commonly used.

Nine subjects participated in the experiment, 7 male and 2 female, ranging from 16 to 36 years old. Both trackers were configured to move a mouse cursor on the display. Each participant of the experiment performed a task where he had to look at 16 targets displayed
singly on a 4x4 grid. For each trial an error measure was sampled at a frequency of 30Hz, excluding premature samples produced by the reaction delay. This error was computed as the mean of the Euclidean distances between the gaze position and the target.

After removing the outliers, an analysis of variance showed that the state-of-the-art solution is twice as accurate as the low-cost open source tracker. Although this result confirmed the dominance of the high-end solutions, the authors pointed out that the mean error of 59 pixels achieved with the low-cost tracker implies that these solutions are at least mature enough to be used.

While a considerable part of the research chose to study the gaze as the only control input, many other studies focused more on the use of the eyes as a support for more reliable techniques, like a traditional mouse cursor.

In (8) M. Kobayashi and T. Igarashi introduced a new approach for a mouse cursor called "ninja cursor". As for the hand-held mouse based devices, the main purpose of this study was to propose a solution to reduce the "mouse rowing" problem. The idea behind the ninja cursors was to distribute multiple pointers on the working area so to reduce the mean distance required to reach a target and thus the movement time. The user will visually engage the cursor which is the closest to the pointed target.

One of the issues reported in the first implementations of this technique was the cursor ambiguity, occurring when two or more pointers are active on different targets at the same time. The solution proposed by the authors to solve the ambiguity uses a waiting queue algorithm:
1. If multiple cursors are inside a target, the closest one to the center is the pointing cursor. If there is only one cursor, it is simply activated.

2. If a cursor is not active and it is going to move into a target, then it is appended to the queue. As long as the cursor is in the queue, it will never go inside the target.

3. When the cursor moves away from the target, it is removed from the queue.

4. When the cursor leaves the target that it is pointing, it becomes inactive. Then the first element in the queue becomes the active cursor.

Figure 6: **Ninja cursors disambiguation experiment task**
The experimental evaluation underlined a major problem in this approach. When the ambiguity occurs, the cursors in the waiting queue require additional mouse movements before becoming active. This waiting state slows down the interaction, confirmed by the participants who preferred to bypass the distracting target rather than becoming stuck waiting.

In (15) the authors presented a technique to improve the disambiguation using the gaze. An eye tracker is used to determine the current active cursor, simply looking at the target of interest.

In their experiment 13 volunteers were recruited, 7 female and 6 male. The screen area was divided into a regular grid so that each cell corresponded to one of the cursors. Each subject performed 10 trials where the task was to select some circles located in a random position.

The results showed that the selection times were affected by both the number of used cursors and the distance needed to reach a target. Generally the one cursor condition had the best performances, however as the distance to move became larger than the width of a monitor, the multiple cursors started to outperform the single cursor. However the general feeling of the participants was that the multiple-cursors mode was the fastest, even though the statistical analysis proved the contrary.
2.3 Body-Driven

One of the most important advantages of a high-resolution display is certainly the availability of a large working area, since it allows a collaborative interaction between users. Either turning the display into a touch-sensitive surface or simply using multiple input devices, such as mice controllers or remote pointers, the users can work simultaneously, independently or cooperating.

Mainly motivated by this feature, the research community has recently started exploring the applicability of interaction paradigms that can make use of the user’s body. These approaches are generally named as "body-driven", since the body acts as the main component which controls the input of the display.

Due to the novelty of these interfaces, the use that can be done of the body may differ greatly from one technique to another. Some of them take advantage of the body only partially, like for example when only part of the body’s torso influences the control. Others instead aim to exploit the body thoroughly, integrating every limb’s movement in the proposed interaction paradigm.

The body-driven solutions are often referred to with the expression "natural interfaces" since the user easily and "naturally" comes to understand their use. In fact, being his own body the input controller, the learning process required to master the new paradigm is insignificant, practically absent.

A noteworthy instance of a body-based approach is proposed in (16) by Booth et al. In this paper the authors describe "Shadow Reaching", a novel interaction paradigm that makes
use of the user’s shadow to interact with a large wall display. More precisely, the shadows are first perspective projected on the display surface to identify the interaction region and then meaningfully used as a control input.

Figure 7: **Shadow reaching concept**

A great advantage of using the shadows is that the user needs only to reposition his body to interact with a different area of the display. For example, even though there is a small price to pay in terms of resolution, the range of interaction can be easily altered by just moving closer or farther respect to the display.
In their work the authors propose three variants of the Shadow Reaching approach:

- **Single Point Interaction**

  In this implementation the role that is usually played by a mouse is instead given to the user’s shadow. A Polhemus tracker equipped with one button is held in the user’s hand and constantly tracked. The hand location returned from the tracker combined with the light source situated behind the user is used to determine the coordinate position of the hand on the display. In addition, the user’s real shadow is shown on the display to provide just a physical embodiment without any computational use.

- **Full-Body Interaction**

  This approach is slightly more complex than the previous one, since it adopts the shadow of the whole body. Although the possible options for an interaction which uses the entire body are various, the authors developed only a small demonstration where the user’s shadow physically interacts with bouncing balls. Instead, the light source is placed behind the screen and the shadows are captured using an infra-red camera located at the opposite side and extracted by mean of simple computer vision techniques. Differently from the single point interaction, this time only a virtual model of the user’s shadow location is rendered on the screen.

- **Magic Shadows**

  Magic Shadows are inspired by a technique named ”Magic Lenses”, introduced for the first time in (17). Magic lenses are movable see-through widgets used to filter and trans-
form on-screen data on their passage. Likewise, in this work each shadow defines uniquely the user’s data when it is projected on them. The only enforcement required for the correctness of the approach is that the shadows have never to intrude one into another. The solutions adopted are either restricting the user’s ability to edit data within a collaborator’s shadow or inhibiting completely any form of action.

In this work the authors performed only a preliminary study, confirming once again that body-based techniques are natural and immediate. As result from the every-day experience with his own shadow, each participant mastered very quickly the interface functioning. For instance, they naturally stepped back when they wanted to increase the shadow size, without any prior explanation.

A similar study, but with a broader focus, is performed in (18) by Kitamura et al. Driven as usual by the failure of the traditional techniques on very large displays, they propose an entire framework for body-centric interaction techniques. Their first contribution is the description of several design guidelines that should be applied to all the approaches based on the use of the user’s body:

1. **Bind the user’s personal space and extra-personal space**

   The brain builds a spatial representation of the body’s coordinate frame. The personal space is the space where the body resides, while the extra-personal space is that outside the user’s reach. Binding is the process in which the brain fuses one space with another, thus enriching the interaction technique. Several studies showed that using shadows to bind these two spaces works effectively.
2. **Do not rely on the visual feedback**

The human body can play an important role thanks to the "prioreception" mechanism, the use of the information coming from muscles, skin or joints. Studies showed that eye-free approaches are effective and therefore it is desirable to not rely on the visual feedback.

3. **Preserving the private space**

The private space is that space that a user does not want to share. Since a large wall display is a notorious collaborative environment, it is very important to preserve the user’s private space with techniques that, for example, allow to keep some distance between the users.

4. **Support body cues**

Also the use of an eye contact or a smile are very important in designing a body-driven interaction. They can be used to support and manage for example the body coordination.

The proposed system is divided into three main modules:

- **Sensing Module**

The sensing module is in charge of generating the body’s raw data. The body can be tracked using either a magnetic tracker or computer vision based tracker. The first option is not affected by occlusion problems. However it comes with a working range of only 2 meters, it needs a calibration and it has a very short battery life-span. The second instead
solves the issues typical of a magnetic tracker. But since it is performed tracking colored balls attached to the user’s joints, it suffers from the occlusion problem.

• **Modelling Module**

An inverse kinematic solver is used to estimate the user’s skeleton from the hands position and the shoulders’ position, along with the length and the rotations of the limbs. The display model is calibrated beforehand and it always maintains the same position. To generate the shadows, a virtual light model is located in the room for each user. Moreover, different light implementations are provided to implement the shadow interaction:

  – **User Following**

    The light is directly located behind the user at a fixed distance from the shoulders location.

  – **Orthographic**

    The light is behind the user but projected with an orthographic and not perspective transformation. An orthographic projection does not distort the user’s shadow.

  – **Manually Positioned**

    The light source is directly controlled by the user, who can place it at will. In this manner, the interaction can be optimized for specific areas of the display.

• **Interaction**

The following interaction techniques are implemented in the proposed framework:
– **Virtual Shadow Embodiment**

A virtual shadow is rendered to provide an embodiment for the user and, as previously described in the guidelines, to bind the extra-personal space. A cursor is attached to each shadow’s hand and triggered with a hand-held device.

– **Body-based Tools**

Virtual tools are associated to specific body locations. Just placing the hand at the corresponding position and pressing a button on the hand-held device, the user can make them appear on the display.

– **Body-based Storage**

Each user’s torso acts as a virtual container from which the personal data can be accessed.

– **Body-based control surfaces**

The control of widget components to adjust numerical values can be performed by mean of the body. For instance, by sliding the hand between two body’s joints, such as the elbow and forearm. Even a three-dimensional adjustment can be performed if more than 2 joints are used.

Even in this work the authors performed only a preliminary study, since a complete evaluation of the entire framework was not feasible. As for all the body-driven approaches, the interaction functioning was easily understood. However, in this case the participants moved a critique to the study due to the occasional ”performance hic-cups”, that caused a delay in the
rendering process. In addition, the authors also pointed out that a suitable improvement would be substituting the hand-held device with a less invasive finger-tracker.

The last but not least body-driven technique that we want to present is the (19) of D. Vogel and R. Balakrishnan. The authors propose a solution destined for large displays but addressing the problems arising from their public use.

Their first contribute is the design of the following principles:

- **Calm Aesthetics**
  
  The interface has to be neither overly reactive, because otherwise it becomes too distracting, nor exaggeratedly slow, otherwise it is felt static and unresponsive.

- **Comprehension**
  
  It must be easily understandable, even if not immediately. The ambiguity can be exploited to attract the user.

- **Notification**
The communication should be inspired by the normal socially acceptable conventions.

- **Short-Duration Activities**

  The possible activities should be short and brief since the display is publicly accessible. Furthermore, walking away should be a sufficient action to interrupt the current interaction.

- **Immediate Usability**

  No extensive prior training should be required.

- **Shared Use**

  The device should be collaborative, meaning that multiple users can interact simultaneously.

- **Combining public and private information**

  The user’s information should be between the public and private sphere. Contents like a calendar can be displayed, while an email’s body cannot.

- **Privacy**

  Techniques to discourage the users’ voyeurism should be adopted, especially because it is an accentuated phenomenon on very large displays.

Inspired by these principles, they describe a framework where the user’s interaction is differentiated into several phases:

- **Ambient Display Phase**
This is the starting point of all the other phases. Some publicly available information is shown and only if the user comes closer, additional text labels fade in the view.

- **Implicit Interaction Phase**

  When the user passes by, an implicit interaction phase begins and the user’s position and orientation start to be tracked. The attention can be attired, for example, by showing a personal alert in which he is informed about an important information he, in particular, should read or watch. If specific signs of engagements are detected, a vertical "proxy bar" appears on the display area showing more information of interest. Similarly, the user can also notify his disinterest for the information with another signal.

- **Subtle Interaction Phase**
If the user comes closer and thus provides the system with a cue of his engagement, more detailed information about his personal state is displayed. Since this information is still visible for all the other users, it is meant to be harmless. The user is already able to interact from the distance by mean of hand gestures and body movements. For example, the palm-down gesture is used to show the information, while up and down movements are used to select the items.

- **Personal Interaction Phase**

  When the user is sufficiently close to the screen, the personal interaction phase starts.

  In this phase, the occlusion granted by the body provides a higher privacy level and therefore more private information can be shown on the display. In addition, to respect the "short-duration" principle, this phase cannot last more than 5 minutes.

  The authors conducted an informal user study with four participants without any prior knowledge of the interface functioning. The evaluation was divided in two distinct parts. In the first part they wore a special vest made of trackers and were asked to talk aloud while exploring. In the second instead a glove was given, so to allow first the gestures required in the subtle interaction phase and then the direct touch proper of the personal interaction phase.

  As usual, all the subjects understood immediately how to interact, even though they had some troubles in adjusting the transition threshold between the subtle phase and the personal phase. As part of future developments, the authors plan to substitute the expensive Vicon tracking system used to track the user with something less intrusive.
2.4 Ray-Casting

In a ray-casting based interaction the user controls completely or partially two reference points to form the line equation of a ray in a three-dimensional space. Then, from the intersection of the rays equation and the displays model, a cursors position on the screen is extracted.

These techniques are notoriously fast, especially if compared with the direct touch on a high-resolution large-sized display. The main reason for this is that usually a cursor’s gain adjustment is intrinsic in the technique implementation.

In addition, as shown in (20), the possible approaches are as many as the possible choices for the two reference points, each of them with advantages and disadvantages. The target of their work is to perform a comparison among the following alternatives and, moreover, investigate if the parallax, the distance between the center of rotation used to specify the pointing direction and the user’s views, affects or not the interaction in two-dimensional tasks.

- **Regular Laser Pointing**

  The user holds a physical device in his hands and uses it like a laser pointer. Therefore, the ray direction is dictated by the device’s physical structure. In their evaluation they adopt a red marked wand whose 6 dof are tracked.

- **Arrow Style Pointing**

  Similar to the laser pointer, but the line is somehow aligned with the user’s eye. In this comparison the wand is still used, however the user has to mimic the same action of an archer using a bow and an arrow, aiming with high precision.
• **Image Plane Pointing**

As painters place their thumb at arm’s length between their eye and a painting to estimate the sizes and the position of the drawn objects, so the ray is formed using the location of one eye and a fingertip. The final result is very similar to a direct touch interaction. In the calibration, the eye used is the user’s dominant eye and its position is approximated using markers positioned on a hat.

• **Fixed-origin Pointing**

The same the image-plane technique with the difference of using any other part of the body except the eyes. Although it was presented in the body-driven section, the "Shadow
Reaching” technique adopts this kind of ray-casting to identify the hand location on the display.

The first two techniques are also called "rotational type”, meaning that they require the user’s rotation to control the cursor on the display. The others instead are named "positional type” because only the body’s position alone already moves the cursor.

Another interesting classification is based on the use of the parallax. Image plane and arrow pointing fall in the parallax-free class, while the rest make use of the parallax.

To compare effectively these techniques, the authors performed three different experiments:

- **Horizontal Targeting**

  Since most of the large wall displays are broader than taller, the authors decided to perform a horizontal targeting task as the first experiment. The targets are horizontal bands covering the height of the display and 12 participants had to alternatively move the cursor onto each of these.

  The data were fit to both a linear and an angular version of the Fitt’s Law and the latter not surprisingly showed a better fit coefficient $R^2$. At the end, the fastest technique was the laser pointing, followed by the arrow pointing, the image-plane pointing and finally the fixed origin pointing. Conversely, the error analysis showed that the worst approach was fixed origin, followed by arrow pointing, laser pointing and the image plane pointing as the most accurate.

- **Vertical Targeting**
This task replicates the previous experiment, with the only difference that the bands are horizontal instead of vertical. As before, for each trial the completion time, the cursor location and the error were recorded. Here the linear fit had a slightly better but statistically significant fit. Furthermore, the arrow pointing technique demonstrated the highest average throughput, followed by the laser, the image-plane and finally the fixed origin technique. Similarly to the previous experiment, the best accuracy performance is achieved by the image-plane pointing, followed by the arrow pointing, the laser pointing and then the fixed origin.

- **Tracing & Steering**

The last experiment involved a task to test the steering and tracing capabilities among the compared techniques. Each subject was instructed to enter a rectangle shown on the display through the non-square entrance and to reach the inner squared area without exiting from the rectangular tunnel. The targets were shown in four different locations, with two different orientations, vertical and horizontal. In this experiment only the completion time and the average distance to the longitudinal line of the tunnel were measured. The statistical analysis showed a fitting result to the Fitt’s Law similar to the horizontal targeting task, since the angular model turned out being the best. Furthermore, the image-plane method performed the best in both the throughput analysis and the accuracy analysis.

In conclusion, these experiments revealed the effectiveness of arrow pointing and laser pointing techniques, suggesting that the parallax is not that important for targeting tasks. However
in tracing tasks the results show a completely different picture, where the image-plane method absolutely dominate on all the other counterparts.

In another work (9) instead, D. Vogel and R. Balakrishnan explore the design of a specific solution, a free-hand pointing technique to remotely control a very large high-resolution display. Driven by the premises that some tasks are best performed at distance, like sorting or sliding pictures, they proposed a solution able to compete with a direct-touch manipulation.

In its design they aimed to achieve the following desirable features:

- **Accuracy**
  
  Being able to select small targets from the distance.

- **Acquisition Speed**
  
  The acquisition should require a minimal effort and be instantaneous.
• **Pointing and Selection Speed**

The device should be fast at acquiring the target.

• **Comfortable Use**

Do not cause fatigue and strain in the user.

• **Smooth transition between interaction distances**

The transition from a close-in interaction to an interaction at distance should be smooth.

Similarly to the movement of a mouse device on its support surface, the proposed solution uses the hand’s motion to move the cursor on the screen. Since the display is vertical, the hand motion is chosen to be vertical to remain consistent.

A classical problem the authors faced was the implementation of a selection in the absence of any buttons. Some solutions found in the literature use a dwell time threshold as a click event or speech recognition. However they both introduce problems, such as an unacceptable lag time or an excessive complexity. The authors instead investigated a replacement for the kinesthetic feedback typical, proposing two different techniques to support a click down action and click up action. When the click down occurs a short animation of a square shrinking is shown and a sound is played at the same time. Likewise, during a click-up action the square expands its size and another sound is played. If after a click down event, no click up is registered within 1 second, a small fixed square appears around the cursor, representing the occurrence of a dragging event.

• **AirTap**
Using this technique, the user taps the air in the same manner he would on a mouse button or on a touch-screen. Since there is no physical constraint for the downward movement, they introduced a simple recognition algorithm that uses features like the speed, the acceleration, the absolute position and the movement axis.

- **ThumbTrigger**

  This mechanism adopts the thumb clutching as a triggering mechanism. The thumb is moved towards the index finger to signal a click event, therefore providing a kinesthetic feedback.

In addition, they also proposed three different techniques to implement the ray-casting:

- **Absolute Position Finger Ray Casting**

  The user’s finger is used to impose the direction of the ray in a very similar manner to a laser pointing approach. The most problematic aspect of this method is that the hand produces a noticeable jitter, thus compromising the cursor stability. To overcome this problem, the authors implemented a "dynamic recursive low pass filter". When the speed is less than 10 mm/s a cut-off frequency of 0.25 Hz is used, while a speed greater than 200 mm/s is the switching condition for the low-pass filter with cut-off frequency of 5 Hz. All the values in between are simply linearly interpolated.

- **Relative Pointing with Clutching**

  In this technique the hand motion is projected orthogonally on a vertical plane destined to map the cursor position. The authors implement a relative mapping respect to a physical
prop: the normal hand motion is used move the cursor, while a clenched fist allows the movement of the coordinate frame. When the latter action is occurring, the cursor’s arrow is animated to provide the user with a visual feedback.

• Hybrid

The absolute finger ray-casting pointing is combined with the relative pointing. The ray-casting is used to move the cursor on the display, while the fist clench triggers the mode to move the coordinate frame.

12 participants participated in the experiment, consisting of clicking different sized circular targets with varying distances. Each set of trials begins with a transition task where the cursor and the first target are hidden until the user holds his hand for 2 second in a pre-fixed starting location. Next, the cursor appears at an experimental distance and the user has to calibrate the technique in a comfortable manner. After the first target selection, a total sequence of three new targets appears on the screen and the user has to select each of them in the minimum time possible as well.

The experiment showed that the finger ray-casting was the fastest technique but with the largest number of errors. Therefore, in situations where the precision matters, the other techniques are more suitable.

Although the previously described works depict the ray-casting approach as a "solo" technique, other authors proposed its use in more complex and multi-modal contexts. This is the case of (21), introduced by T. Igarashi et al.
The authors improved a technique proposed in (22) by Grossman and Balakrishnam named "bubble cursor". A bubble cursor is a cursor controlled by a mouse device whose activation area changes to facilitate the selection of objects shown on the screen. The main drawback of this approach is that in a densely populated screen working area, it becomes necessary to disambiguate the target effectively.

![Figure 12: Speech filtered bubble-ray](image)

Driven by the premise that ray-casting performs better on big targets, they first replaced the mouse with it, so to allow a free-hand remote selection of the objects shown on a large display. Then, they added filtering capability that uses the user’s speech to solve the ambiguity problems. A person simply moves the bubble ray towards the object of interest and then, using the voice, informs the system of a particular property the desired object has. The objects without this property are filtered out.
In a preliminary study, the authors performed a comparison among the ray-casting approach, the bubble-ray approach and the speech-filtered bubble ray approach. Each of the 30 involved participants was requested to select rapidly a target with a particular shape located on a screen full of distractors. Once a target is selected, the screen refreshes with a new set of distractors, chosen among 6 different layouts.

Both the cursor position and time were logged and a (6 layout) x (3 technique) ANOVA test was performed. The results showed a main effect of the technique, with the speech bubble as the fastest and the ray-casting as the slowest. Likewise, the same results were obtained in the error analysis, with the speech bubble technique showing the least average error respect to the others.

Even the questionnaire showed a generalized user’s preference for the speech bubble method. However among the participants, few did not like the fact they had to speak, considering it as an additional requirement that other techniques do not have.
In this work we propose an interface for a remote coarse-grained interaction on a high resolution display based on ray-casting, where the head and a finger are chosen as the two points forming the ray, as shown in Figure 13.

A ray-casting approach represents a good choice for a remote interaction with a high resolution display. These techniques are generally fast thanks to a cursor’s gain adjustment that
is intrinsic in their implementation. Moreover the points we chose to form the ray reduce the
cursor’s jitter that would be caused by selecting two short distanced points, as shown in (20).

Despite the literature’s positive premises, unfortunately the natural hand tremor typically
present when interacting in a free-space remains and therefore a solution to handle the problem
is required.

In (9) the authors use a low-pass filter on the time series representing the finger’s three-
dimensional position, attempting to preserve a free-space interaction. Although the overall
cursor’s jitter is reduced, simple actions like clicking or dragging result difficult in a free-
space context and, therefore, the solution appears less attractive for a user. To overcome
these problems, a solution might be adopting vocal commands, as shown in (21) for a simple
cursor’s disambiguation task. However this approach is generally complex and can be easily
compromised by undesired sounds, as it happens in a crowded room.

Recent studies (23) have investigated if the introduction of a haptic-feedback can be ben-
eficial or not in a remote pointing task, like a ray-casting approach. They showed that, even
though their technique performed the best, the sensory overload, particularly if strong, can
heavily alter the user’s sensory perception, leading to a discrepancy between the measured
performance and the users’ preference. Furthermore, when adopted in a free-space context, a
haptic-feedback such as a vibration even increases the hand’s jitter.

In our ray-casting approach we opted for a weaker sensory feedback called tactile feedback.
A tactile feedback is sensed only by the exterior-receptive sense of touch, thus it is less forceful,
restricted to the surface of the finger’s skin and obviously not subject to sensory overload.
This also provides the user with an "anchor" for the finger to stabilize itself. We introduced a rigid transparent surface between the user and the display and transformed it into a touchscreen area, applying exclusively computer vision techniques to the images obtained through a color/depth camera.

Many commercial products have largely proved the effectiveness of a tactile feedback. In particular, no noticeable noise is produced by the hand’s tremor when a finger is kept in contact with the surface. In addition, the current choice of the two ray’s forming points provides a more natural and immediate interaction. In fact, the cursor’s position remains aligned with the user’s line of sight, so that he can interact with the screen images based on the objects he sees through the surface. The user interacts directly with what he sees on the external display through the
touch planar surface, perceiving a constant correspondence between his finger and the screen cursor on the distant display, regardless of his physical point of view.

Figure 15: **Line of sight coherent mapping**

This implies that the mapping between the external display and the touch surface changes dynamically according to the user’s head position. To give an example, let’s assume a person is viewing a distant object through a window from position A, as shown on the left of Figure 15. In our system the window is a symbolic representation of the touch-surface, while the distant object is an entity shown on an external display. Moving his head from position A to position B, the visual projection of this object on the window changes accordingly to a perspective...
mapping. The points that were previously mapped on the window from the distant object for position A, are now different because the head moved to position B, as shown on the right of Figure 15, and thus they have to be dynamically changed, so to preserve the correspondence with the user’s line of sight.
3.1 Component Description

The system is composed as follows:

- **Transparent Surface**

  This surface enables the user to interact with the external display like one would using a touch-screen interaction. The material allows the passage of both visible and infrared spectrum waves. The first because the user has to be able to see through the surface in order to interact with the display. The latter because the tracking device, that emits an infrared structured light, has to estimate the depth information at the surface, and beyond where the user resides. A transparent surface made of plexiglass proved to be adequate, but a special material capable of reducing the IR reflections would certainly improve the results.

- **Tracking Camera**

  In this demonstration we use a camera that is both a color sensor and a depth sensor. While the first is only used in the calibration process to identify a special checker-board pattern, the second is a key component and it is strictly required by our algorithms in order to work. The depth sensor returns depth measurements of the area in front of it and stores them in a data structure, called depth map. This data structure allows the tracking of both the head and the finger position in a three-dimensional space. In our prototype we chose a Microsoft Kinect Sensor, a good compromise between quality and cost.
Figure 16: Plexiglas surface
• **External Display**

Every computer driven display can be used in our head-finger ray-casting based approach, conceptually even non-planar if a representative geometrical model is available. However the best results are obtained if a screen bigger than the surface is chosen, so that it becomes possible to exploit the benefits of a mapping to a scaled image of the display. For instance, a high-resolution display is a sufficiently large screen to prove our concept. In our prototype we are using EVL’s passive stereo tiled LCD display, with a total resolution of 8160 x 2304.
Figure 18: **Large high-resolution screen**
3.2 Software Architecture

![Diagram showing software components](image)

**Figure 19: Software components**

The system is split into two main software components to allow more flexibility and future developments, as shown in Figure 19. In our implementation we separate the system into two distinct processes that communicate using a TCP socket connection: the server side or data generator and the client side or data consumer.

- **Data Generator**

  This component is in charge of computing the positions of the cursor to be displayed on the screen. Therefore it is involved in the calibration process, the detection of the user’s finger and head and, finally, the ray intersection with the display.

  As shown in the scheme of Figure 20, this process is in turn composed by two distinct parallel tasks.

  - **Finger Detection**
Figure 20: **Data generation**
In this task each frame is first pre-processed using image analysis algorithms and then provided as an input to a component that performs the detection of a finger. A successfully detected location is stored in a special data structure that uses the last $N$ observations to determine the presence or absence of the finger on the touch-surface, called "Dynamic Accumulator".

- **Head Detection**

  A similar procedure is followed in the head detection task. The only exception is represented by the possibility of providing a translation correction for the position identified as the head location.

Using the models identified in a calibration phase, these two locations identify in space the ray that determines a cursor’s position intersecting the planar model of the display.

- **Data Consumer**

  This component, unaware of the generation process, retrieves sequentially the cursor’s locations sent by the data generator and "uses" them. The way they are used is left to the programmer’s implementation. As it will be shown in the next sections, the purpose of this component is to perform an enhancement on the generated data, remaining coherent with the generation process.
Figure 21: **Data generation**
CHAPTER 4

METHODOLOGY

This chapter is a detailed description of the proposed interface, emphasizing both the problems and the applied solutions.

The following will guide the reader through the evolution process of this work:

- **Models Identification**
  This section describes the mathematical models adopted to identify the objects in the working space.

- **Camera**
  The main actor of this section is the "Microsoft Kinect Sensor" and all the mathematical routines used to perform geometrical transformations.

- **Calibration**
  Here the calibration process is described, including all the steps performed by the user and the mathematical transformations involved.

- **Finger Detection**
  This section describes the fingertip detection in the working space.

- **Head Tracking**
The head tracking section deals with the methodologies used to obtain a corrected center of projection based on head location.

- **View-port Mapping**

  The last part first describes a problem called *Mapping-in-the-small* and then the adopted solution.
4.1 Models Identification

Figure 22: Plane through 3 points

As described in the previous chapter, the components that form the required geometrical model are the touch-surface, which provides a tactile feedback to the user, and the external large display, where the controlled cursor is moved. Since these components are vital for the whole system’s functioning, allowing a user to interact with the display, this chapter also describes our attempt to achieve the best representation for them.

Both the touch-surface and the display can be effectively modeled as a rectangular section on a plane or, equivalently, as four collinear points in the three-dimensional space, because they are flat and with a fixed position. Therefore, a crucial point of this whole process is performing a robust estimation of the plane in which they lay.
\[
\vec{N} \cdot (\vec{X} - \vec{X}_0) = 0
\] (4.1)

\[
\vec{N} = (a, b, c)^T \quad \vec{X}_0 = (x_0, y_0, z_0)
\]

Among the different mathematical representations of a plane, there is a popular formulation called "dot-form". Shown in Equation 4.1, this representation defines the plane as "the geometrical place of all the vectors which are orthogonal to a given direction". This direction is commonly called "normal" and is represented in Equation 4.1 by the vector \( \vec{N} \). The vector \( \vec{X}_0 \), represents the component of translation, generally used when the plane does not pass through the coordinate system’s origin.

Using this representation, a plane can be completely expressed by mean of only three points, as shown in Equation 4.2. The vectors \( \vec{Q}, \vec{P} \) and \( \vec{R} \) are represented in Figure 22.

\[
\left[ (\vec{Q} - \vec{P}) \times (\vec{R} - \vec{P}) \right] \cdot (\vec{X} - \vec{P}) = 0
\] (4.2)

Although the Equation 4.2 states that only three points are required to identify the plane’s model, in a real-world context the situation changes drastically. In fact, the points involved in the model identification process are sampled by a device with a limited accuracy, therefore implying that every measurement is affected by two types of error. The first is produced during the signal quantization, as shown in Figure 23, where the actual points fall short of the discrete sample locations. The second is caused by the signal noise, that is typically Gaussian distributed.
To overcome this issue, a clever solution consists of using more than the minimum required number of points to determine the plane equation’s coefficients, since the additional information can mitigate the effect of these errors.

4.1.1 MSE Data Fitting

When a linear system is formed by more equations than required, it is commonly referred to as an "over-determined" linear system. Such systems cannot be solved to produce an exact solution unless the equations in excess do not add any information. For example, when three lines intersect the same single point (Figure 24) it is equivalent to say that one of the equations of the associated linear system linearly depends on the other two. Likewise, any arbitrary set of collinear points identifies the same plane model in a three-dimensional space.

However, as previously mentioned, when dealing with the limitations associated with a real-world case, the situation changes quickly. Linear systems, theoretically are supposed to have a
unique solution, but can lead to a non-solution because of the data’s perturbations caused by the device’s errors. For instance, four points lying in the same planar surface may lose their collinearity property once sampled, therefore leading to no solutions.

In these cases, the best option is to find a solution that, rather than being exact, minimizes a certain cost function based on the available data. When the assumed model is chosen properly, the estimated model is that which best fits the available data.

\[
E \left[ \left( \hat{\theta} - \theta \right)^2 \right] \simeq \frac{1}{n} \sum_{k} \varepsilon_k^2
\]  

(4.3)
A very popular cost function is the "mean squared error", shown in Equation 4.3. In statistics, this cost function represents the difference between the actual observations and the response of the estimated model. Therefore it can be used to verify whether the predicted model fits the data or not.

4.1.1.1 Line

The mathematical method to estimate the line equation that best fits the available points, is known as "simple linear regression". Here we want to find the parameters $a$ and $b$ of the line model in Equation 4.4 that minimizes a cost function, such as the least squared error.

$$y = a + bx$$  \hspace{1cm} (4.4)
If we assume to have \( n \) points, namely \((x_k, y_k)\), and a line model whose coefficients are \( a \) and \( b \), the error \( \varepsilon_k \) (Equation 4.5) relative to the \( k \)-th point is computed as the distance measured along the \( y \)-axis from the point to the line model, shown in Figure 25.

\[ \varepsilon_k = y_k - a - bx_k \quad (4.5) \]

The "least squared error" cost function \( J \) can then be expressed with the formula Equation 4.6.

\[ J = \frac{1}{n} \sum_{k=1}^{n} \varepsilon_k^2 = \frac{1}{n} \sum_{k=1}^{n} (y_k - a - bx_k)^2 \quad (4.6) \]

To find the model’s coefficients which minimize the cost function, we have first to compute the partial derivatives of \( J \) respect to \( a \) and \( b \) and then equate them to zero, thus obtaining the linear system shown in Equation 4.7.

\[ \frac{\partial J(a,b)}{\partial a} = -2 \frac{1}{n} \sum_{k=1}^{n} (y_k - a - bx_k) = 0 \]
\[ \frac{\partial J(a,b)}{\partial b} = -2 \frac{1}{n} \sum_{k=1}^{n} (y_k - a - bx_k) x_k = 0 \quad (4.7) \]

The solution of Equation 4.7 is given by the coefficients of Equation 4.8, where \( \bar{X} \) and \( \bar{Y} \) are the sample means respectively of the set composed by all the \( x_k \) coordinates and the set composed by all the \( y_k \) coordinates:
\[
b = \left(\frac{\sum_{k=1}^{n} x_k y_k}{\sum_{k=1}^{n} x_k^2}\right) - n\bar{X}\bar{Y} - n\bar{X}^2
\]

\[
a = \bar{Y} - b\bar{X}
\]  

(4.8)

As a careful reader may have already noticed, this approach conceals a major problem. Since the coefficient of the \( y \) variable is fixed unitary, the proposed model cannot represent vertical lines. In addition, due to the so-called round-off errors, the model also fails when the cloud of points shows a very vertical trend.

Among the possible solutions, we opted for the simplest one since this operation is performed only once during the calibration process. Two best fitting equations are computed, the first with the \texttt{lineq} and the second using the \texttt{lineq}. Then, the model with the least squared error is chosen.

\[
x = c + dy
\]  

(4.9)

An alternative method involves choosing a line model where also the coefficient of the \( y \) variable is a parameter to be estimated. Although we do not report it here, it can be easily inferred from the description given in the paragraph about the best fitting plane estimation.

4.1.1.2 Plane

One way of finding the best fitting plane equation is just incrementing the previously described method’s dimensionality and then using the equation Equation 4.10.
Figure 26: **Fitting plane**

\[ y = a + bx + cz \] (4.10)

As stated before, the constrained coefficient of the \( y \) variable represents a limitation of the proposed model, since the Equation 4.10, for instance, cannot represent planes parallel to the \( xz \)-plane. To overcome this obstacle, we adopt the more general model indicated by the equation Equation 4.11, where \((a, b, c, d)^T\) are all coefficients to be estimated.

\[ ax + by + cz - d = 0 \] (4.11)

Unlike before, this time the error \( \varepsilon_i \) is computed as the orthogonal distance from a point \((x_i, y_i, z_i)\) to the plane model, shown in Equation 4.12 and graphically observable in Figure 26.
\[ \varepsilon_i = \frac{ax_i + by_i + cz_i - d}{\sqrt{a^2 + b^2 + c^2}} \]  

(4.12)

Consequently, the least squares cost function becomes equation Equation 4.13, where \( \sqrt{a^2 + b^2 + c^2} \) is constrained to 1 to further simplify the calculations.

\[ J = \frac{1}{n} \sum_{k=1}^{n} \varepsilon_k^2 = \frac{1}{n} \sum_{k=1}^{n} \left( \frac{ax_i + by_i + cz_i - d}{\sqrt{a^2 + b^2 + c^2}} \right)^2 \]  

(4.13)

After having first computed the partial derivatives of \( J \) respect to \( a \), \( b \) and \( c \) and then equating them to zero, the resulting linear system Equation 4.14 has to be solved.

\[ \frac{\partial J(a,b,c,d)}{\partial a} = \sum_{k=1}^{n} 2(ax_k + by_k + cz_k - d) x_k = 0 \]

\[ \frac{\partial J(a,b,c,d)}{\partial b} = \sum_{k=1}^{n} 2(ax_k + by_k + cz_k - d) y_k = 0 \]  

(4.14)

\[ \frac{\partial J(a,b,c,d)}{\partial c} = \sum_{k=1}^{n} 2(ax_k + by_k + cz_k - d) z_k = 0 \]

\[ \frac{\partial J(a,b,c,d)}{\partial d} = -\sum_{k=1}^{n} 2(ax_k + by_k + cz_k - d) = 0 \]

The fourth equation of Equation 4.14 can first be rewritten as shown in Equation 4.15 and then \( d \) computed with Equation 4.16 after the estimation of \( a \), \( b \) and \( c \), where the vector \((x_0, y_0, z_0)^T\) of the Equation 4.17 equation represents the barycenter of the sampled points.

\[ \sum_{k=1}^{n} (ax_k + by_k + cz_k) = \sum_{k=1}^{n} d = nd \]  

(4.15)

\[ d = ax_0 + by_0 + cz_0 \]  

(4.16)
\[ x_0 = \frac{\sum_{k=1}^{n} x_k}{n}, \quad y_0 = \frac{\sum_{k=1}^{n} y_k}{n}, \quad z_0 = \frac{\sum_{k=1}^{n} z_k}{n} \quad (4.17) \]

Therefore, the problem is reduced to solving the homogeneous linear system Equation 4.18.

The solution \((a, b, c)^T\) is obtained by performing the "Singular Value Decomposition" (SVD) of the associated matrix, since the singular vector \(V_\lambda\) associated to the minimum singular value is a solution constrained to a unitary norm.

\[
\begin{pmatrix}
\sum_{k=1}^{n} (x_k - x_0) x_k & \sum_{k=1}^{n} (y_k - y_0) x_k & \sum_{k=1}^{n} (z_k - z_0) x_k \\
\sum_{k=1}^{n} (x_k - x_0) y_k & \sum_{k=1}^{n} (y_k - y_0) y_k & \sum_{k=1}^{n} (z_k - z_0) y_k \\
\sum_{k=1}^{n} (x_k - x_0) z_k & \sum_{k=1}^{n} (y_k - y_0) z_k & \sum_{k=1}^{n} (z_k - z_0) z_k
\end{pmatrix}
\begin{pmatrix}
a \\
b \\
c
\end{pmatrix}
= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad (4.18)
4.2 Camera

As previously mentioned, our system requires the use of a special camera capable of sampling the image color but also estimating the depth of the objects within its field of view. The last feature, in particular, is extremely important, because it allows first and foremost the tracking of the user's body that is used for the interaction and secondly it facilitates the calibration process.

Driven by the high interest shown by the scientific community and its affordable price, we chose to use a device called "Microsoft Kinect" as our prototype.

Figure 27: Microsoft kinect

The Kinect, originally known by the code name "Project Nathal" (24), is an input device made by Microsoft and primarily destined for the XBox 360 game console market.

Build to compete with other consoles like Nintendo Wii and Sony Playstation, the idea behind the project was to allow the users to control a game without using a common input controller but by mean of natural body movement. In fact, Microsoft advertised a launch
budget of 500 million USD, a much larger sum than that used for the XBox console, where the marketing campaign, called "You are the controller", aimed to reach as many users as possible.

This idea proved to be very successful. First launched in North America on November 4, 2010 and then in Europe on November 10, 2010, the Kinect holds the Guinness World Record for being the "fastest selling consumer electronics device". 8 million camera units were sold in the first 60 days (25).

The Kinect sensor is fundamentally a two-cameras system, connected to a motorized base that can change its pointing direction, as shown in Figure 28.

![Figure 28: Kinect schematics](image)

- Color Camera
A RGB camera with 8 bits per channel and a 640x480 pixel resolution. It works at a 30 Hz sampling rate.

- **Depth Camera**

  Composed by an IR pattern emitter and an IR receiver, it allows the estimation of a ”depth-map”, a 640x480 matrix containing a per-pixel depth, orthogonally measured from the camera. It works at a 30 Hz sampling rate.

  The depth is estimated using a patented technique called *Light Coding*, a variant of the well-known triangulation method used in epipolar geometry.  

  A static pseudo-random pattern, hard-wired in the sensor, is projected on the environment using the infrared emitter. Next, by mean of the infrared receiver, the resulting light-structured pattern, altered by the incident objects, is first sampled and then used to estimate the depth, exploiting the disparity between the original pattern and the retrieved pattern. The final result is a per-pixel 11-bits coded distance.

### 4.2.1 Depth Linearization

Although 11-bits translate into a range of values between 0 and 2047, the actual device's precision drops as the distance increases. Nathan Crock (26) and many others have shown how the actual distance assumes a logarithmic trend, as observable in Figure 29.

One of the transformations that can be used to perform a linearization and change the scale to millimeters, as shown in Figure 29, is the Equation 4.19, proposed by the ROS project (27).
4.2.2 Pin-hole Model

Most cameras can be effectively described with a projection model called a "pin-hole model". This model involves the use of two mathematical entities, the view-plane and the center of projection. The first represents the location where objects observed from the camera are perspective projection. The latter, combined with a point on the object, is used to determine the direction of projection.

A perspective projection is computed using the matrix transformation shown in Equation 4.20. Since the result is expressed in homogeneous coordinates, a "perspective divide" has to be performed on the vector \((x, y, w)^T\) to obtain the Cartesian representation. This transfor-
mation consists of dividing all the vector’s elements by the third element $w$. The parameters $f_x, f_y, c_x, c_y$ of the projection matrix can be obtained by mean of a standard camera’s calibration technique (28). They are generally called "intrinsics" because they represent the camera’s internal property, not a function of its location in space.

$$
\begin{bmatrix}
  x \\
  y \\
  w
\end{bmatrix} = \begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  X \\
  Y \\
  Z
\end{bmatrix}
$$

Equivalently, the matrix form of the Equation 4.20 can be alternatively expressed as Equation 4.21, which is much simpler to compute.

$$
x = f_x \frac{X}{Z} + c_x \quad y = f_y \frac{Y}{Z} + c_y
$$

This model, however, cannot be used as it is for an off-the-shelf camera because it is too idyllic. In fact, due to limitations of the production process, the camera’s lenses cannot be
made perfectly parabolic but are instead shaped using a spherical model. This is particularly 
true for cheap cameras where the manufacturing process causes obvious spherical aberrations. 
When this happens, straight lines are not preserved, leading to a visual result as that shown in 
Figure 31.

![Camera aberration](image)

**Figure 31: Camera aberration**

The pixel coordinates obtained by using the Equation 4.20 have to be undistorted by the 
spherical-model correction introduced by Brown in (29) and shown in Equation 4.22. The 
parameters $k_1, k_2, k_3, p_1, p_2$ are called distortion parameters.
\[ x_{und} = x_{dist}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_{dist} y_{dist} + p_2 (r^2 + 2x_{dist}^2) \]

\[ y_{und} = y_{dist}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1 (r^2 + 2y_{dist}^2) + 2p_2 x_{dist} y_{dist} \]  

(4.22)

\[ r^2 = x_{dist}^2 + y_{dist}^2 \]

Fortunately, our device choice greatly simplifies the calibration task. In fact, the Kinect’s factory calibration intrinsics are freely accessible using the APIs provided by the OpenNI framework. This framework provides a set of open-source functions, to interface with natural interaction devices, devices that capture body movements and sounds to allow a more natural interaction of users with computers (30).

However, to apply the Equation 4.20 the information retrieved using the APIs must first be converted since the returned field of view parameters are expressed in radians, shown in Equation 4.23. To do so, the formula Equation 4.24 is applied.

\[ Fov_h = 1.014469 \quad Fov_v = 0.789809 \]  

(4.23)

\[ f_x = \frac{c_x}{\tan\left(\frac{Fov_h}{2}\right)} \quad f_y = \frac{c_y}{\tan\left(\frac{Fov_v}{2}\right)} \]  

(4.24)

### 4.2.3 Projection Inverse

Usually, the projection is a non-invertible transformation, meaning that a point which was projected on a view plane cannot be restored to its original location. In fact, the three-dimensional points which project onto the same bi-dimensional location are infinite and all
located along the same projection line, formed by the center of projection and the projected point.

This situation, however, changes radically when a depth measure is available for each projected coordinate, as it happens with a depth sensor. In this case the degree of freedom of the inverse transformation can be constrained, therefore leading to a unique solution. The inverted relation becomes the Equation 4.25 or explicitly the Equation 4.26.

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix} =
\begin{pmatrix}
\frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\
0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
w = Z
\end{pmatrix}
\] (4.25)

\[
X = (x - c_x) \frac{w}{f_x}
\]

\[
Y = (y - c_y) \frac{w}{f_y}
\]

\[
Z = w
\] (4.26)

Using the Equation 4.25, for instance, a 3D reconstruction of the environment observed from the camera, although rudimentary, can be performed, as shown in Figure 32.

Each camera’s pixel is inversely projected to its original three-dimensional location and the associated color is obtained by sampling the corresponding value on the RGB image. Since the two cameras have the same parameters but different center of projections, they have to be aligned before the color sampling could be performed. To do so, a horizontal translation of 7.5 centimeters is required (31).
Fortunately, the OpenNI framework relieves us from this ask, performing automatically the required stereo alignment. Therefore, to obtain the color we only need to query the pixel at the corresponding position.

The result, despite its simplicity, is key to visually debugging all the geometric and calibration transformations involved in this work.

4.2.4 Projection On A Plane

As previously stated, a point after a projection transformation cannot be restored unless the associated depth information is available. Once the latter is available, the equation Equation 4.26 can be applied as already shown.

A more general case, that also includes the one previously mentioned, is the projection on a plane. Instead of restoring the projection of a point to its original position given the depth information, we want to obtain its location on a plane in space. This problem is equivalent to
finding the intersection between the projection ray and the constraining plane $ax + by + cz = d$, as shown in Figure 33.

Figure 33: Projection on plane

Once again this relation is invertible and leads to a solution if the Equation 4.27 is solved.

$$x_s = f_x \left( \frac{X}{Z} \right) + c_x = f_x \left( \frac{X}{\frac{d}{c} - \frac{a}{c} x - \frac{b}{c} y} \right) + c_x$$

$$y_s = f_y \left( \frac{Y}{Z} \right) + c_y = f_y \left( \frac{Y}{\frac{d}{c} - \frac{a}{c} x - \frac{b}{c} y} \right) + c_y$$

$$w = \frac{d}{c} - \frac{a}{c} x - \frac{b}{c} y$$

(4.27)

The solution to the Equation 4.27 is obtained by first solving the Equation 4.28 and then evaluating the plane equation with the resulting vector $(X, Y)^T$ to compute the value of $Z$. 
Moreover, the equation Equation 4.27 can also be seen as a generalization of the inverse transformation of the projection. In fact, the latter is equivalent to a projection on a plane with the form $z = constant$. 

\[
\begin{pmatrix}
\frac{a}{c}(c_x - x) - f_x & \frac{b}{c}(c_x - x) \\
\frac{a}{c}(c_y - y) & \frac{b}{c}(c_y - y) - f_y
\end{pmatrix}
\begin{pmatrix}
X \\
Y
\end{pmatrix}
= 
\begin{pmatrix}
\frac{d}{c}(c_x - x_s) \\
\frac{d}{c}(c_y - y_s)
\end{pmatrix}
\tag{4.28}
\]
4.3 Calibration

The calibration is a fundamental step necessary to identify the mathematical models of all the components involved. For this reason, the methods we propose have to guarantee the robust identification of the following entities:

- **Display**
  
The external display the user is facing, positioned behind the Kinect camera.

- **Plexiglass**
  
The transparent surface positioned in front of the Kinect, where the user interacts using his finger.

4.3.1 Overview

In this paragraph we list all the steps needed to perform the system's calibration. This procedure is a semi-automatic procedure, since it requires only partial assistance from the user. The steps involved are:

1. Move the camera to the "identification position" (Figure 34), on the user side of surface facing both the touch-surface and the large display.

2. Sample an arbitrary number of points on the display, either manually or automatically, and use them to estimate the best fitting plane that solves the MSE minimization problem.

3. Compute the display’s corners on the previously estimated plane, identifying a checkerboard pattern displayed on the screen.
4. Attach a double-sided checker-board pattern on the touch-surface and identify all the corners on the currently visible side.

5. Move the camera to the "working position" (Figure 34), which is between the display and the touch-surface, facing the user.

6. Identify all the checker-board’s corners on the other currently visible side.

7. Using the points collected from both sides of the checker-board, estimate the affine matrix transformation between the "identification position" reference system and the "working position" reference system.

8. Transform the four display’s corners from the "identification position" to the "working position" reference system using the just estimated transformation.

### 4.3.2 Display Identification

In this paragraph we describe our robust method to identify the four display’s corners in the three-dimensional space exploiting the possession of a depth sensor. As previously mentioned, this method is composed of two steps:

1. **Plane Estimation**

   To estimate the plane in which the large display lies in, we use the MSE minimization approach described earlier. The only requirement is to collect as many points as possible and feed them into our plane estimator.

   These points may be sampled either manually or automatically. In the first case, the user collects them by just clicking with the mouse cursor on the image’s part representing
Figure 34: Calibration positions
the display shown in our application. In the other case, instead, a calibration pattern temporarily attached on the display is automatically detected. In any case, at the end the sampled bi-dimensional points are restored with the formula Equation 4.26 to their three-dimensional location and used to estimate the plane. An example of correctly estimated plane is shown in Figure 35.

2. **Corners Identification**

Once the plane where the display lies on is successfully estimated, we need to identify the corners location on it. To do so, we provide two methods in our application:

- **Manual**
The user manually selects the points of the display’s edges by clicking on the camera’s color image. The selection starts with the top edge and then proceeds in a clockwise order. Once a prefixed but arbitrarily chosen number of points of an edge is sampled, their best fitting line is computed using the Equation 4.7. Finally, the corners result from the analytical intersection between all the pairs of consecutive edges. Given two lines in homogeneous representation (Equation 4.29), their intersection is given by Equation 4.30.

\[ ax + by + c = 0 \rightarrow (a, b, c)^T \]  

\[ \vec{P}_{int} = (a_1, b_1, c_1)^T \times (a_2, b_2, c_2)^T \]  

Their three-dimensional location is then easily obtained by projecting them on the previously estimated plane by mean of the Equation 4.28.

- **Automatic**

A faster and more effective method to identify the display is the automatic approach. Instead of relying on the user’s capabilities of accurately clicking on the display’s edges, this approach makes use of a computer vision technique to ease and automate the task. After the display’s plane is successfully estimated, a specially crafted checker-board pattern (Figure 36), shown in full-screen on the display, is detected and the location of all its “inner” corners are identified, as shown in Figure 38.
Similarly to the manual procedure, the best fitting lines of the inner corners are first computed to identify the corners using the line’s intersection and then projected on the estimated display’s plane using the Equation 4.27.

Figure 36: Special checker-board

Since the extracted corners do not match with the actual display’s corners, their three-dimensional position has to be corrected. To do so, it is sufficient to perform an extension of two checker-board’s squares, one for each corner. The resulting corner is computed using the Equation 4.31, where the respectively Equation 4.32
represents the width and the height of a square. The vectors $\vec{P}_0, \vec{P}_1$ and $\vec{P}_2$ are those indicated in Figure 37. The result is shown in the reconstruction of Figure 39.

Figure 37: Corner extension

\[ P_{\text{new}} = 2s_{\text{width}} \frac{\vec{P}_2 - \vec{P}_0}{|\vec{P}_2 - \vec{P}_0|} + 2s_{\text{height}} \frac{\vec{P}_1 - \vec{P}_0}{|\vec{P}_1 - \vec{P}_0|} \]  \hspace{1cm} (4.31)
\[ s_{\text{width}} = \frac{|\vec{P}_2 - \vec{P}_0|}{n_w} \quad s_{\text{height}} = \frac{|\vec{P}_1 - \vec{P}_0|}{n_h} \] (4.32)

Figure 38: Plane identification
Figure 39: Corner extension
4.3.3 Cameras Transformation Estimation

After having performed the identification of the display’s corners as described in the previous paragraph, the camera has to be moved to the "working position", that is between the touch-surface and the large display, pointing at the system’s user. Unfortunately, once the camera is moved, the previously identified corners become useless because they refer to the wrong reference system which is the "identification position" reference system. To solve this issue, it is necessary to find the transformation that transforms a point from the "identification position" reference system to the "working position" reference system.

Since setting the transformation parameters manually is an extremely inaccurate and tedious task to perform, we propose two different working methods for its automatic estimation in the next section.

4.3.3.1 Common Reference System Method

This approach involves the use of a detectable calibration object, that is fully observable from both the camera’s positions, to estimate the required affine transformation. Since we needed to detect the same points from two distinct and opposite camera’s locations, we opted for a double-sided checker-board pattern, where the corners on both faces coincide. As it will be shown shortly, this choice allows an easy identification the a same reference system from each of the two camera’s positions.

Without moving the camera from the "identification position", the double-sided checker-board pattern is attached on the touch-surface, being very careful in choosing a position that will be completely observable also from the "working position". As previously done, the corners
of the current checker-board’s face are detected on the color image and their three-dimensional location is computed with Equation 4.25.

The reference system whose transformation we want to achieve is located on the calibration object. More precisely, from the current camera position the origin $\vec{O}_1$ is the point of intersection between the top edge and the left edge of the checker-board pattern and the unit vectors determining the rotational component are oriented as follows:

- $\vec{v}_1$ Origin to up-right corner
- $\vec{u}_1$ Origin to down-left corner
- $\vec{n}_1$ Plane’s normal

While the plane’s normal $\vec{n}_1$ can be easily found estimating the best fitting plane to the checker-board pattern’s corners, the procedure required for the unit vectors $\vec{v}_1$ and $\vec{u}_1$ changes slightly. First of all, we perform a transformation on the corners belonging to the top edge and the left edge, called ”rectification”. With this transformation the corners are orthogonally projected on their bi-dimensional best fitting line, as shown in Figure 40 and in Figure 41.

Given a line with equation $ax + by + c = 0$ and a point $P_{unrect}$, its orthogonal projection on the line is given by the formula Equation 4.36, where $M_{Proj}$ is an orthogonal projection matrix and $T_1$ and $T_2$ are respectively the matrix to transform a point from the view-port reference system to a reference system aligned to the line and its inverse.
Figure 40: Checker-board identification (color)
Figure 41: Checker-board identification (depth)
Next, the unit vectors $\mathbf{v}_1$ and $\mathbf{u}_1$ are obtained first subtracting a point arbitrarily chosen on the line, with the origin $\mathbf{O}_1$ and then normalizing the result, as shown in Equation 4.37.

$$
\mathbf{v}_1 = \frac{\mathbf{P}_x^{(1)} - \mathbf{O}_1^{(1)}}{\| \mathbf{P}_x^{(1)} - \mathbf{O}_1^{(1)} \|} \quad \mathbf{u}_1 = \frac{\mathbf{P}_y^{(1)} - \mathbf{O}_1^{(1)}}{\| \mathbf{P}_y^{(1)} - \mathbf{O}_1^{(1)} \|}
$$

(4.37)

Once these vectors are all successfully computed, the first required affine transformation, used to transform a point from the "identification position" reference system to the calibration
object reference system (Figure 42), is expressed in matrix form as shown in Equation 4.39. Instead, its inverse is the Equation 4.39.

Figure 42: Built reference system
Afterwards, the camera is moved from the "identification position" to the "working position", without altering the location of the double-sided checker-board pattern. As before the affine transformation to transform a point from the "working position" to the previously described same reference system of the shared object is estimated. However, since the camera is now facing the other side of the calibration object, some means have to be taken:

- The sign of the currently estimated normal has to be inverted, so to be aligned with the previously computed normal. In this work our implementation of the fitting plane estimator always returns a plane with a normal coherently oriented with the camera’s pointing direction.

- Conversely, the origin is identified by the intersection between the top edge and the right edge.
The top edge’s unit vector is now directed from the origin to the left corner of the calibration pattern.

As before, the results are two transformations, the matrix Equation 4.41 to transform points from the camera’s "working position" to the shared object’s reference system and its associated inverse Equation 4.40.

\[
T_{\text{kin}^2 \rightarrow \text{plexi}} =
\begin{pmatrix}
v_x^{(2)} & v_y^{(2)} & v_z^{(2)} & 0 \\
u_x^{(2)} & u_y^{(2)} & u_z^{(2)} & 0 \\
n_x^{(2)} & n_y^{(2)} & n_z^{(2)} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 0 & -O_z^{(2)} \\
0 & 0 & 0 & -O_y^{(2)} \\
0 & 0 & 0 & -O_x^{(2)} \\
0 & 0 & 0 & 1
\end{pmatrix}
\tag{4.40}
\]

\[
T_{\text{plexi} \rightarrow \text{kin}^2} =
\begin{pmatrix}
0 & 0 & 0 & O_x^{(2)} \\
0 & 0 & 0 & O_y^{(2)} \\
0 & 0 & 0 & O_z^{(2)} \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
v_x^{(2)} & u_x^{(2)} & n_x^{(2)} & 0 \\
v_y^{(2)} & u_y^{(2)} & n_y^{(2)} & 0 \\
v_z^{(2)} & u_z^{(2)} & n_z^{(2)} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\tag{4.41}
\]

Combining the Equation 4.39 and the Equation 4.40 as shown in the Equation 4.42, the required transformation to transform points from the "working position" to the "identification position" is finally obtained.

\[
T_{\text{kin}^1 \rightarrow \text{kin}^2} = T_{\text{plexi} \rightarrow \text{kin}^2} T_{\text{kin}^1 \rightarrow \text{plexi}}
\tag{4.42}
\]
4.3.3.2 Affine 3D Homography Method

The Affine 3D Homography approach involves once again the use of the double-sided checkerboard pattern, completely observable from both the camera’s positions and the sampling of each face’s corners with the ultimate purpose of estimating affine transformation between the two different locations. The only difference resides in how these points are handled during the estimation process, that we call ”Affine 3D Homography” estimation.

Before entering in our specific context, it is necessary to provide a brief description about the concept of a homography. A two-dimensional homography is any projection transformation that maps points of a space $P^2$ to points of another space $P^2$, as shown for example in Figure 43. An important theorem also states that each of these projection transformations is performed using a non-singular 3x3 matrix, generally indicated with the letter $H$ (28). Therefore, when expressed in homogeneous coordinates, rotation, translation or scaling matrices all represent particular instances of a bi-dimensional homography transformations.

![Figure 43: Homography transformation](image)
Dealing with perspective transformations, usually the problem to solve is estimating the matrix $H$, given a set of at least four 2D to 2D correspondences. As shown by R. Hartley and A. Zisserman in (28), determining the $H$ matrix can be done using a well-known algorithm called ”Discrete Linear Transformation” or DLT algorithm.

Inspired by this approach, we attempted to generalize and apply it to our case, the estimation of the affine transformation between the ”identification position” and the ”working position”. This transformation can be expressed with the 4x4 matrix in homogeneous coordinates shown in Equation 4.43, where the first three elements of the last row are zeroed out, because it represents an affinity.

$$
H_{1 \rightarrow 2} = \begin{pmatrix}
    h_1 & h_2 & h_3 & h_4 \\
    h_5 & h_6 & h_7 & h_8 \\
    h_9 & h_{10} & h_{11} & h_{12} \\
    0 & 0 & 0 & h_{13}
\end{pmatrix} \quad (4.43)
$$

As for the two-dimensional case, we used the DLT algorithm which guarantees the estimation of the matrix $H$ up to a scale factor $\alpha_k$, as shown in Equation 4.44. Since the three-dimensional points are represented in homogeneous coordinates, the scale factor has no influence and thus it can either be removed after the use of the transformation, dividing all the resulting vector’s components $(X_i, Y_i, Z_i, W_i)$ by the fourth component $W_i$, or dividing each matrix element beforehand by $h_{13}$, since the transformation is known to be affine.
\[ \alpha_k x_k' = H x_k \]  

(4.44)

By left-multiplying both the sides of the equation Equation 4.44 with \( x_k'^T A \), where \( A \) is an anti-symmetric matrix, the Equation 4.44 simplifies to the equation Equation 4.45. For every anti-symmetric matrix \( A \), in fact, \( x^T A y = -y^T A x \) and therefore, when \( x = y \) it is that \( x^T A x = 0 \).

\[
x_k'^T A \alpha_k x_k' = x_k'^T A H x_k \rightarrow \alpha_k x_k'^T A x_k = x_k'^T A H x_k \rightarrow 0 = x_k'^T A H x_k
\]  

(4.45)

The expression Equation 4.45 represents a single linear equation and using algebraic manipulations it can be rewritten as the Equation 4.46. \( s_1, s_2, s_3 \) and \( s_4 \) are the columns of the 4x4 chosen anti-symmetric matrix \( A \).

\[
\begin{pmatrix} 
  x_k'^T s_1 x_k & x_k'^T s_2 x_k & x_k'^T s_3 x_k & x_k'^T s_4 x_k
\end{pmatrix}
\begin{pmatrix} 
  h_1 \\
  h_2 \\
  h_3 \\
  \vdots \\
  h_{16}
\end{pmatrix} = 0
\]  

(4.46)

The elementary anti-symmetric matrices of dimension \( n \) are as many as the number of the sub-diagonal elements, that is \( \frac{n^2 - n}{2} \). In our case there are 6 since \( n = 4 \), as shown in Equation 4.47.
Therefore, for each given correspondence $x_k' \leftrightarrow x_k$, six equations can be derived using each of the Equation 4.47, as shown in Equation 4.48.

$$
\begin{pmatrix}
-x_2 x'^T & x_1 x'^T & 0 & 0 \\
-x_3 x'^T & 0 & x_1 x'^T & 0 \\
-x_4 x'^T & 0 & 0 & x_1 x'^T \\
0 & -x_3 x'^T & x_2 x'^T & 0 \\
0 & 0 & -x_4 x'^T & x_3 x'^T \\
0 & 0 & -x_4 x'^T & x_2 x'^T
\end{pmatrix}
\begin{pmatrix}
h_1 \\
h_2 \\
\vdots \\
h_{13}
\end{pmatrix}
= 0 \quad (4.48)
$$

Even though the number of equations is six, the rank of the 6x13 matrix shown in the Equation 4.48 is only 3. This means that each point correspondence $x_k' \leftrightarrow x_k$ produces at
most only three linearly independent equations and thus three of the six equations constitutes redundant information

\[
A \left( \begin{array}{cccc}
    h_1 & h_2 & \cdots & h_{13}
\end{array} \right)^T = 0 \tag{4.49}
\]

To find the 13 parameters of \( H \), the linear system Equation 4.49 has to be solved, where \( A \) is the matrix obtained stacking three linearly independent equations for each available correspondence. Specifically, since the number of unknown parameters is 13 and we demand a non-trivial solution, the total number of correspondences required to stack the necessary 12 linearly independent equations is 4.

When more than 4 correspondences \( x'_k \leftrightarrow x_k \) are available, the linear system becomes over-determined, leading most of the time to a non-unique solution. To overcome this issue, the research for an exact solution of Equation 4.49 is transformed into a cost-minimization problem, where the minimizing cost is the algebraic norm \( \|Ah\| \). A solution \( h \), constrained to \( \|h\| = 1 \), is given by the unit singular vector corresponding to the smallest singular value of the Singular Value Decomposition (SVD) of the matrix \( A \). Finally, the parameters of the affinity transformation are obtained dividing each element of the solution \( h \) by \( h_{13} \).

However, a problem that arises in our specific case is that the set of sampled matched points are collinear and therefore do not provide all the required information necessary to estimate the matrix. In fact, the available data shows that the length is preserved only on the planar subspace where the sampled points lie on, but no information can be extracted regarding an eventual scale factor along the plane’s normal direction, since there are no points.
Although the data are not sufficient for a solution as they are, by observing carefully the nature of this particular problem, it can be noted that the lengths are always preserved because the required transformation matrix is a composition of only translations and rotations. Therefore, the missing information can be artificially provided extending both the sampled set with additional points: for each element in the original set, a new corresponding point is added, obtained as its summation with the plane’s normal, multiplied by an arbitrary constant, as shown in Figure 44. The scale invariance is asserted by imposing this constant to be equal in both the sets.

Figure 44: Scale invariance compensation
4.4 Finger Detection

In this section we provide a detailed description of the user’s finger tracking, implemented by mean of computer vision techniques. In particular, we achieved an implementation with the following two properties:

- **Simplicity**

  The algorithm is simple and therefore fast allowing more flexibility for future developments, like a motion gesture recognition task.
• Robustness

The limited depth map resolution provided by the sensor, combined with a very noisy signal caused by the depth estimation technique, has a strong impact on the final result. Therefore we provide a robust solution.

4.4.1 Overview

This section describes a simple but effective algorithm to track a user’s finger interacting on the touch-surface, first essential component in our ray-casting approach.

Using the depth provided by the sensor, an image can be easily filtered leaving only the part of the finger that is of interest, such as where the interaction occurs. For example, if pixels outside a given minimum and maximum depth threshold are zeroed out, the resulting image will represent a section of the space. In particular, we observed that when a finger enters vertically such a section, the filtered result is a blob, a small solid circle on the image, as shown in Figure 46. Moreover, the barycenter of this blob results a good representative for a fingertip location on the image.

Driven by these premises, in our detection approach we subdivide the three-dimensional space dedicated to the finger motion into several parallel and equally-spaced planes, starting from the reference plane of the touch-surface and directed towards the user (Figure 47). Each pair of sequential planes defines a ”section” where the user’s finger presence can be detected.

As previously mentioned, the sensor inaccuracy and the signal noise reduces the robustness of this approach. Consequently, we propose an intelligent data structure, called ”dynamic accumulator”, to improve the stability of the finger-tip. As will be shown in the next section,
Figure 46: Blob example

Figure 47: Space layering
this data structure acts as an observation window that buffers the detection results of the last $n$ frames. Then, on the basis of its current state, a finger-tip location on the image may or may not be computed from the buffer.

The use of a history window leads to several advantages:

- **Stability**

  The extracted cursor can be synthesized using all the last $n$ detected positions, leading to an improvement, such as a smoother estimate of finger position.

- **Outliers Detection**

  The history’s spatial locality typical of a direct-touch interaction allows the identification of outliers, points unlikely to be a true estimate of the finger-tip position to the buffer.

- **Time Series Analysis**

  Since the last $n$ detected points represent a time series, special patterns, such as finger’s gestures, can be recognized using advanced machine learning techniques.

### 4.4.2 Threshold Map

Before starting to track the user’s finger position, also the touch-surface requires to be properly calibrated. More precisely, we need to provide a method capable of efficiently defining the minimum and maximum depth threshold values for a given section.

At first, with the camera still located in the ”working position”, the corners’ three-dimensional positions detected on the currently visible face of the double-sided checker-board pattern are used to estimate the plane model associated to the touch-surface. Then, for each camera’s
pixel coordinate, its associated depth value on this estimated plane is sampled using the Equation 4.27 and stored in the corresponding cell of a bi-dimensional array, called the "threshold map".

From this reference threshold map, any other parallel plane’s threshold map can be easily computed by just adding a positive offset to its values which lies on the camera’s z-axis, as shown in as shown in Figure 48. However, since we are interested in expressing such offset....
according to an orthogonal distance along the plane’s normal $\vec{N}$ and not the camera’s z-axis $\vec{Z}$, the corrective factor (Equation 4.50) is necessary to transform one into the other.

Figure 49: Ortho correction

$$\frac{Dist_{\perp}}{Dist_z} = \cos \theta = \hat{\vec{Z}}_{axis} \cdot \vec{N}_{plane} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = n_z \rightarrow [Dist_{\perp} = n_z Dist_z]$$ (4.50)
Hence the formula to identify the two depth values limiting a k-section associated to the coordinate \((x_i, y_i)\), providing a section’s thickness parameter \(\Delta S_t\) measured along the plane’s normal \(\vec{N}\), is given by (Equation 4.51). \(m\) is just another parameter to translate rigidly all the sections from the touch-surface plane, while \(t(x, y)\) is the depth value stored in the reference threshold map at the position \((x, y)\).

\[
T_{k_1} = n_z [t(x, y) + m + (k + 1) \Delta S_t] \\
T_{k_2} = n_z [t(x, y) + m + k \Delta S_t]
\]

(4.51)

Additionally, the user is given the possibility to arbitrarily define a quadrilateral region to exclude portions of the image containing artifacts from the detection process, as shown in Figure 50. Depth pixels outside this defined area are zeroed out, thus not providing any contribution to the blob detection process.

Once an area is chosen by selecting four points in the view-port, verifying the presence of a pixel is an easy and relatively fast procedure. With the proper choice of three consecutive vertices, every quadrilateral can be always split into two non-overlapping triangles, for which a fast test in barycentric coordinates is available.

Given the triangle shown in Figure 51 formed by the vertices \(P_0, P_1\) and \(P_2\), the transformation required to transform points from the view-port reference system to the barycentric coordinates of the triangle is represented by the equation Equation 4.52. A pixel \((x, y)^T\) is inside the triangle if and only if its associated barycentric coordinates \((u, v)^T\) satisfy \(u > 0\) and \(v > 0\) and \(u + v < 1\).
Figure 50: Raw blob mask

\[
\begin{pmatrix}
u \\ v
\end{pmatrix} = \frac{1}{P_{1x}P_{2y} - P_{2x}P_{1y}} \begin{pmatrix}
P_{2y} & -P_{2x} \\ -P_{1y} & P_{1x}
\end{pmatrix} \begin{pmatrix}
x \\ y
\end{pmatrix}
\]

(4.52)

\[
P_{1x} = x_1 - x_0 \quad P_{2x} = x_2 - x_0 \quad P_{1y} = y_1 - y_0 \quad P_{2y} = y_2 - y_0
\]

(4.53)

\[
P_0 = (x_0, y_0)^T \quad P_1 = (x_1, y_1)^T \quad P_2 = (x_2, y_2)^T
\]

4.4.3 Finger-tip Extraction

As previously described, in our detection approach we subdivide the three-dimensional space dedicated to the finger motion into several parallel and equally-spaced planes, starting from the
reference plane of the touch-surface and directed towards the user. Each pair of sequential planes, named "section", is used to detect an interacting finger.

Once the reference threshold map has been estimated, the membership of a coordinate \((x, y)^T\) to a \(k\)-th section, starting from the touch-surface’s plane, can be easily verified using the expression Equation 4.51. To accomplish this, the finger detection process involves the creation of a binary image associated with each section. For each section the depth-map retrieved by the depth sensor is subject to a binary decision process, assigning 1 to the position \((x, y)\) when \(d(x, y) \leq T_{k_2}\) and 0 otherwise. In this manner, when a finger-tip is within a section, the resulting image shows a circular-shaped "blob", made of 1-pixels.
Additionally, to perform a smoothing of the binary blob’s edges and eventually remove "salt-and-pepper" noise produced by both the sensor and the binary decision process, a median filter is applied to the whole image. Good results are obtained with a 5x5 kernel with an image resolution of 640x480 pixels.

Once the binary mask is obtained and adequately filtered, the contours can be extracted using the algorithm described by Suzuki in (32), whose implementation is provided by the OpenCV library. This algorithm identifies the outermost pixels that enclose a connected region made of 1, as shown in Figure 52, and returns it as a list of bi-dimensional points.

![Contours example](image)

Figure 52: **Contours example**
The pixel chosen as the finger-tip location is represented by the Equation 4.54, that is the barycenter of the detected contour. Furthermore, its three-dimensional position is computed projecting this coordinate on the closest plane to the touch-surface associated at the current section.

\[
(x_{\text{bar}}, y_{\text{bar}})^T = \left( \sum_{i=1}^{n} \frac{x_i}{n}, \sum_{i=1}^{n} \frac{y_i}{n} \right)^T
\] (4.54)

The finger-tip’s search is performed sequentially in each section, starting from the one closest to the touch-surface. Since at the current state of this work we are assuming the presence of only one finger in the working area, if no blob is found in the current section, the search moves to the next one, until all sections have been explored. This algorithm is summarized as follows, where \(N\) represents the number of sections in which the space is subdivided. Handler A represents the code whose task is using the extracted barycenter coordinate, as, it will be shown in the next sections. Likewise, Handler B is the code that handles the situation in which no barycenter is found at all.

4.4.4 Dynamic Point Accumulator

As anticipated, the simple detection of a finger-tip interacting in a single frame is not sufficient due to the previously mentioned sensor limitations. For this reason, in this section we propose a solution to extract a smoother and more robust position, based on the history of the last \(n\) detected finger-tip locations: the ”dynamic point accumulator”.

The idea behind this approach is to ”accumulate” the last \(n\) detected results, with the final purpose of synthesizing a finger-tip location enhanced in terms of stability and in addition not
Algorithm 1 Blob Detection

\[
\begin{align*}
\text{found} & \leftarrow \text{false} \\
\text{section} & \leftarrow 1 \\
\text{while} & \text{ section } \leq N \text{ and } \text{found } \neq \text{ true } \text{ do} \\
\text{found} & \leftarrow \text{detectedBlob(section)} \\
\text{if } & \text{found} = \text{true then} \\
& \text{barycenter } \leftarrow \text{extractBarycenter()} \\
& \text{...} \\
& \text{Handler A} \\
& \text{...} \\
\text{end if} \\
& \text{section } \leftarrow \text{section } + 1 \\
\text{end while} \\
\text{if } & \text{found} = \text{false then} \\
& \text{...} \\
& \text{Handler B} \\
& \text{...} \\
\text{end if}
\end{align*}
\]

being influenced by outliers. Points which become old are forgotten and thus discarded from
the data structure.

Moreover, the data structure is also addressed as ”dynamic” because, as it will be shown in
the next sections, the extraction of the finger-tip’s position from the accumulated data uses an
adaptive mechanism, based on the manner the user is moving his finger on the touch-surface.

4.4.4.1 Data Structure

The data structure utilized to implement the history of the detected points is a queue, a first-
in-first-out data structure (FIFO). Since the history stores only the last \( n \) detected positions,
an efficient implementation is provided by a ”circular buffer”.

A circular buffer is a queue data structure characterized by a maximum fixed size. In fact, as shown in Figure 53, it is implemented by mean of an array, where the head location, used for the extraction, and the tail location, meant for the insertion, are tracked using two distinct pointers.

In our implementation of the circular buffer, each element has the following content:

- **Position**

  The coordinate \((x, y)\) of a detected finger-tip, if any.

- **Flag**

  A semantic flag expressed using an integer value with one of the following meanings
- **Invalid Point**

  Signaled with a 0-flag, the element represents an invalid point, meaning that no finger-tip was detected in the correspondent frame

- **Valid Point**

  Signaled with a non-zero flag and ranging from 1 to $n$, the number of sections, the element represents a valid finger-tip location detected in the section corresponding to the flag’s number.

These flags are extremely important because they allow us to perform statistical operations on the buffer content. Their count is updated into a histogram and the following rules are applied:

- If the percentage of invalid points is below an established threshold (e.g. 20% of the buffer dimension), a finger-tip position is computed and becomes accessible for use, as it will be shown in the next section. In all the other cases no finger-tip is available.

- The class of section associated to an accessible fingertip, is the section with the maximum value in the histogram.

4.4.4.2 **Outliers**

The interaction of a user with a surface like a touch-screen has some interesting properties that can be exploited to enhance the stability of the algorithm. The most remarkable one is without a doubt the spatial locality.
As shown in Figure 54, a user interacting with a touch-screen draws continuous trajectories with his finger from the start to the end. This observation implies that when a finger-tip point detected in frame $k$ will most likely be detected closer rather than farther in the next frame $k+1$.

$$d = \sqrt{(x_{kt} - x_{t-1})^2 + (y_{kt} - y_{t-1})^2} \quad (4.55)$$

This property can be a very useful criterion in the outliers’ identification process, especially since the camera’s sampling is performed at 30 Hz. Before inserting a candidate finger-tip point in the buffer, the Euclidean distance Equation 4.55 from the previously generated point is computed and tested with a user defined threshold, as shown in Figure 55. If this distance is
greater than the given threshold, the finger-tip candidate point is rejected and the evaluation procedure moves to the next blob.

4.4.4.3 Adaptive Filtering

This section describes the implemented extraction process of a finger-tip location from the buffer’s content, so to enhance the robustness of the cursor’s signal. The approach is based on the concept of "smoothing" or "noise-filtering", which is the use of the past $n$ observation to synthesize a location of the finger-tip at the current frame.
There are many viable alternatives to implement a signal smoother, most of which represent low-pass filters, such as the Gaussian Filter or the Butterworth Filter, whose spectrum is that of a bell function, such as the one shown in Figure 56. However, all of them present a major issue that, apart from being computationally expensive, there is a significant delay introduced between the actual finger-tip position and its associated smoothed or filtered version.

To solve these problems, we opted for an adaptive solution consisting of performing a weighted moving average (Equation 4.56) of the valid points but with weights that are automatically adjusted on the basis of the current finger’s state. The weights are assigned using an exponential progression, where a weight has the form $w_k = b^k$ and $b$ is the base of the power.
\[ M_{\text{weighted}} = \frac{\sum_{k=0}^{N} w_k x_k}{w_k} \]  

(4.56)

According to the current average cursor’s speed, a different base is selected. More precisely, the base is computed by mean of a tunable interpolation function that varies accordingly to the finger-tip’s average speed.

When the finger stands still or moves very slowly on the touch-surface, a steady and steep low-pass filtered signal is required, since the user needs precision. Thus, the extracted base is numerically very close to 1, meaning that all the elements in the buffer have the same weight. Such a filter, that represents just a simple sample mean, has a low-pass frequency representation as that shown in Figure 57, where the cut-off frequency decreases as the number of involved samples increment.

Figure 57: Sample mean frequency domain
Vice-versa, when the motion is fast, the user neither requires a precise interaction nor can he notice the presence of the noise. Instead more importance has to be given in nullifying the delay produced by the signal smoother. Therefore a higher base is chosen, so to give more weight to the most recent valid points and, thus, produce a cursor with a practically unnoticeable delay.

The interpolation functions we tested are a linear function (Equation 4.57), with two saturation levels and a sinusoidal (Equation 4.56) interpolation function. Their graphs are shown in Figure 58. Although they both lead to acceptable results, the latter guarantees continuous derivatives over all $\mathbb{R}$ and thus it is preferred.

Figure 58: Linear and sinusoidal interpolation
\begin{align}
\begin{cases}
    B_M & s_c \geq S_M \\
    \left(\frac{s_c - s_m}{S_M - s_m}\right) (B_M - b_m) + b_m & s_m \leq s_c \leq S_M \\
    b_m & s_c \leq s_m
\end{cases}
\end{align}

(4.57)

\(B_M\) is maximum base, \(b_m\) is the minimum base, \(S_M\) is the maximum average speed and \(s_s\) is the minimum average speed. The current average speed is indicated with \(s_c\).

\begin{align}
\begin{cases}
    B_M & s_c \geq S_M \\
    B_2 \sin \left(\frac{\pi}{2} \left(\frac{s_c - S_{mid}}{S_c - S_{mid}}\right)\right) + B_{mid} & s_m \leq s_c \leq S_M \\
    b_m & s_c \leq s_m
\end{cases}
\end{align}

(4.58)

\begin{align*}
B_1 = \frac{B_M - b_m}{2} & \quad S_1 = \frac{S_M - s_m}{2} \\
B_{mid} = \frac{B_M + b_m}{2} & \quad S_{mid} = \frac{S_M + s_m}{2}
\end{align*}

4.4.4.4 Points Accumulation

After each detection loop, either a valid finger-tip location with a semantic flag representing the section of membership or an invalid point with a 0-flag is generated. Both the point and the flag become the argument of the following function, whose task is to attempt the insertion in the dynamic accumulator.

Instead, the function call "updateInformation" keeps the dynamic accumulator state up to date, determining for example if the conditions for extracting a finger-tip are met or updating the cursor’s average speed.
Algorithm 2 Accumulation

if isInvalidPoint(point, flag) then
    queueInvalidPoint()
    discardOldestPoint()
    updateInformation()
    return true
else
    if getInvalidPoints() = getSize() then
        queuePoint(point, flag)
        discardOldestPoint()
        updateInformation()
        return true
    else
        if getError(point) < getErrorThresh() then
            queuePoint(point, flag)
            discardOldestPoint()
            updateInformation()
            return true
        else
            return false
        end if
    end if
else
    return false
end if
Algorithm 3 Update Information

if $\text{getAverageSpeed}() \leq \text{minSpeed}$ then
  setBase(minBase)
else if $\text{minSpeed} < \text{getAverageSpeed}() \leq \text{maxSpeed}$ then
  setBase(getInterpolation(...))
else
  setBase(maxBase)
end if

oldAveragedPoint ← currentAveragePoint
if $\text{getInvalidPoints}() \leq \text{getSize}() \times \text{acceptedInvalidPercentage}$ then
  currentAveragePoint ← computeWeightedAveragePoint()
  queueSpeed($\text{getInstantaneousSpeed}()$)
else
  setInvalid(currentAveragePoint)
  queueSpeed(0)
end if
discardOldestSpeed()
4.5 **Head Tracking**

The second step in our ray forming process is the identification of a second reference point. As shown in the literature, a point moving rigidly with the head is a good practical choice because it implies a reduced signal jitter.

This system is designed so that the contact position on the touch-surface is aligned with the user’s visual line of sight using the head, not the eye, as the reference. This choice leads a user to a more natural and immediate form of interaction, since a constant correspondence with what he sees through the touch-surface is present.

However a one-to-one matching with the user’s finger is not possible because the human vision is stereoscopic. Each eye of a human sees a shifted view of the world. Only afterwards, through a process called ”fusion” performed by the brain, the two disparate images are merged into one. This process results in the perception of an image, where the objects’ depth is more vivid compared to a monocular image.

Although many advantages follow from a stereo-vision, a problem related to the eyes’ fixation occurs in our case. The lines of sight of the user’s eyes converge only onto the object he is fixating on, thus allowing the brain to perform the stereo fusion. Therefore, all the objects located on a plane closer to or further away from the fixation plane appear possibility de-focused and/or doubled.

In our case, because the user’s eyes are both fixated on the distant display, a finger moving on a nearby touch-surface appears doubled in our brain-fused image. In fact, as previously remarked, the finger’s position is closer to the subject than that of the distant display, where
the eyes’ lines of sight converge. Therefore a fused image of the screen object-finger cannot be achieved.

To circumvent this issue, we simply choose a point that moves rigidly with the head and that is coherent with the desired sight alignment. We established this point to be the “cyclopean” center of projection, the mid-point between the two eyes’ center of projection.

In our system we exploit the built-in Kinect’s algorithm to track body sections developed by M. Cook et. al. (33), since its implementation is guaranteed to be real-time. In their work, the authors adopt an object recognition approach of the body’s segments, on a per-pixel basis, and then extract the body’s skeleton joints using a confidence score. They train a random forest classifier with a huge synthetic dataset, so to obtain a model as general as possible for unseen poses. Moreover, the classifier is implemented in the Kinect’s hardware to obtain the real-time performances. The only requirement for a new user appearing in the scene is to perform a body-size calibration.

Since in our work we are not interested in the whole body’s parts, we only use the head position returned among the skeleton joints. Future developments might exploit the other body’s joints for a richer form of interaction, like a full-body gesture recognition.

4.5.1 Cyclopean Center of Projection

The head-point returned by the Kinect’s tracking algorithm does not represent the cyclopean center of projection but only an estimated location inside the head. Therefore a compensation to improve the result is desired.
In this work we propose a procedure to automatically compute the required correction. Our technique, however, only attempts to compensate the translation error, since no information can be retrieved from the head tracking algorithm about the head’s orientation. Future works might adopt better tracking solutions, maybe enabling directly the eye tracking.

Figure 59: Eye’s COP estimation

To compute the compensation required for the cyclopean center of projection, we first estimate the actual center of projection of either eye for a fixed head position. Then the
compensation is given by the difference between the mid-point between the two eyes’ center of projections and the same fixed head position retrieved with the tracking algorithm.

\[ P_{\text{comp}} = P_{\text{head}} - P_{\text{eye}} \] (4.59)

To estimate a single eye’s center of projection, a cost function’s minimization is used again (Equation 4.60). We define an eye’s center of projection as the point in world-space that minimizes a mean squared error, where the error \( \varepsilon_k \) is its orthogonal distance from a k-line of sight’s ray (Equation 4.61)

\[
\min \left\{ J = \frac{1}{N} \sum_{k=1}^{N} \varepsilon_k^2 \right\} \quad (4.60)
\]

\[
\varepsilon_k = \sqrt{\| \vec{P}_k - \vec{P}_0 \|^2 - \left( \frac{(\vec{P}_k - \vec{P}_0) \cdot \vec{P}_D}{\| \vec{P}_D \|} \right)^2} \quad (4.61)
\]

With the head fixed and one eye closed, assuming negligible saccadic movements, a line of sight’s ray is identified by two points, a point on the external display and its visually corresponding point on the touch-surface, as shown in Figure 59. Computing and zeroing out the derivatives of the cost function (Equation 4.62), we obtain a linear system to be solved with the form \( AX = b \), where \( A \) is the Equation 4.63 and \( b \) is the Equation 4.64. For the nature of the problem, this linear system has a unique solution, the best estimation for the eye’s center of projection.
\[
\frac{\partial J}{\partial x} = \sum_{k=1}^{N} \left[ 2 \left( x - x_0^{(k)} \right) \left( 1 - \left( x_d^{(k)} \right)^2 \right) - 2 \left( y - y_0^{(k)} \right) x_d^{(k)} y_d^{(k)} - 2 \left( z - z_0^{(k)} \right) x_d^{(k)} z_d^{(k)} \right] = 0
\]
\[
\frac{\partial J}{\partial y} = \sum_{k=1}^{N} \left[ 2 \left( y - y_0^{(k)} \right) \left( 1 - \left( y_d^{(k)} \right)^2 \right) - 2 \left( x - x_0^{(k)} \right) x_d^{(k)} y_d^{(k)} - 2 \left( z - z_0^{(k)} \right) y_d^{(k)} z_d^{(k)} \right] = 0
\]
\[
\frac{\partial J}{\partial z} = \sum_{k=1}^{N} \left[ 2 \left( z - z_0^{(k)} \right) \left( 1 - \left( z_d^{(k)} \right)^2 \right) - 2 \left( x - x_0^{(k)} \right) x_d^{(k)} z_d^{(k)} - 2 \left( y - y_0^{(k)} \right) y_d^{(k)} z_d^{(k)} \right] = 0
\]

\[
A = \begin{pmatrix}
\sum_k \left( 1 - \left( x_d^{(k)} \right)^2 \right) & -\sum_k x_d^{(k)} y_d^{(k)} & -\sum_k x_d^{(k)} z_d^{(k)} \\
-\sum_k x_d^{(k)} y_d^{(k)} & \sum_k \left( 1 - \left( y_d^{(k)} \right)^2 \right) & -\sum_k y_d^{(k)} z_d^{(k)} \\
-\sum_k x_d^{(k)} z_d^{(k)} & -\sum_k y_d^{(k)} z_d^{(k)} & \sum_k \left( 1 - \left( z_d^{(k)} \right)^2 \right)
\end{pmatrix}
\]

\[
b = \begin{pmatrix}
\sum_k \left( 1 - \left( x_d^{(k)} \right)^2 \right) x_0^{(k)} y_0^{(k)} & -\sum_k y_0^{(k)} x_0^{(k)} & -\sum_k z_0^{(k)} x_0^{(k)} \\
-\sum_k y_0^{(k)} x_0^{(k)} & \sum_k \left( 1 - \left( y_d^{(k)} \right)^2 \right) y_0^{(k)} & -\sum_k z_0^{(k)} y_0^{(k)} \\
-\sum_k z_0^{(k)} x_0^{(k)} & -\sum_k z_0^{(k)} y_0^{(k)} & \sum_k \left( 1 - \left( z_d^{(k)} \right)^2 \right) z_0^{(k)}
\end{pmatrix}
\]

To collect the pairs of points forming the line of sight’s rays, we display a checker-board on the external display and ask the user to touch the correspondent point for each corner on the touch-surface according to his monocular vision. The space locations of the displayed checker-board’s corners are easily computed subdividing the screen’s edges estimated during
the calibration. The touch-surface locations, instead, are the projections of the finger positions corresponding to the checker-board corners on the surface plane previously estimated.
4.6 View-port Mapping

The cursor location on the remote display is identified with a ray-casting approach. When the user interacts with the touch-surface, a ray is built using the point located rigidly on the head, corrected so to have the cyclopean center of projection and the finger moving on or nearby the surface. The point on the display is then given by the intersection between this ray and the display’s plane equation, resulting from the formula Equation 4.65. $\vec{P}_{\text{plane}}$ is a point on the display’s plane, $\vec{P}_{\text{head}}$ is the head location, $\vec{P}_{\text{fing}}$ is the finger location and $\vec{N}$ is the display’s normal.

$$\vec{P}_x = \vec{P}_{\text{head}} + \left[ \frac{\left( \vec{P}_{\text{plane}} - \vec{P}_{\text{h}} \right) \cdot \vec{N}}{\left( \vec{P}_{\text{fing}} - \vec{P}_{\text{head}} \right) \cdot \vec{N}} \right] \left( \vec{P}_{\text{fing}} - \vec{P}_{\text{head}} \right)$$ (4.65)

Using the display’s corner locations in world-space, we first build and apply a 4x4 affine transformation (Equation 4.66) to reduce an intersected point into the display’s plane bidimensional space. $\vec{C}_1$ is the corner chosen as the reference system’s origin, while $\vec{C}_2$ and $\vec{C}_3$ are the adjacent corners.

Finally, this bidimensional point, obtained by applying (Equation 4.66) and discarding the third coordinate, is converted into a display’s viewport pixel coordinate by mean of a planar homography matrix. A DLT algorithm is applied to perform the 3x3 matrix estimation (28), using the correspondence between the four display’s corners converted in the display’s plane space with (Equation 4.66) and the four viewport’s corner coordinates, extracted from the screen resolution.
The points generated with the ray intersection procedure represent the set of locations of a cursor operating on the external display. Since this work arises only as a proof of concept, we simply drive the absolute position of a mouse pointer using the operating system APIs. However, further researches might lead to the development of different usages, as for example the implementation of applications that can exploit a multi-touch paradigm.

4.6.1 Cursor Super-sampling

The pipeline so far described represents the software component that was previously introduced with the name of "data generator". This generator computes and stores the cursor’s positions into a FIFO buffer, together with a semantic value extracted from the dynamic accumulator data structure, expressing, for instance, the action of a user touching or just moving nearby the touch-surface. The next described software component is our implementation of the "data consumer", the application which retrieves and "uses" the points returned by the data generator in a buffer.

The initial approach we adopted was to simply set the absolute mouse position to every location received from the buffer. Although most of the times this solution is sufficient, an issue with high resolution displays arises. In fact, the difference of resolution between the display...
and the depth sensor causes visible motion gaps in the reproduced mouse trajectory, leaving an unpleasant feeling of discontinuity to the user.

Since the precision problem cannot be solved unless a higher quality depth sensor is used, we focused mainly on hiding this motion discontinuity. To do so, we exploited the knowledge that both the eyes’ sampling rate and the data consumption rate are much higher than the sensor’s generation frequency, being around the 30 fps. Therefore this additional time can be used to fill the motion discontinuity gaps with newly generated points at the cost of a non-perceptible delay.

Figure 60: **Data consumption**
As shown in Figure 60, the data consumer stores the points returned from the data generator in a FIFO data structure, together with a timestamp of generation. Afterwards, these points are augmented by a transformer component, which in our implementation is a line rasterizer, and pushed into another queue with a new timestamp.

The line is subdivided into a set of equidistant points by a recursive procedure commonly called "binary splitting". Starting from the initial two points, the line is first split into two parts finding the mid-point. Then, for each newly generated segment, the procedure is recursively repeated for a maximum number of times, called "split factor". The other stopping condition is when the segment cannot be split anymore, because its two extremes’ coordinates differ only by one pixel.

![Binary split diagram](image)

**Figure 61: Binary split**
When the point retrieved from the buffer is the first of a motion trajectory, the timestamp is set to zero, otherwise the formula (Equation 4.67) is used. $T_2$ is the current timestamp, $T_1$ is the previous timestamp and $N_{rast}$ represents the number of points returned by the line rasterization, from the previous point to the current.

$$T_{pix} = \frac{T_k - T_{k-1}}{N_{rast} - 1}$$ \hspace{1cm} (4.67)

These values are critical because they enable the mouse trajectory to be reproduced coherently with the generation process. In fact, a player thread is assigned to sequentially pop the augmented points from the buffer, pause for the currently specified time and change the
cursor’s coordinates to the read absolute location. The final result is a fluid and continuous trajectory.

The software decoupling of this component allows the developer to freely specify the consuming process as desired. More complex transformations, like special filters or higher order interpolations, might be used but they were out of the scope of this work.
CHAPTER 5

PRELIMINARY STUDY

In this chapter we describe the user study we conducted to perform a preliminary evaluation regarding the suitability of our approach. In particular, the following aspects are presented in detail:

- **Experiment**

  This section describes the experiment performed by the participants involved in this study. Both the candidates selection criteria and the task are presented thoroughly.

- **Results**

  In this section the results obtained from the previously described experimental evaluation are reported and discussed.
5.1 Experiment

A total of 10 healthy subjects participated in this study, and all of them signed a consent form that conformed to federal and University guidelines as approved by the Institutional Review Boards of the University of Illinois at Chicago. Only one subject was female.

5.1.1 Inclusion Criteria

The included subjects in the study met the following criteria:

- Free from any visual or motor pathology or history of joint pain
- Age between 18 and 65 years at the time of study participation

5.1.2 Exclusion Criteria

The study excluded any subject which demonstrated any of these deficits:

- Partial or full body’s mobility impairment (paresis, lack of coordination, mutilations)
- Cognitive impairment (to understand and accomplish the experiment)
- Severe sensory deficits (especially sight/haptic deficits)

5.1.3 Task

The proposed experiment consists of a game in which a participant, controlling a cursor pointer on the remote external display using our interface, has to beat a computer controlled cursor. The objective of the game is to reach a randomly appearing target before the adversary cursor pointer does and as fast as possible.
The user’s cursor, the adversary cursor and the target are all represented by a circle with a fixed radius. In particular, both the user’s and the adversary’s cursors have an active radius of 50 pixels which is much smaller than the target’s radius of 288 pixels.

Each trial is defined as the period of time that starts with the random appearance of the target and ends when the user’s cursor or the adversary’s cursor has reached the target. A target has been reached when the circle associated with a cursor is completely within the target area or, using formulas, when $||\vec{c}_1 - \vec{c}_2|| < r_{targ} - r_{curs}$, where $\vec{c}_1$ and $\vec{c}_2$ are respectively the
center of the cursor and the center of the target, $r_{targ}$ is the radius of the target and $r_{curs}$ is the radius of the cursor.

Each subject performs the same 150 consecutive trials. When a target has successfully been reached either by the user’s cursor or the adversary’s cursor, the next target appears at a fixed distance of 2826 pixels from the previous one, as well as the adversary’s position is set such that it starts at the same distance of the user’s cursor.

For each trial, the adversary’s cursor follows an Hermitian curve trajectory to the target and is set to reach it in a fixed amount of time. $t$ is a numeric parameter, ranging from 0 to 1 and obtained with the formula $t_{current} = \frac{t_{current}}{t_{total}}$, where $t_{current}$ is the time in seconds passed from when the trial started and $t_{total}$ the time in seconds required to the adversary’s cursor to reach the targets. $P_{adv}$ is the position of the adversary’s cursor given $t$, $P_{start}$ is the adversary’s cursor start position, $P_{end}$ is the adversary’s cursor end position.

$$P_{adv}(t) = (2t^3 - 3t^2 + 1) P_{start} + (-2t^3 + 3t^2) P_{end} +$$

$$+ (t^3 - 2t^2 + t) (P_{end} - P_{start}) + (t^3 - t^2) \left( \frac{P_{user} - P_{start}}{||P_{user} - P_{start}||} \right) 30$$

The time for the adversary to reach the target represents a “level of difficulty” for the task. We set the game’s level of difficulty to adapt to the user’s performance, meaning that if the user wins 3 times in a row the adversary’s time to reach the target decreases. Vice-versa, if the user loses 3 times in a row the adversary’s time increases. The game starts with the time
required by the adversary to reach the target set to 3.0 seconds. The step used to increase or
decrease the level of difficulty is 0.1 seconds.

To implement this adaptive mechanism, we use a counter variable which initially is set to 3.
Each time the user wins, this variable is incremented by 1 unit. Alternatively, when the user
loses, the variable is decremented of 1 unit. If the counting variable reaches the value of 6, the
level of difficulty increases and thus the adversary time is reduced of 0.1 seconds. Instead, if the
counting variable reaches the value of 0, the level of difficulty decreases and the adversary time
is increased by 0.1 seconds. In both cases, when the level of difficulty changes, the counting
variable is restored to the value of 3.

The purpose of this task is to identify the level of difficulty the user can possibly reach
with our interface and show the goodness of our proposal for an interaction with a large-sized
high-resolution display.

5.2 Results

5.2.1 Game Performances

Figure 64 represents the trend of the subjects’ reached level of difficulty. The horizontal
axis is the trial number while the vertical axis is the time the adversary’s cursor is set to reach
the target.

In the first 70 trials the curve decreases smoothly for all the participants, meaning that the
level of difficulty is below the subjects’ performances. In other words, the user wins with no
difficulties and constantly, versus an adversary tuned to reach the target in a time greater than
1 second.
When the 1-second level of difficulty is passed, each participant starts to be challenged more seriously by the adversary. In particular, from Figure 64 it can be inferred that the subjects’ performances reach an asymptotic value which, as it will be shown in the next sections, floats between 0.6 and 0.9 seconds.
5.2.2 Movement Time

Figure 65 shows respectively the mean and the median of the movement time performed on the 150 trials, while Figure 66 is the variance. Their numeric values are reported in seconds in Table I and Table II.

The red bar in the histogram is the mean time computed with no discrimination on all the available data. On the contrary, the blue bar is the mean time but filtered from outlier elements. We determined the outliers checking the condition \((MT < q_1 - kIQR \quad OR \quad MT > q_3 + kIQR)\), where \(MT\) is the movement time we are currently checking, \(q_1\) and \(q_3\) are respectively the first
Figure 66: Movement time - variance

and the third quartile, $IQR$ is the interquartile range, defined as the difference $q_3 - q_1$ and $k$ is an experimental parameter we set to 1.5 to achieve a mild filtering.

Assuming normally distributed data, we also computed a 95% confidence interval of the mean for each subject, shown in Figure 67 for the filtered data and in Figure 68 for all the data. Since the variance is unknown, we used t-distribution based confidence intervals. The graphs show that the intervals generated from the filtered data are narrower and thus more accurate.

We also performed a one-way ANOVA with a significance level $\alpha = 95\%$ of the subjects’ trials. An unbalanced test was performed to test if the means of each participant’s trials were statistically different or not. The resulting p-value of $1.42e-55$ rejects the null hypothesis in which all the means are equals. As suggested by Figure 67, there is a statistical difference among the subjects’ performances.
## Table I: Filtered Data

<table>
<thead>
<tr>
<th>Subject</th>
<th>Mean MT</th>
<th>Median MT</th>
<th>Variance MT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.724704935</td>
<td>0.712949</td>
<td>0.023677503</td>
</tr>
<tr>
<td>2</td>
<td>0.656823951</td>
<td>0.644533</td>
<td>0.011749926</td>
</tr>
<tr>
<td>3</td>
<td>0.707937521</td>
<td>0.6999935</td>
<td>0.009885778</td>
</tr>
<tr>
<td>4</td>
<td>0.54187401</td>
<td>0.545287</td>
<td>0.002778742</td>
</tr>
<tr>
<td>5</td>
<td>0.761032963</td>
<td>0.781527</td>
<td>0.068004263</td>
</tr>
<tr>
<td>6</td>
<td>0.635414061</td>
<td>0.6327435</td>
<td>0.006384747</td>
</tr>
<tr>
<td>7</td>
<td>0.739024029</td>
<td>0.720475</td>
<td>0.022650249</td>
</tr>
<tr>
<td>8</td>
<td>0.792539693</td>
<td>0.767317</td>
<td>0.029209285</td>
</tr>
<tr>
<td>9</td>
<td>0.843416916</td>
<td>0.842548</td>
<td>0.028844175</td>
</tr>
<tr>
<td>10</td>
<td>0.863765616</td>
<td>0.79749</td>
<td>0.05625611</td>
</tr>
</tbody>
</table>

## Table II: Non-Filtered Data

<table>
<thead>
<tr>
<th>Subject</th>
<th>Mean MT</th>
<th>Median MT</th>
<th>Variance MT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.737745898</td>
<td>0.713227</td>
<td>0.041823425</td>
</tr>
<tr>
<td>2</td>
<td>0.682573482</td>
<td>0.647197</td>
<td>0.049875268</td>
</tr>
<tr>
<td>3</td>
<td>0.7839322</td>
<td>0.718338</td>
<td>0.077629727</td>
</tr>
<tr>
<td>4</td>
<td>0.583752155</td>
<td>0.545287</td>
<td>0.032175537</td>
</tr>
<tr>
<td>5</td>
<td>0.803500432</td>
<td>0.785213</td>
<td>0.116143739</td>
</tr>
<tr>
<td>6</td>
<td>0.699365288</td>
<td>0.635714</td>
<td>0.053780825</td>
</tr>
<tr>
<td>7</td>
<td>0.797274866</td>
<td>0.736541</td>
<td>0.05996133</td>
</tr>
<tr>
<td>8</td>
<td>0.887522315</td>
<td>0.777772</td>
<td>0.127207688</td>
</tr>
<tr>
<td>9</td>
<td>0.867524173</td>
<td>0.8496985</td>
<td>0.04943375</td>
</tr>
<tr>
<td>10</td>
<td>0.935527872</td>
<td>0.824098</td>
<td>0.170364817</td>
</tr>
</tbody>
</table>
Figure 67: Confidence intervals - filtered data

Figure 68: Confidence intervals - non-filtered data
### TABLE III: CONFIDENCE INTERVAL FOR MEAN - FILTERED DATA

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6952</td>
<td>0.7542</td>
</tr>
<tr>
<td>2</td>
<td>0.6355</td>
<td>0.6781</td>
</tr>
<tr>
<td>3</td>
<td>0.6878</td>
<td>0.7281</td>
</tr>
<tr>
<td>4</td>
<td>0.5312</td>
<td>0.5525</td>
</tr>
<tr>
<td>5</td>
<td>0.7111</td>
<td>0.811</td>
</tr>
<tr>
<td>6</td>
<td>0.6194</td>
<td>0.6514</td>
</tr>
<tr>
<td>7</td>
<td>0.7095</td>
<td>0.7686</td>
</tr>
<tr>
<td>8</td>
<td>0.7588</td>
<td>0.8263</td>
</tr>
<tr>
<td>9</td>
<td>0.8109</td>
<td>0.876</td>
</tr>
<tr>
<td>10</td>
<td>0.8165</td>
<td>0.9111</td>
</tr>
</tbody>
</table>

### TABLE IV: CONFIDENCE INTERVAL FOR MEAN - NON-FILTERED DATA

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6987</td>
<td>0.7768</td>
</tr>
<tr>
<td>2</td>
<td>0.6404</td>
<td>0.7248</td>
</tr>
<tr>
<td>3</td>
<td>0.7313</td>
<td>0.8366</td>
</tr>
<tr>
<td>4</td>
<td>0.5499</td>
<td>0.6176</td>
</tr>
<tr>
<td>5</td>
<td>0.7394</td>
<td>0.8676</td>
</tr>
<tr>
<td>6</td>
<td>0.6557</td>
<td>0.743</td>
</tr>
<tr>
<td>7</td>
<td>0.7514</td>
<td>0.8431</td>
</tr>
<tr>
<td>8</td>
<td>0.8204</td>
<td>0.9546</td>
</tr>
<tr>
<td>9</td>
<td>0.8255</td>
<td>0.9095</td>
</tr>
<tr>
<td>10</td>
<td>0.8572</td>
<td>1.0139</td>
</tr>
</tbody>
</table>
The Post-hoc Tukey-Cramer tests, for example, show that subject 4 performed significantly better than all the other participants in the experiment. Table V reports the 95% confidence intervals for $\mu_A - \mu_B$, where $\mu_A$ is the actual mean of subject A and $\mu_B$ is the actual mean of subject B.

5.2.3 Discussions

The results reported in the previous section represent a preliminary study aimed at understanding the goodness of our interface. In particular, we wanted to evaluate the potential of this invention with a simple but of commonly use task, like the control of a cursor in a game.

These results are undoubtedly promising: Table I and II show that the average movement time per user ranges between 0.6 seconds and 0.9 seconds, with a very low data variability. The change in terms of variance is caused by the presence of few outliers, present in the data due to either the user’s errors or the tracking device’s errors.
In this experiment we also chose a task having a high number of movements to better understand the stress caused by our interface to the user. We thought that 150 consecutive reaching trials were a sufficient number. Although Figure 64 shows that the user can compete with an adversary that reaches the target in 0.5 - 0.6 seconds, it also shows that this level of performance cannot be kept for very long time. In fact, when the experiment is almost finished, the user starts to feel tired and thus his performances slightly drop to a value of 0.8 seconds.
CHAPTER 6

CONCLUSIONS

This work described the development of a next generation computer interface that will
promote a natural and versatile interaction with large, high-resolution display systems. The
features of our new interface provide the user with the ability to choose an object on a remote
screen by merely pointing his finger at the object and then performing tasks, such as moving the
same object to a new desired location on the screen. This new interface for large high-resolution
computer displays will make interactions more intuitive and less confusing than if we merely
apply our current technology to these new systems.

We have invented an interface for coarse-grained interaction with a large-sized, high-resolution
display which uses a distant interaction technique called remote pointing. As described thor-
oughly in the previous chapters, remote pointing is a technique that allows the user to interact
with the display from an appreciable distance. Driven by recent investigations, we placed a
transparent surface between the user and the display and transformed it into a touch-sensitive
space using our algorithms. This surface acts as a dynamic touch-screen, meaning that its
mapping to the remote screen is not static but changes according to the users position, so to
allow the cursors location to remain aligned with the users line of sight.

The system consists of a tracker, and a transparent surface that the user operates to interact
with objects shown on the remote display. To date we have a working prototype assembled
from spare parts found in our laboratory (Plexiglas surface, a framework to mount various
components for this device) and a Microsoft Kinect camera tracking system that we share with others. The software to control the objects displayed on the screen through user inputs works reasonably well, as shown in the previously presented results. Moreover, user tests are already being conducted to better understand the performance of our solution, compared to the other available technologies. Due to a lack of available time we were not able to include all of them in this thesis.

Unfortunately, the precision limitations caused mainly by the use of a low resolution tracker led to a final result which is not as good as expected from our theoretical models. In addition, the current hardware needs to be upgraded and a stable hardware system (where vital components need to be purchased since they are being shared with others) needs to be built using better components than we are currently using. At this point in its development we are evaluating whether it would be better to subcontract the construction of a beta version of our demonstration to a commercial entity or if it should be done in-house. We will have to examine these options further since we are now at the point where a more robust system will be needed to prove its viability to potential industry partners.

Growing utility and lower cost for a cluster of many smaller high-resolution displays to produce a very large high-resolution (100 million pixels) wall display are rapidly penetrating the academic, DoD, and some technologically advanced companies offices. Ultimately they will also arrive in homes in large numbers. As these large-screen systems penetrate the market, the simplicity and ease of use that are characteristics of our new interface system will be the
preferred way to interact with such large remote display systems. The demand for such easy to use interfaces like ours will steadily increase.

6.1 Future Works

This work represents a first solid step towards the creation of a new generation interface for coarse grained interaction with a large-size high-resolution display. Many are the possible future developments which can be performed to improve and add features to our invention, as well as to modify our concept to fit future uses.

A first important required modification, which has already been scheduled for the near future, is using a different and more suitable set-up for the interface. Since our target was the production of a working prototype, we only used spare parts found in our laboratory.

• Plexiglas Surface

We found a large transparent surface made of Plexiglas unused in the laboratory and we used it because theoretically this material allows the infra-red light waves to pass. Unfortunately many problems arose from the choice of this surface that we did not expect initially. First of all, an infra-red ray of light which is orthogonal to the Plexiglas generates a disturbing reflection on the surface. The tracking device erroneously detects this reflection as a physical presence nearby the point of orthogonal incidence, instead of being transparent. For this reason the camera was located above and on the left of the Plexiglas surface, with a downward pointing direction. That way the point of orthogonal incidence was isolated to the left side of the camera’s view-port and therefore the reflection problem was neutralized. The drawback of this solution is that there is a waste of
available resolution because not all the camera’s view-port areas can be used to detect the finger presence.

- **Plexiglas Supports**

Another flaw arising from this choice is represented by the elasticity properties of this specific material. Differently from other materials, this surface bends very easily when the user presses his finger strongly against it because the support does not hold it tightly enough. This problem is particularly remarked, because we adapted two vertical supports so to make a standing Plexiglas surface, with the central part completely free from any reflective material. To improve the result a material as rigid as glass should be used.

- **Tracking Device**

Driven by the desire of building a low-cost solution, we opted for a Microsoft Kinect tracking camera device. Even though its price is very desirable, the low resolution has a great impact on the final result. In fact, we ended up exploiting only a reduced amount of the total available resolution, due to the reflection problems we have described previously. Moreover, also the particular depth estimation technique, implemented by the device, presents some relevant pitfalls. The measured values read from the sensor become worse as the camera’s distance increases from the location we want to estimate. For a distance close to 5 meters the resolution is only 5 centimetres.

Many possible future developments can also be performed aiming to create new applications for our interface. Since 3D displays are becoming a pervasive phenomenon, an interesting
study to conduct would be the development of an interaction solution for a three-dimensional stereoscopic content. The remote-pointing ray-casting based approach we used turns out being very flexible because it can be generalized to intersect also objects which are not planar, like the screen-display. This form of interaction becomes certainly more interesting because a user can have different three-dimensional views of the objects he is interacting with. By simply moving his head, he can see objects that were previously occluded by others and select them, exploiting the previously presented line-of-sight based interaction.

Even though good results are obtained in coarse-grained tasks with large-sized high-resolution displays, our approach might also be studied in completely different contexts. For instance, alternatively to a physical display, the image could be projected directly onto a room's empty wall. With this choice, the system becomes cheaper and can benefit of an increased mobility, since the hassle of moving a projector is much smaller compared to that of mounting a high-resolution display into a new position. However the price to pay is in terms of a significant resolution drop, caused by the adoption of the light projector.

Another possibility could be changing the current touch-surfaces calibration that is fixed, with a dynamic solution. The introduction of sensors mounted on the surface can enable a real-time location tracking in the cameras working position, thus allowing the user to easily change his point of view without recalibrating.

Instead, from the point of view of the touch-interaction, the single finger paradigm could evolve into a multi-touch paradigm, where more than one blob is being tracked on the surface. This involves being able to distinguish between different fingers' trajectories, without introduc-
ing considerable performance losses. Besides, richer forms of interaction become possible using multiple fingers, like for example the recognition of special gestures.
CITED LITERATURE


27. Project, R.: Depth linearization.


VITA

• Name
  Antonio Faraldo

• Education
  B.Sc. Computer Engineering, Politecnico di Milano, Milan, Italy, 2009