HaptiQ: A Haptic Device for Graph Exploration by People with Visual Disabilities

University of St Andrews

Author:

Simone Ivan Conte

Supervisor:

Dr. Miguel A. Nacenta

April 4, 2014

0.1 Abstract

Today's technology is not as accessible for blind and visually impaired people. Overcoming this digital divide gap for people with visual disabilities is one of today's most overlooked challenges. Current technology provides either expensive or non-flexible devices. This dissertation investigates both commercial devices commonly used by visually impaired people, and the state of the art of haptic feedback devices.

This project presents the first vector-based display with haptic-audio feedback for people with visual disabilities: the HaptiQ. Two different devices of varying resolution were 3D-printed as prototypes. In addition, a general easy-to-use API is provided together with two basic example applications. The first application shows how the HaptiQ can be used to explore simple graphs. The second program instead displays mathematical functions and is used to show how the HaptiQ facilitates edge recognition. The sets of behaviour patterns used so far are heavily based on the use of Braille-like systems.

With this project a new set of dynamic behaviour patterns is visually encoded in a way which favours the recognition of edges, corners and directions.

0.2 Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 11,642 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona de library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Simone Ivan Conte

0.3 Acknowledgments

I would like to thank my supervisor Miguel A. Nacenta for the opportunities, support and motivation he gave me.

Thanks to Saad Attieh and Régis Ongaro-Carcy for their helpful feedback on the project.

I would like to acknowledge the School of Computer Science, the SACHI research group, and SICSA that made this project possible by allowing me to use their facilities and by financing it.

Finally, I would like to thank my parents and my brother for their immense and invaluable love and support, which helped me arrive where I am today.

Contents

List of Figures

Introduction

The use of technology for the blind and the visually impaired is not obvious. The digital divide gap is significant and has an evident impact not only on the life style of the visually impaired, but also on how they interact with others and the surrounding environment. This can lead to one's exclusion from society. Therefore, presenting information in a correct, fast and easy-to-access manner to the visually impaired is a fundamental challenge in a world where technology evolves at an exponential rate.

So far, Braille can be considered the most used form of tactile graphics since the XIX century[9]. Braille is a coding system for alphanumeric symbols, with each braille character being a combination of raised dots or tactile bumps positioned within a fixed grid. While Braille has been shown in the past to be excellent for textual information, the trend is changing. In fact, touch screen devices, which are becoming always more pervasive, do not allow Braille to be reproduced.

In addition to the technology revolution we are experiencing, nowadays, the amount of data daily produced is increasing. Most of the time it is not feasible to represent data and its information in textual form, whether it is a small data set or big data. Therefore, in the last couple of decades data visualisation has been playing a crucial role in many areas, such as economics, physics, and chemistry aiding scholars and professionals to better understand problems and show their results to the public. Graphical visualisation can also help users to better understand more abstract concepts, such as graphs (e.g. automata, trees, etc) and functions behaviour. The progress in information visualisation and presentation has been substantial in the last 20 years, with the increase in computational power and larger displays. However, this progress has not had a similar strong positive impact on the way the visually impaired interact with technology.

It is then necessary to undertake such challenge in order to minimise the digital divide for the visually impaired. Traditional devices include screen readers, like JAWS[1], and dynamic Braille displays[2, 18, 21]. However, these tools are generally very expensive, bulky and lack of flexibility and adaptability. For instance, screen readers cannot be used for graphics, while dynamic displays can represent only a small portion of a display and being able to locate their relative position can become very hard. Currently researchers and industry are increasing their effort toward bridging this gap. This project is based on the work of Marquardt et al.(2009), who developed the Haptic Tabletop Puck (HTP), an inexpensive tactile feedback input device to be used on digital tabletop surfaces. Friction, height, texture, and malleability are communicated through a combination of properties of the HTP: a rod and a brake pad controlled by two servo motors, and a pressure sensor on the rod. The HTP, however, uses only one actuator on a finger to convey information to the user, which makes it unsuitable for use by people with visual disabilities, since directions and edges of tactile objects cannot be detected with ease.

Here I present the HaptiQ (pronounced Haptic Cue), a Tangible User Interface (TUI)[12] with tactile and audio feedback input which takes the HTP to the next level for people with visual impairments. In addition an easy-to-use API and a new set of dynamic Tactons is presented. Tactons are "structured, abstract, tactile messages which can be used to communicate information non-visually"[7].

This project will be available to download under the GNU General Public License in accordance to one of the aims of the project: building an accessible, inexpensive and easy-to-build device for the visually impaired.

1.1 Contributions

The contributions of this project can be summarised as follows:

- Create an inexpensive haptic-audio feedback device for the visually impaired
- Create a device that can be 3D-printed at home
- Create a modular device, allowing further studies to explore different types of tactons

- Design and implement a dual-API (both high and low level) for the device which is easy-to-use
- Project available to the community under the GNU General Public License

1.2 Report Outline

The remainder of this report is structured as follows. **Chapter 2** states the overall objectives of this project in order of importance and relevance. In **Chapter 3** a detailed overview of the related work is presented, in order to get a flavour of the current state of the art and how this project fits in it. The research techniques and methodology used through the development phase of this project is described in **Chapter 4**. **Chapter 5** addresses any ethical considerations. The design and implementation aspects of the HaptiQ, the API and other aspects of this project are described in **Chapters 6, 7**. The current work is evaluated in **Chapter 8** and future work in **Chapter 9**. Finally **Chapter 10** provides a conclusive summary of this project.

Objectives

1. Primary
    (a) Design and implement an haptic device for graph exploration by people with visual disabilities.
    (b) Develop an API to control one or more haptic devices.
    (c) Develop an API to easily develop WPF client applications.
    (d) Design and implement a new set of dynamic behaviours.
    (e) Design and implement a basic application for Graphs exploration.
2. Secondary
    (a) Increase the haptic device resolution.
    (b) Allow the haptic device to provide audio-feedback as well.
    (c) Design and implement an application for simple mathematical functions exploration.
3. Tertiary
    (a) Enable the haptic device to be used collaboratively.
    (b) Enable the haptic device to sense textures.

All the primary and secondary objectives have been achieved.

The secondary objectives, in semester one, included also the implementation of dynamic pointing. This feature, however, was removed for reasons discussed later in the Evaluation chapter.

In addition to the objectives above, I pursued this project with the following other goals in mind:

- Investigate and evaluate the current state of the art.
- Develop a device which fits within the current state of the art.
- Develop a modular device which can be easily adapted to various conditions and needs.

Background

The following chapter presents the state of the art in haptic feedback technologies and the current commercial technology for the visually impaired. The related work taken into consideration is taxonomised based on how the various techniques and devices work and how users interact with them.

Research in haptic perception and visualization has grown significantly in the last 25 years [17]. Being able to cover most of the work in haptics is not feasible. Therefore, only the most significant related work to this project will be addressed.

3.1 Commercial technology for the visually impaired

The devices and techniques currently available on the market and used by most of the visually impaired differs from what researchers have been able to achieve in the last decade. The main reason is that the most advanced technology produced by researchers is either expensive or still a prototype, or both.

Traditional Braille and tactile maps are by far the most intuitive and easy-to-use ways to access textual and graphical information. However, these approaches are static and do not allow access to electronic devices. The resolution of tactile maps is usually low, since font size tends to be large and zoom-in actions are not allowed. Dynamic Braille displays [2, 18, 21] allow traditional techniques to be used for technological devices. However, only a small portion of the display can be covered using these devices and being able to keep track of the current position within the display becomes challenging.

The auditory system for blind people is probably the most important of their senses. Using the tactile system to perceive the surrounding world, in fact, can be hard to learn,

especially for people who become blind in their adult stage of their life. Screen readers, like JAWS[1], or auditory-aiding software, like VoiceOver[3], allow the visually impaired to explore textual information and navigate maps as well. Systems that rely just on audio feedback allow a user to follow lines and directions by emitting sounds at different frequencies. It obviously follows that a user needs to constantly explore the area around a specific point in the map in order to perceive the direction of a line or understanding the shape of an object.

3.2 Multimodal Interactive Maps

An attempt to allow blind people to access geographical maps through digital devices is via multimodal interactive maps (MIMs). MIMs are input-output devices that can make use of different technologies to allow users to access or insert information (i.e Input: speech recognition, touch input. Output: audio, tactile feedback). The work presented by Broke et al [6] is the most recent type of MIM, to my knowledge. People who are visually impaired usually tend to use all their fingers when exploring objects. Therefore, enhancing multitouch input allows users to perceive maps with greater ease. Nonetheless, the maps have to be printed for each new visualisation.

3.3 Tangible User Interfaces with Mechanical Actuators

Haptic feedback devices that use mechanical actuations are becoming increasingly popular. In fact, these are usually more flexible and adaptable to displays of different sizes. They also provide better tactile cues than audio-based techniques and MIMs.

The first device to take in consideration is the Haptic Tabletop Puck (HTP)[14], which the HaptiQ can be considered the successor of. The HTP is a simple, inexpensive and small haptic device that suits particularly well for use on digital tabletop surfaces. The tactile cue, however, is applied through one single point only. The HTP has not been designed specifically for visually impaired people. Therefore, it is not very suitable for more complex tasks such as recognising a line or a direction and follow it.

This type of problem has been observed also with the SensAble PHANToM[15, 24].

Participants were asked to follow one or more lines. But the experiment showed negative results, since having only one point contact did not allowed the participants to well understand the shape of the line, especially at the end points or corners. Similarly to users of VoiceOver, blind people using the PHANToM or the HTP need to guess the direction of lines and eventually randomly explore the surrounding.

Harrison and Hudson [11] propose dynamically changeable physical buttons that use either pneumatic actuation or Shape Memory Materials to enhance displays with tactile feedback. This type of displays is proved to be cheap, easy to produce and effective. They are also convenient for static public displays (i.e. ATMs) or cars panel control, but not for more generic displays, such as tabletops, since specific masks need to be created for each visualisation.

Recently the Tangible Media Group, MIT, has presented the inFORM system, a 2.5D shape display that uses mechanical actuators to provide feedback to the user[10]. Top-down projection and a Kinect are used to provide visual feedback and acquire user input. inFORM provides excellent haptic feedback and equivalent responsiveness, nonetheless the table is bulky and expensive.

3.4 Haptic Feedback by alternative mediums

Recently Bau et al. presented TeslaTouch[5] and REVEL[4], two haptic feedback systems based on electrovibration and reverse electrovibration respectively. The purpose of these systems is to create Augmented Reality Tactile displays. These could be both extrinsic, integrated in the environment, or intrinsic, augmenting the user. A study on visually impaired people, using the TeslaTouch, has also been conducted [23]. Participants found hard to recognise dots, as used in Braille, especially due to the fact that the tactile feedback is perceived only when the finger moves. Straight lines could be recognised, allowing some shapes be perceived (circle, square and triangle), but not with accurate precision. Another limitation of vibrotractile based devices is that it cannot be used by multiple users concurrently.

Other techniques used to improve haptic feedback on touchscreens rely on electromagnetism. FingerFlux [22] uses magnets attached on fingertips to provide “attraction, repul-

sion, vibrations and directional haptic feedback" in a table with electromagnets pixels. The main issue related to FingerFlux is that the resolution of the display cannot be increased over a certain threshold, otherwise the electromagnets interfere with each other. In addition, FingerFlux allows to create horizontal forces, but it is not possible to render sharp surfaces.

UltraHaptics provides multi-point haptic feedback above an interactive surface using acoustic radiation force to generate haptic cues on targets in mid-air [8]. The system has been shown to be fairly accurate and easy to use, yet accuracy degrades quickly as the user's hand is above 200mm.

Haptic feedback can also be transmitted by ejecting air at high pressure. AIREAL is a scalable, inexpensive and practical system that uses air to augment reality of interactive surfaces and environments [19]. AIREAL works well within gaming environments and haptic cues of external environments, but the low resolution limits its use for objects and edge recognition, especially for the visually impaired.

Research and Methodology

The nature of this project is more research than engineering oriented. It follows that common software development practices, such as SCRUM, were not used. Instead, the development process was articulated in three main phases: background research, hardware and software investigation, and hardware and software development. Each of these phases will be discussed below.

4.1 Project phases

The background research phase was necessary to understand what the current state of the technology in haptic devices is. This phase was run mainly in the first two months of the project.

The hardware and software investigation phase consisted in getting familiar with the 3D Printer, the Phidgets API and exploring what possible hardware could be used. For instance, hardware experimentation on Quantum Tunnelling Composite (QTC) was pursued to get pressure input from the device. The final evaluation was negative due to the reduced size of QTC tiles, which do not allow an easy installation on the HaptiQ. I then decided to use basic a Force-Sensing resistor instead.

The hardware and software development phase followed the investigation phase logically, but not necessarily temporarily. Both the hardware and the software presented in this report are the product of an iterative exploratory design process. Weekly meetings with the supervisor were arranged. Each meeting consisted in reporting the work done in the previous week, planning the work for the week after and discussing design problems and features. These brainstorming meetings were fundamental in order to achieve the final

HaptiQ. Monthly meetings with Saad Attieh were arranged to discuss various design choices. I found his feedback invaluable.

One of the aims of this project is to develop an haptic device easy to build with an easy-to-use API. On the software side, the API has been designed having in mind that the code will be used for future work. Therefore, I wrote extensive XML documentation for the API calls and provided abstract instances, where necessary, to allow future users to extend the API. The attached manual contains a basic usage example of the API and the list of the required hardware components to 3D print and assemble the HaptiQ.

4.2 Management software

Version control systems have been used to facilitate the software development of the API and maintain the 3D models of the HaptiQ always up-to-date. The Mercurial version control system, managed by the School of Computer Science, has been used from the beginning of the project until almost the very end. On the 18th March 2014, with Régis Ongaro-Carcy joining the project, I decided to switch to GitHub [1]. The main reasons behind this choice are that Régis does not have access to Mercurial repositories within the School of Computer Science and GitHub provides an issue tracking system that facilitates project work with more than one person.

At the start of the project I used the issue tracker Bugify [2]. However, later in the project development process I stopped using this tool because the project evolved too quickly and Bugify slowed down my process rather than speeding it up.

4.3 Testing

One of the most challenging parts of this project was testing. Being the implemented API highly dependent on both the HaptiQ device (Phidget boards, servos, and pressure sensors) and the input device (table, camera, et cetera) the amount of Unit testing was minimal. This required me to manually ran the API multiple times and under various conditions in order to ensure that the software worked properly. A similar approach was

[1] GitHub. `http://www.github.com/`. [Online; last checked: 03/04/2014].
[2] Frondiz. Bugify. `http://www.bugify.com/`. [Online; last checked: 27/03/2014].

undertaken to test the hardware. I regularly checked every part of the hardware to ensure that nothing was broken and that all pieces were installed correctly. A simple logger was implemented to facilitate debugging.

Ethical Considerations

No user study was pursued, therefore this project does not have any Ethical Considerations and does not require ethics approval.

5.1 Collaborations

This project involved collaborations with two students: Saad Attieh and Régis Ongaro-Carcy.

Saad Attieh, University of St Andrews, joined the project as a collaborator, under the approval of the School of Computer Science of the University of St Andrews, in November 2013. Meetings with Saad were established about every month and he has provided very useful feedback on the device design and tactons.

Régis Ongaro-Carcy, Institut de Recherche en Informatique de Toulouse - IRIT - of the Université de Toulouse, joined the project on the 18th March 2014. Régis is using the HaptiQ as part of his Master's thesis focused on designing a strategic game, based on geographical information, for visually impaired people. He has provided some insightful feedback. While using the API, Régis found some bugs or features that should be added. He reported these issues through the issue tracking system of GitHub.

Design

The HaptiQ aims to solve some of the problems identified in the related work. The hardware was developed using an iterative exploratory process. Therefore the design phases largely overlap with the implementation ones. The API provided has also undergone through many iterations. In this chapter I will first discuss the earliest prototypes. Then the 4-HaptiQ and the 8-HaptiQ hardware designs are illustrated. Finally, a detailed overview of the API is provided.

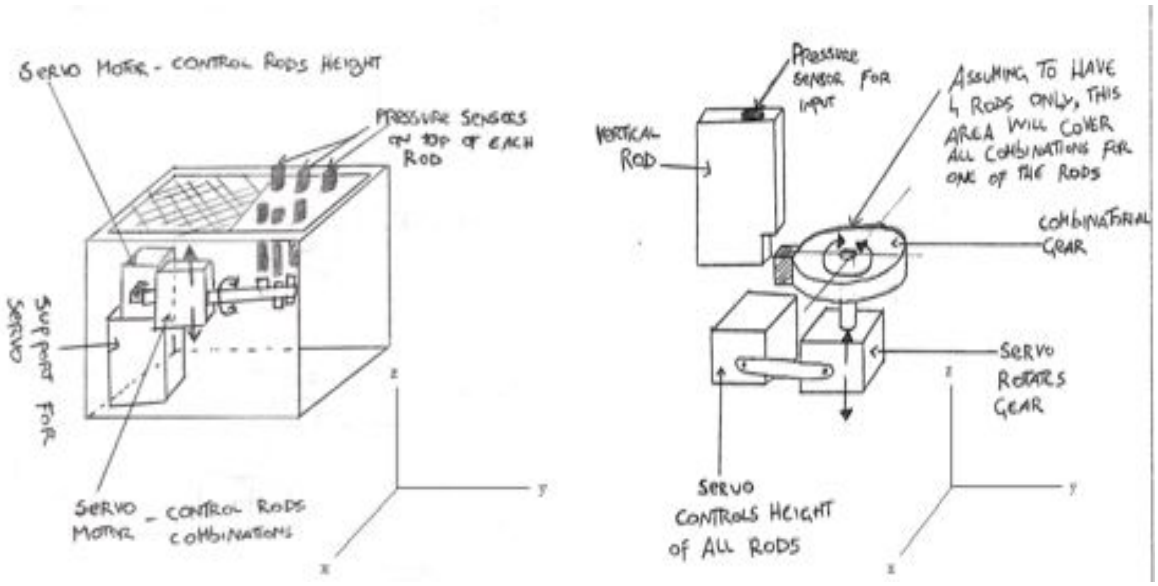
6.1 The Hardware

6.1.1 Early Prototypes

The hardware design of the HaptiQ has been initially based on the HTP design. Similarly to the HTP, the HaptiQ also uses mini-servos to control its actuators. However, while the HTP controls only one actuator, the HaptiQ controls four or eight of them. It immediately follows that as the number of actuators increases, so does the number of servos and the size of the device. In order to minimise the size, I designed three main prototypes by sketches (see Figure 6.1).

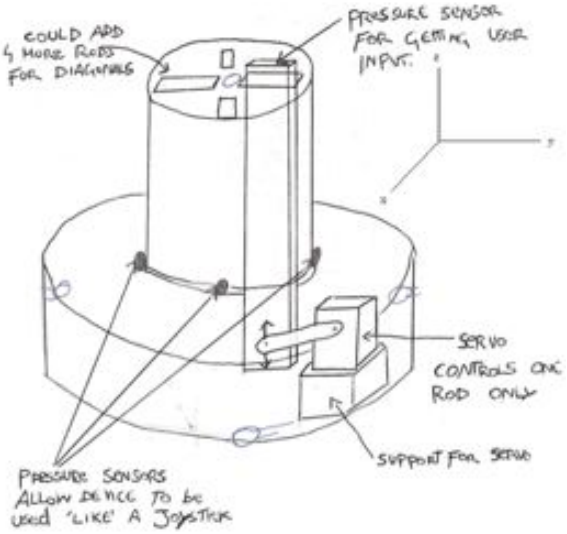
The first prototype extends the rotational axis in the xy plane (see Figure 6.1a). The extension is designed as a gear that allows to choose combinations of three aligned actuators. An additional actuator would then be used to raise the selected actuators. This design reduces the number of servos, but all raised actuators will have the same height. Note that the focus of this design sketch was to minimise the number of servos, that is why actuators are shown as points, similar to Braille displays.

In the second prototype I tried to minimize the total number of servos to two. One



(a) Grid based

(b) Gear based



(c) One servo per actuator

Figure 6.1: HaptiQ early prototypes

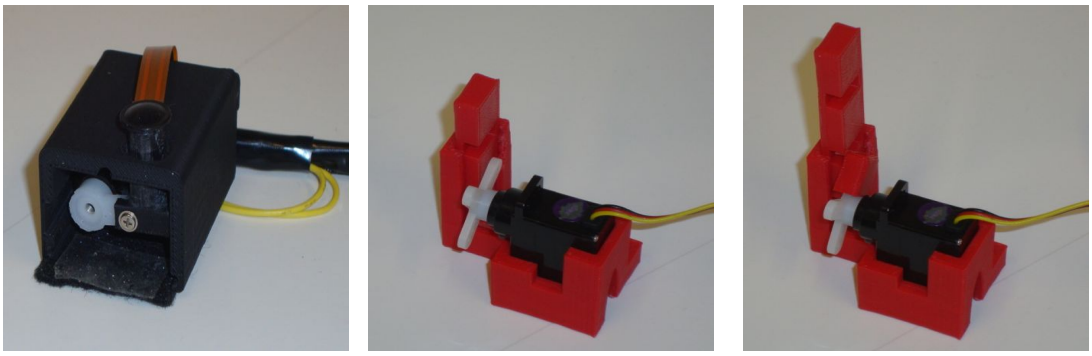
servo would be used to rotate a combinatorial gear about the z-axis (see Figure 6.1b). A second servo can then be used to raise the first servo and all the rods. This approach can be considered very versatile and could allow an high number of actuators by keeping the number of servos always to two. Nonetheless, the actuators would all be at the same height when raised as in the first prototype. Since height can be used to convey additional information, this prototype was discarded and a third prototype was created.

The third prototype does not focus on minimising the number of servos any more, but rather on the possible functions that could be added to the HaptiQ. In this design each servo controls an actuator only and these are disposed circularly around the centre of the device (see Figure 6.1c). The device would have a *joystick* shape. Three pressure sensors could be added either at the bottom of the device or in between the internal and the external annuli. Using simple triangulation on the pressure sensors values, it could be possible to use the device like a gaming joystick and eventually augment its area of interaction. However blind people would find this feature disrupting because understanding how far in the xy plane the device is augmented can become a challenging task.

6.1.2 4-HaptiQ

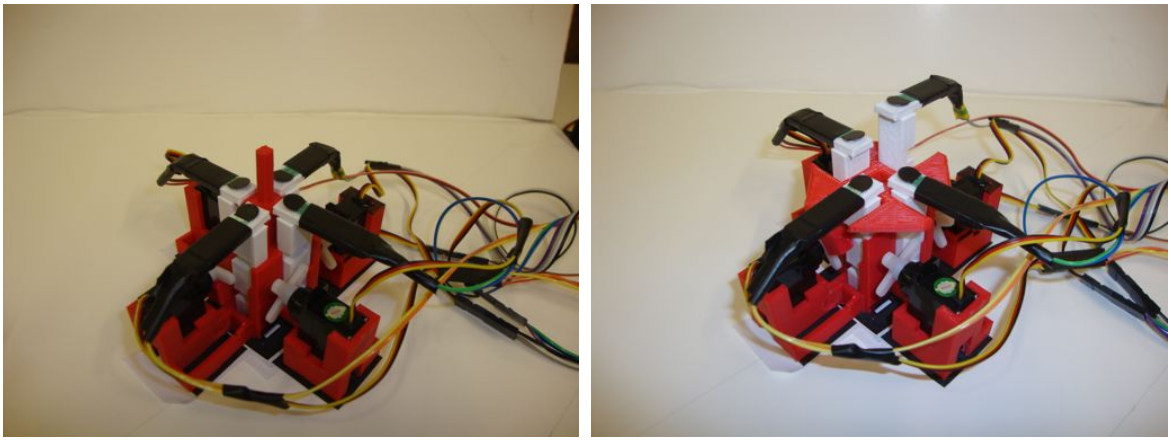
The 4-HaptiQ is the first functional HaptiQ. This is an experimental project, with future work discussed in chapter 9, so I will occasionally refer to the device with the term prototype. This version has four actuators as shown in Figure 6.3. Unlike the third of the early prototypes, the main focus is on the mechanics and functions aspects. The most successful aspect of the HTP, most probably, is the use of a small servo to mechanically move the rod along the z-axis. Nonetheless, the mechanic design used in the HTP has some flows that the HaptiQ tries to solve. The HTP, in fact, transforms the rotational motion of the servo to a linear motion only by ensuring that the servo works under a small range and that the rod slices outside the device through a small opening. In the HaptiQ the actuators slide through a guide that allows motion only in the z-direction. In addition, each servo is linked to its actuator by using one or more screws inserted in between small openings in the actuator (see Figures 6.2b and 6.2c).

This version of the HaptiQ does not provide any case. The uneven surface leads to



(a) HTP (b) HaptiQ MinPos (c) HaptiQ MaxPos

Figure 6.2: HTP and HaptiQ actuator mechanics



(a) Mid-Point reference actuator (b) Plane reference actuator

Figure 6.3: HaptiQ reference static actuators

the lack of a reference point or plane for the actuators. This is a major problem in cases where the actuators change position while the user is not currently laying his hand on the device. In fact, it follows that when the blind user places the hand on top of the device again, he or her is not able to tell what the current state of the actuators is compared to the previous one. Two solutions are provided for the 4-HaptiQ: a mid-point reference and a plane reference static actuators (see Figure 6.3). It is not clear which type of reference actuator works best though. My hypothesis is that the plane reference actuator should be more comfortable when using the device and provides the same functionality than the point reference actuator.

The HaptiQ is the first vector-based display for blind users. Whether this design provides

better feedback to users or not is unknown, because no study with participants could be conducted. Therefore, the actuators are designed so to support different tops in order to be able to evaluate the HaptiQ against a point-based haptic feedback TUI. In the version here presented, only vector-like tops are provided, but other type of tops can be easily printed and fasten to the actuators.

Finally, the HaptiQ does not have any break to simulate friction. While it could be possible to add this feature, I decided that the primary goal of this project was just to focus on the *vectorisation* of the actuators.

6.1.3 8-HaptiQ

The result of the final design iteration is the 8-HaptiQ. Using the same design of the 4-HaptiQ for eight actuators would lead to a considerable increase in size of the device. Therefore, the 8-HaptiQ has been completely re-engineered (see Figure 6.4).

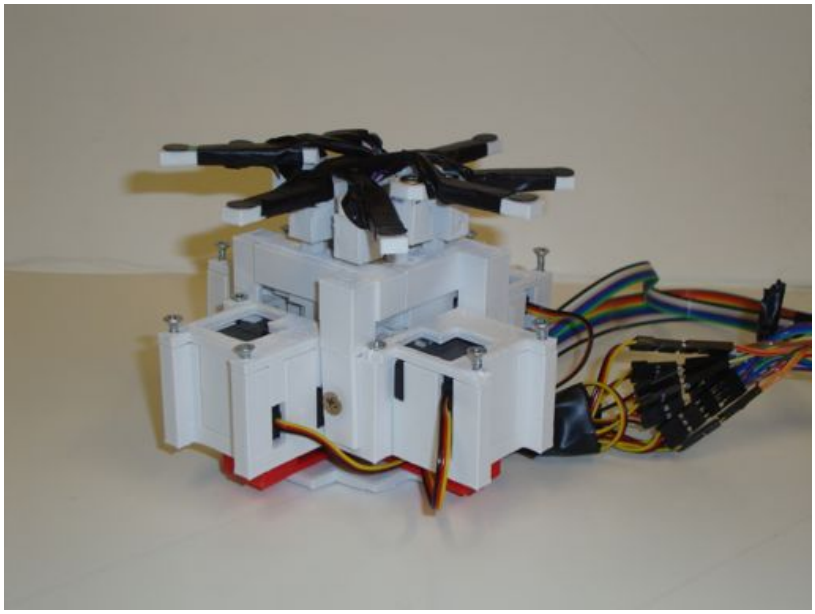


Figure 6.4: 8-HaptiQ

The 4-HaptiQ not only is not optimised in terms of space, but its wiring is also not well engineered. As it is possible to observe in Figure 6.3 the wiring does not allow the device to be used comfortably, especially when one wants to rotate it. The servos in the 8-HaptiQ are organised in couples. Unlike its predecessor, the servos are positioned horizontally (see

Figure 6.5) and the screws are moved more toward the centre of the rotatory mechanism of the servos. These design choices allow the device to be more compact and still preserve the same functionalities of the 4-HaptiQ. The device is lift up by an additional structure laying below it, allowing the wiring to be less disruptive.

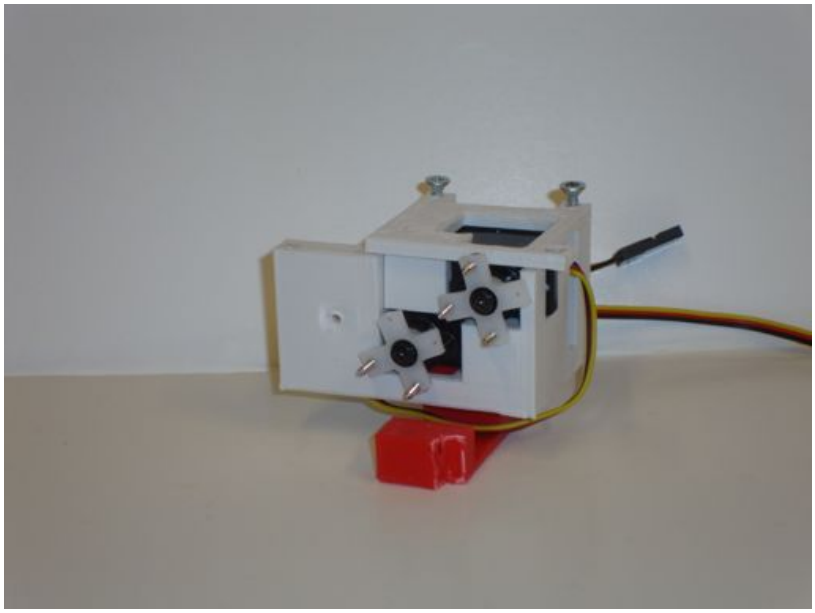


Figure 6.5: 8-HaptiQ servos couple

With the gathering of the wires for the sensors to the center of the device, the position of the sensors is also inverted compared to the 4-HaptiQ. While in the 4-HaptiQ the sensors meet in the middle, with this new version of the HaptiQ they are spread to the outside. It is not clear whether this design choice is an advantage or not in terms of usability. Alternatives to this solution will be provided in the near future, as discussed in the Future Work chapter.

The 8-HaptiQ takes a different approach for the design of the actuators guides. In the first version, a base was provided for the actuator, which limits this to move further down when this is at its minimum position (see Figure 6.2b). But to provide an optimal wiring, the 8-HaptiQ combines the guides for all the actuators with the reference plane (see Figure 6.6). Additional connecting screws allow the actuators to be always controlled by the servos and not to fall below its minimum position.

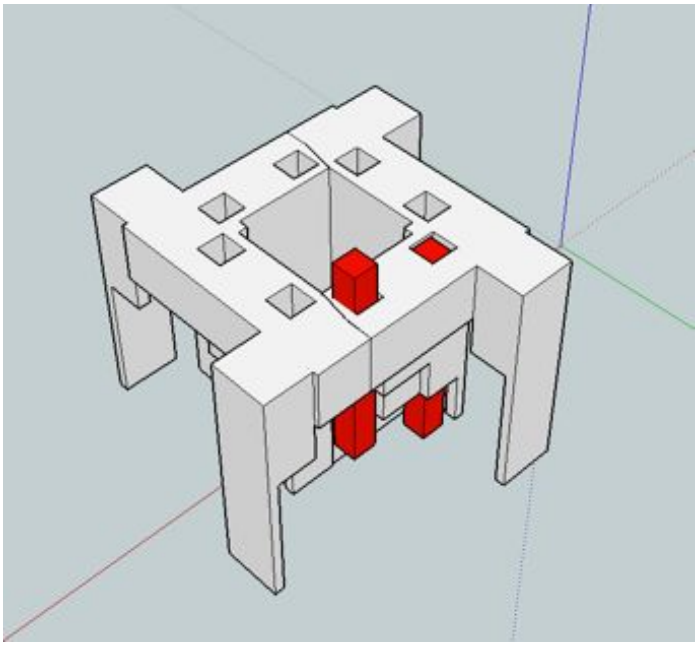


Figure 6.6: 8-HaptiQ reference plane and actuators slots

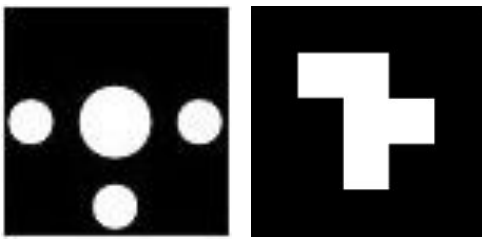
6.1.4 Fiducial Markers

The HaptiQ, like the HTP, is designed to work on interactive tabletops, specifically on Microsoft PixelSense tables (supporting Microsoft Surface SDK 2.0) [3]. The Microsoft Surface Bytetag is the standard fiducial marker for the PixelSense tables (see Figure 6.7a). But Bytetags are hard to print, if high accuracy is wanted, and expensive, if one wants to buy them from the official store.

The HaptiQ, in addition to the Microsoft Bytetags, supports glyphs, as defined by the Glyph Recognition And Tracking Framework (GRATF)[4] (see Figure 6.7b).

Both types of fiducial markers are to be displaced on the bottom of the device, if a PixelSense table (or equivalent) is used. Otherwise, it could be possible to place a glyph on top of the device, provided that an appropriate structure is built, and recognise it using a web-cam.

[3]Microsoft. `http://www.pixelsense.com/`. [Online; last checked: 26/04/2014].
[4]GRATF. `http://www.aforgenet.com/projects/gratf/`. [Online; last checked: 04/04/2014].



(a) Bytetag (b) Glyph

Figure 6.7: Fiducial Markers

6.2 The API

This section discusses the design choices taken for the API of the HaptiQ. The general idea is similar to the HapticTouch toolkit [13]. However, this has been completely redesigned to support:

- multiple actuators
- multiple pressure sensors
- a completely new approach to HapticObjects and HapticBehaviours
- custom actions called when input is received by the user
- additional API to get input position of the device from different hardware

Each of these will be discussed in the following sections.

6.2.1 Overall Structure

The API consists of two main components: the HaptiQ API used to control the device and the Input API (see Figure 6.8). The HaptiQ API component is designed to provide high and low access to the HaptiQ. This allows the API to be versatile and to be used by beginner, intermediate and advanced developers. The HaptiQ API retrieves the position of the device through the Input API.

The following two sections describe these components in more details.

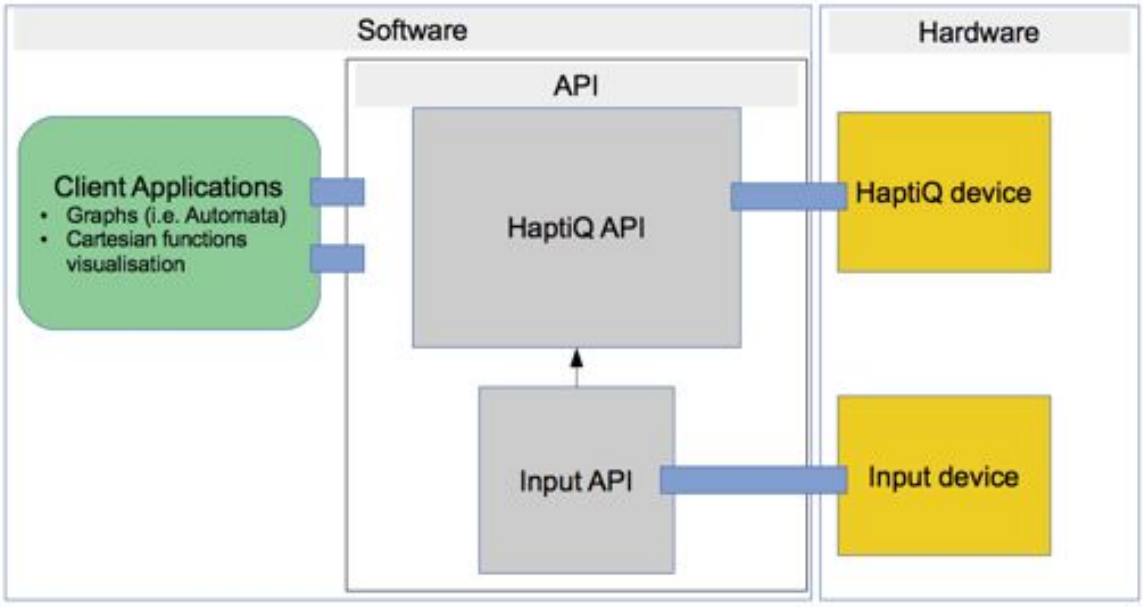


Figure 6.8: API General Overview

6.2.2 Input API

Initially, the HaptiQ, like the HTP, was designed to work on a Microsoft PixelSense table only. Microsoft Surface Bytetags are located underneath the HTP to get the location of the device in the table. However, official Bytetags are expensive and printing them with a normal printer does not lead to satisfying results. In addition, the HaptiQ will be used in the near future on digital tables that do not support the Surface SDK. Therefore, in order to abstract the HaptiQ API from any hardware input dependence, a separate API component has been created (see Figure 6.9).

The architectural structure of the Input API is based on the classic Factory pattern. This adds a level of abstraction over the hardware that should be used to get the position of the HaptiQ. Currently the API supports Bytetags and glyphs recognition. But client applications can also add their own customised class to support different position detection methods.

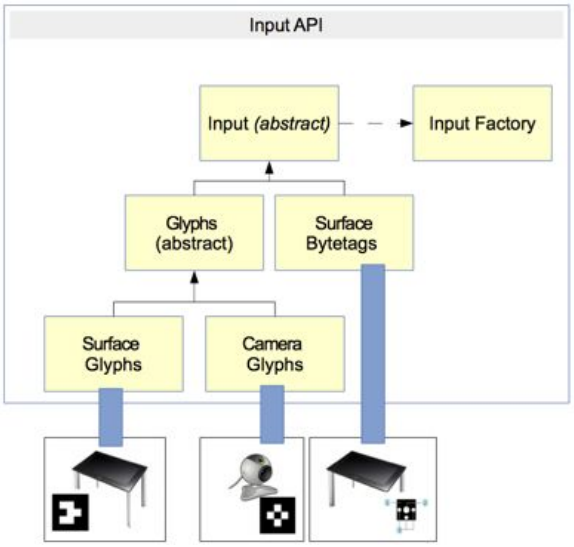


Figure 6.9: Input API

6.2.3 HaptiQ API

The HaptiQ API plays the main role within the API by connecting the Input API, the HaptiQ device and the client applications all together (see Figure 6.10).

The HaptiQsManager is designed using the Singleton pattern, ensuring that only one instance exists, and controls the different parts of the internal structure of the HaptiQ API. On creation, the manager launches the configuration manager. At this stage, the API looks in the current directory for pre-saved configurations of HaptiQs. If there are not valid configurations matching the current attached devices to the machine, a configuration windows form is started, allowing users to set up devices. Otherwise, HaptiQ objects are created for each configured device.

Each HaptiQ is a thread process on its own, which runs the current behaviours cyclically. The HaptiQ class is responsible both for controlling the actuators and getting the input via the pressure sensors.

The manager, however, is responsible to get the devices position through the Input API. Using the observer pattern, the HapticObjects registered by the client applications are notified whenever a new input is received. If the input is relevant to a particular HapticObject, then a behaviour is returned to the HaptiQ device that caused it.

The API allows behaviours to be added, removed, and substituted. This means that

the current state of a device can be the combination of multiple behaviours. This adds an extra dimension to the haptic cues that can be transferred to the user.

On the other hand, each HaptiQ can keep track of pressure events, without the intervention of the HaptiQsManager. Pressure data events are automatically fired to registered HapticObjects. Each HapticObject reacts differently to pressure input.

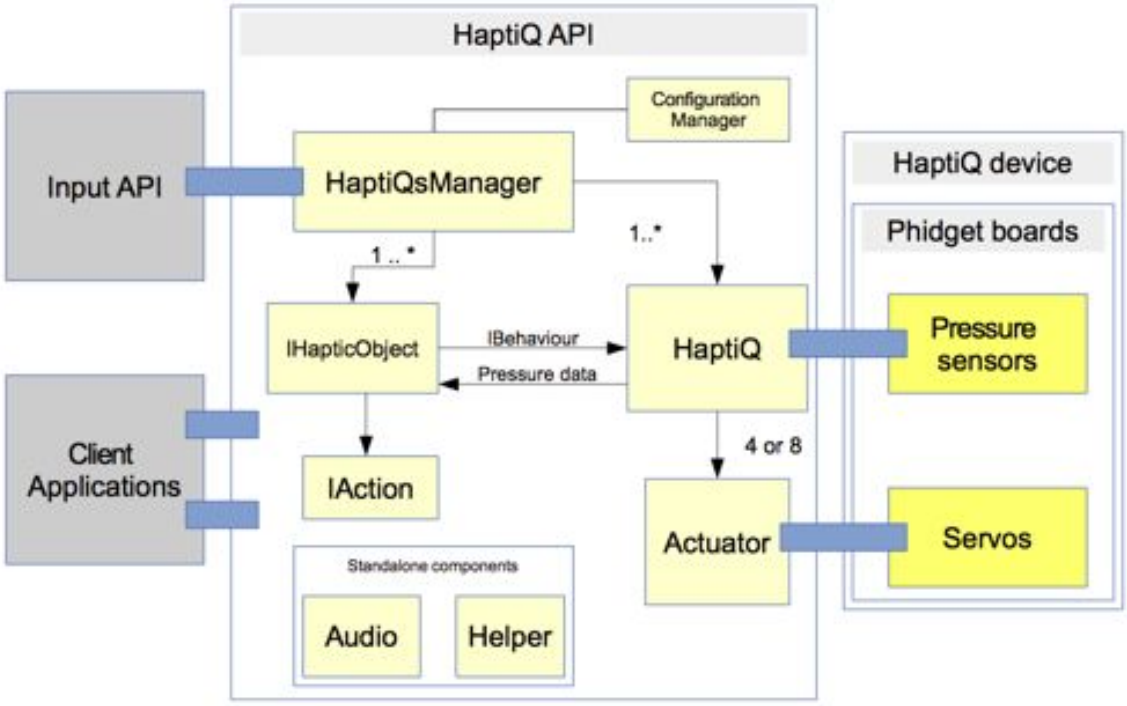


Figure 6.10: HaptiQ API

Haptic Objects

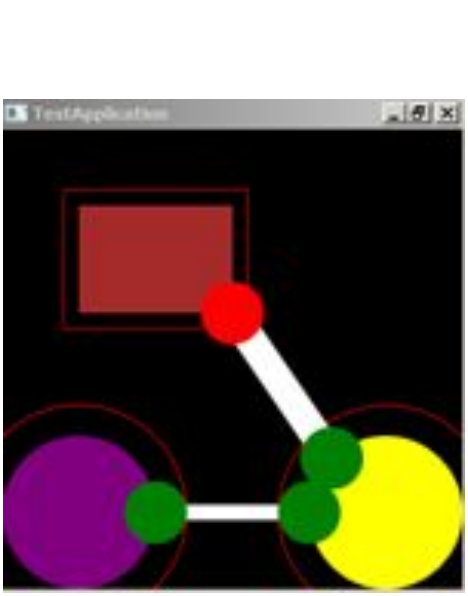
Basic haptic objects are provided by the API and are designed to be used by WPF (Windows Presentation Foundation) applications without having to concern about behaviours and additional low level mechanisms. All haptic objects can carry textual information. In fact, applications should prioritise the information one wants to convey, rather than the aesthetics of the objects. Therefore, only simple shapes are provided. Obviously, one can extend the HapticShape abstract class to provide more interesting shapes, but this is outside the scope of this project.

Haptic objects are the main visual components of the client applications. Therefore,

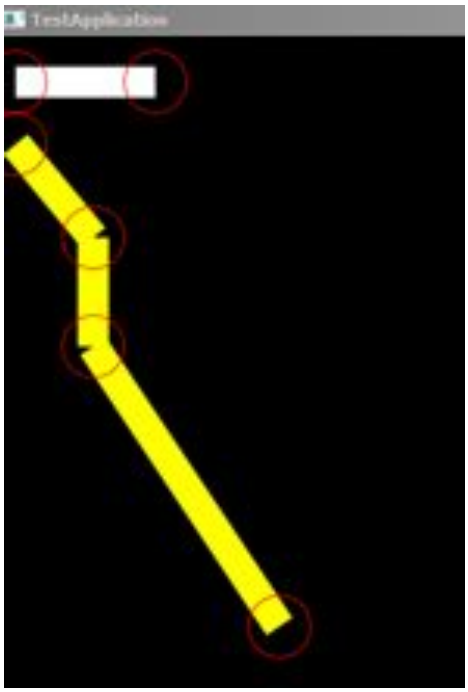
clients can also register custom actions to be run when pressure events are received by a specific object.

Figure 6.11 shows examples of the haptic objects supported by the API. Rectangles and circles can be used as identity objects (e.g. nodes in a graph). Identity objects can also be linked. Links can have directions (from red to green) or not. The API allows also to specify lines and polylines. One of the example applications provided with the API, for example, uses polylines to display mathematical functions.

The used Fiducial Markers do not provide very precise input location of the device. In order to compensate, the HapticObjects provided are extended (red borders in Figure 6.11). This makes it harder to the user to leave the object, compensating for the inaccuracy of detecting the device location.



(a) Identity objects and links



(b) Line objects

Figure 6.11: Haptic Objects

Audio Feedback

The audio component is a standalone one, allowing any part of the API, as well as client applications, to output beeps or textual information as audio.

While this is a small component, it still plays an important role in the API. Ungar and Blades [20] discuss whether the conjoint retention hypothesis, combining spatial and linguistic information, can provide richer haptic cueing. Their study did not show any significant improvement when adding audio feedback. However, they results were not definitive. Therefore, I decided to follow the trend of many commercial applications, like VoiceOver, which use audio to provide textual and position feedback too.

In the API provided, the Beep feedback is used only for HapticLines and HapticPolylines. The reason is that following lines is harder than identifying the existence of shapes. This approach is similar to the one taken in audio feedback applications, such as VoiceOver.

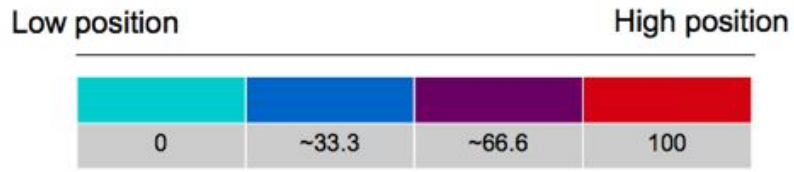
The speech synthesiser, instead, is used to read out textual information stored in the HapticObjects.

6.3 Tactons

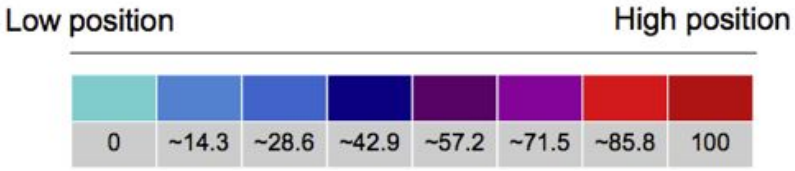
Pietrzak et al. [16] propose a set of tactons that can be used on pin array devices and that could be used to indicate edges and directions. These tactons are designed for a 4×4 pins grid. The HaptiQ provides a maximum of eight actuators, uses lines rather than points to convey haptic cues and can provide height information as well (pin arrays typically provide only 0/1 information). Here, I present a new set of tactons. From this section on, the terms tacton and behaviour will be used interchangeably.

A tacton is used by an HapticObject to notify a device how to react when this is located on top of it. In the provided API there is a strong relationship between each HapticObject and a behaviour. Nonetheless, behaviours can be used by client defined HapticObjects or directly from a client application.

In the following sections, the scales in Figure 6.12 will be used to represent the heights of the actuators.



(a) 4-HaptiQ



(b) 8-HaptiQ

Figure 6.12: Blue-Red scale used to indicate the height of the actuators in tactons diagrams. Numerical values represent the positions of the actuators in percentage

6.3.1 Flat and Max

Flat and Max are the simplest tactons of the set (see Figure 6.13). The Flat tacton is typically used to indicate that underneath the device there are no objects or information. The Max behaviour, instead, can be used to represent the existence of an object. For instance, when the device is inside an HapticRectangle the Max behaviour is played.

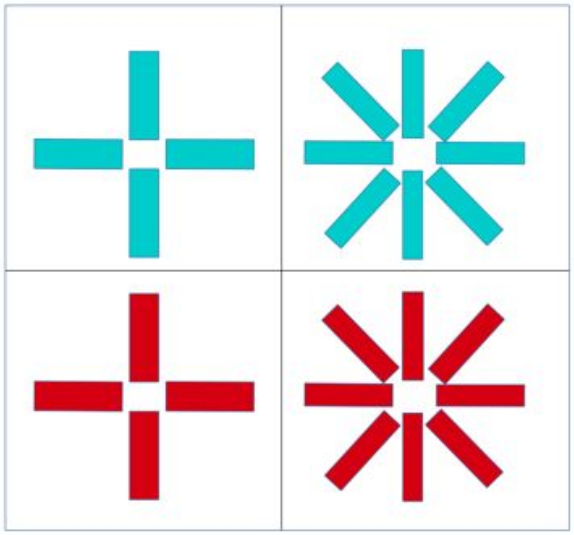
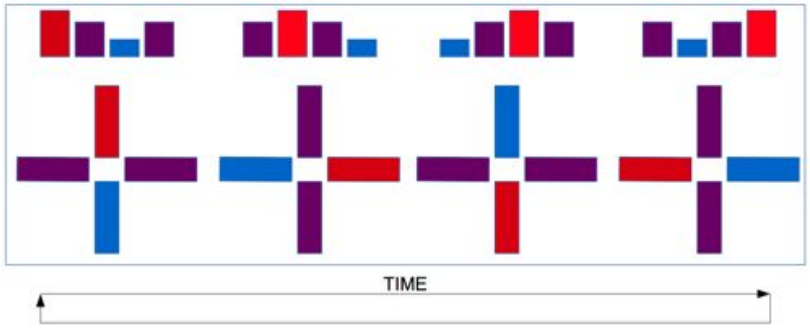


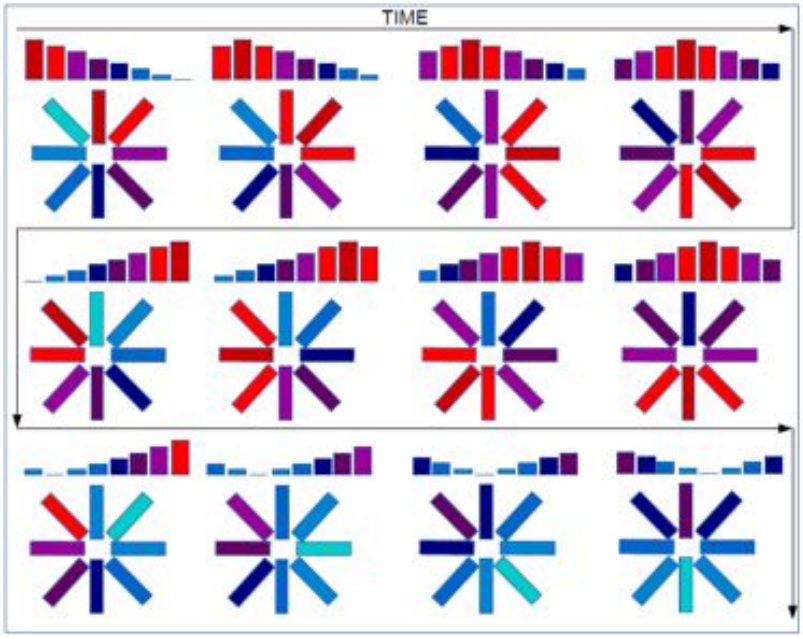
Figure 6.13: Basic tactons: Flat and Max

6.3.2 Notification

The Notification tacton, similarly to the Max tacton, can also be used to notify an object. In the case being, this tacton is used to notify an HapticCircle. This tacton consists into cyclically changing the positions of the actuators (see Figure 6.14). Each rod always increases or decreases its height position. But in the case the maximum or minimum height threshold is reached, the actuator direction (increase or decrease in height) is inverted.



(a) 4-HaptiQ



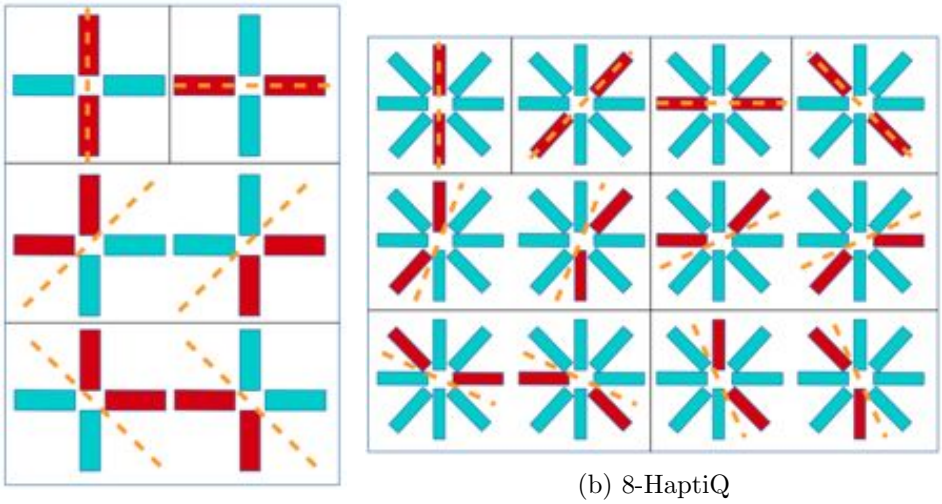
(b) First 12 cases for 8-HaptiQ

Figure 6.14: Notification tacton

6.3.3 Edges and Corners

The Edge-Corner tacton is probably the most significant one of all the tactons by allowing users to perceive edges and corners.

Figure 6.15 show how the combinations of actuators, in both the prototypes, are used to represent edges. The resolution of the device is increased by alternatively actuating adjacent actuators. It is possible to achieve a precision of 45° and 22.5° for the 4-HaptiQ and the 8-HaptiQ respectively.

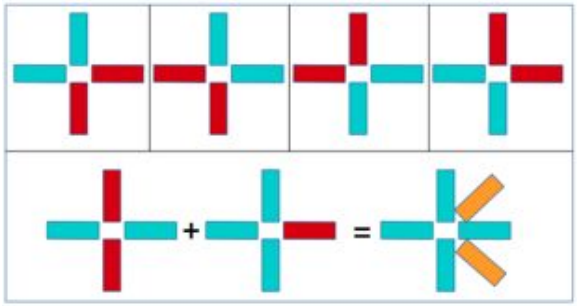


(a) 4-HaptiQ

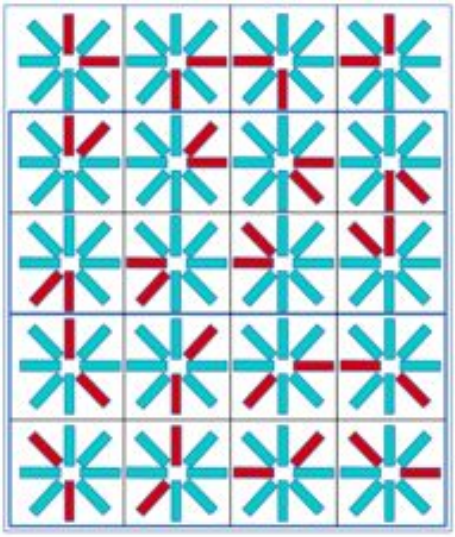
(b) 8-HaptiQ

Figure 6.15: Edge tacton patterns

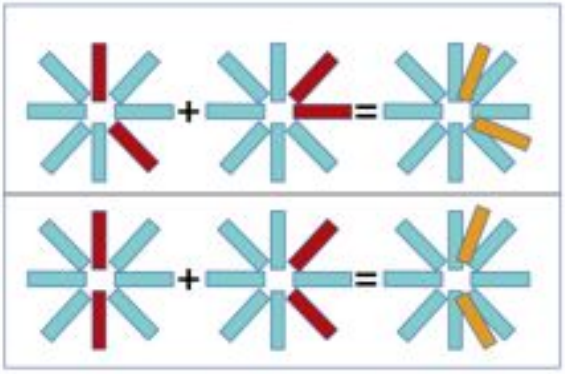
It is also possible to represent corners, which are merely the combinations of two edges. The 4-HaptiQ can represent corners of 90° with a precision of 45° (see Figure 6.16a). Indeed, the 8-HaptiQ can represent corners of 45°, 90° and 135°, with a precision of 22.5° (see Figures 6.16b, 6.16c).



(a) 4-HaptiQ



(b) 8-HaptiQ



(c) 8-HaptiQ (combinations)

Figure 6.16: Corner tacton patterns

The Edge-Corner tacton is used by the HapticRectangle, to represent its borders, and the HapticLine and HapticPolyline objects.

6.3.4 Pulsation and Linear

The Pulsation and Linear tactons are used with the HapticLink to represent the sign of the link between two Haptic objects. Figure 6.17 shows two cases of the pulsation tacton for the 4-HaptiQ. By symmetry it is possible to represent 8 (or 16 for the 8-HaptiQ) directions. This tacton also allows frequency to be specified, so that pulsation increases as one gets closer to the target. The static actuators in this tacton are also fundamental to identify orientation of the direction.

The Linear tacton, on the other hand, is the union of the pulsation tactons, but without

any pulsation behaviour. Thus, the height of the actuators is used rather than the pulsation frequency in order to convey different information about a direction. For example, the tacton can be encoded so that the height of the actuators increases linearly as the distance between the device and the target approaches zero.

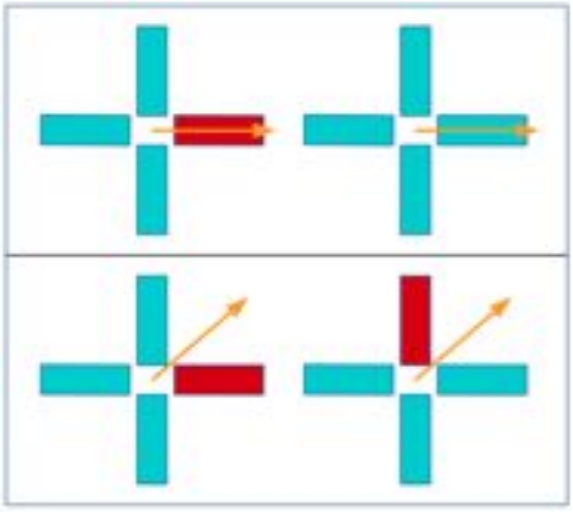


Figure 6.17: Pulsation tacton patterns

6.4 Applications

The HaptiQ can provide a new set of haptic cues which allow the visually impaired to better perceive lines, edges and directions. The two example applications shown below illustrate the types of applications that can be created using the HaptiQ API. Note that these applications have been designed as proof-of-concepts and not as applications ready to be deployed to the end users.

6.4.1 Graph Visualiser

The first application is a basic graph visualiser. The application is designed to be used by people with visual disabilities, with the assistance of a person with normal or corrected vision. The assistant can create graphs using the provided graphical user interface (see Figure 6.18). Graphs contain identity objects and links as well as lines and polylines.
In order to create new object, the assistant has to select one of the options on the left hand

side. A new window will be prompted asking for the properties of the haptic object. The procedure is the same for all the objects, except for links where the user has to select which objects to link by physically touching the objects with a finger or a bytetag.

The design is not complete, since it does not allow objects to be deleted and graphs to be saved for later use. However, this is a good example of a possible application for the HaptiQ. In fact, the majority of blind people have difficulties in visualising graphs such as finite automata, trees, or network topologies.

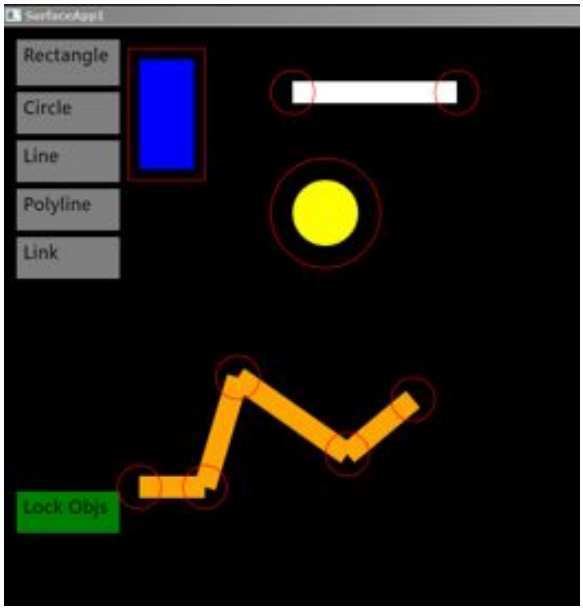
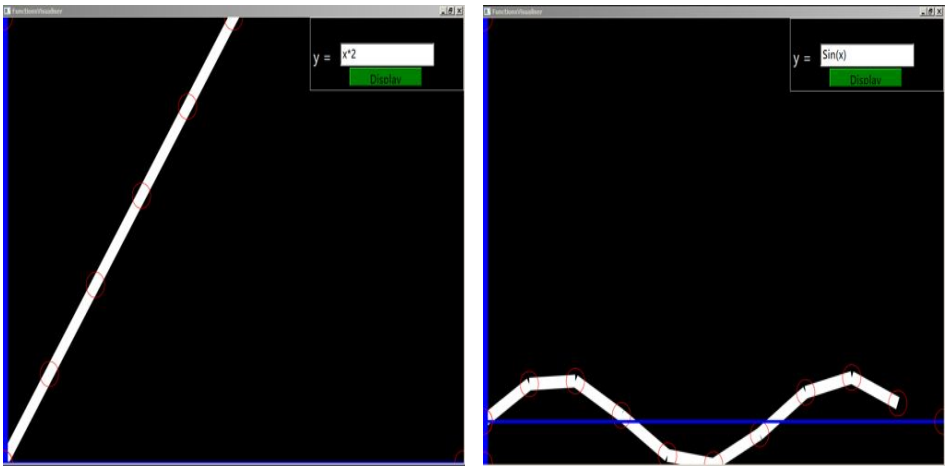


Figure 6.18: Graph Visualiser

6.4.2 Function Visualiser

The second application that this project uses to prove the flexibility and usability of the HaptiQ and its API is a function visualiser. One of the most challenging tasks for the visually impaired is to be able to *visualise* mathematical functions. The design of the function visualiser is also very simple. The user inputs a function, as text, and the application then evaluates and displays it (see Figure 6.19). The application uses haptic lines and polylines to represent the functions. Cartesian axes are also added for completeness.



(a) Linear function (b) Sine function

Figure 6.19: Function Visualiser

Implementation

This chapter provides an overview of the important implementation details about the HaptiQ, as described in the Design chapter. Firstly, the hardware implementation is discussed, followed by the API. A dedicated section will discuss the tactons implementation. Finally, implementation details about the two example applications are covered in the last section of this chapter.

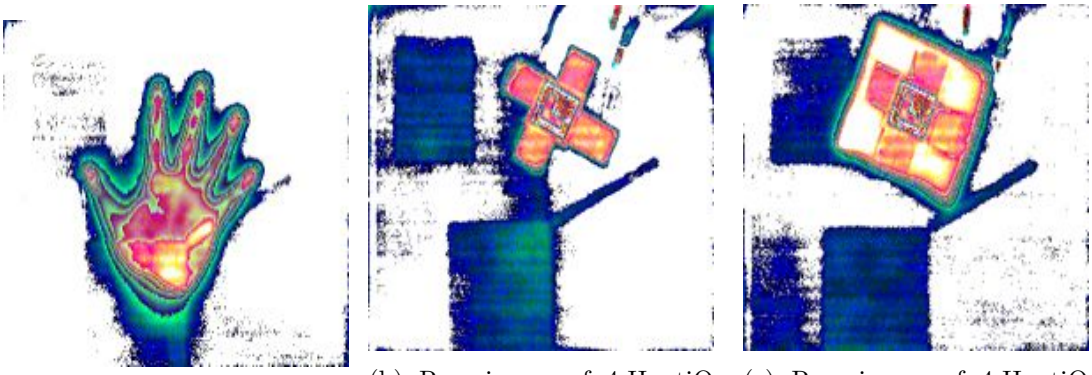
7.1 The Hardware

Building the hardware components of the two HaptiQ devices has been a challenge on its own. Both the 4-HaptiQ and the 8-HaptiQ are 3D-modelled using Google Sketchup [5]. Models are then exported in the STL format, a standard in CAD software, and commissioned to a 3D printer. I used a MakerBot®Replicator^{TM}2x. This is an experimental 3D printer, therefore high precision (<2mm) is hard to achieve and requires many attempts before being able to print the wanted model.

Phidgets boards[6] are used to control the micro servomotors, *Hitech HS-65MG*[7], and get pressure inputs, via force-sensing resistors. The wiring is more of a demanding task, rather than a challenging one, requiring many hours of work.

The different components of the device are assembled using super glue and screws (see the HaptiQ manual for more information on printing and assembling the device).

[5]Google SketchUp. `http://www.sketchup.com` . [Online; last checked: 03/04/2014].
[6]Phidges. `http://www.phidgets.com/` . [Online; last checked: 04/04/2014].
[7]HITECH RCD USA. `http://hitecrcd.com/` . [Online; last checked: 04/04/2014].



(a) Raw image of Hand

(b) Raw image of 4-HaptiQ with Glyph - type 1

(c) Raw image of 4-HaptiQ with Glyph - type 2

Figure 7.1: Raw images

7.1.1 Fiducial Markers

The provided API supports the recognition of two types of fiducial markers: Microsoft Bytetags and GRATF glyphs. The HaptiQs, built for this project, place the fiducial markers on the bottom side. However, in the case of the the GRATF glyphs, the fiducial marker can also be placed on top of the device. This is true if we assume that an appropriate structure around the HaptiQ is built, and a web-cam or similar is used to retrieve top-view images of the device with the glyph.

The Bytetags, according to Microsoft, can be printed on standard office paper [8]. However, during the first iteration phase of this project it was evident that printed Bytetags lead to poor results, with the fiducial markers being tracked at low frequency.

The alternative solution is to use glyphs. The Surface SDK 2.0 allows to retrieve raw image data from the interactive table. Having access to this data, it is possible to apply classical image matching algorithms to identify particular objects. This is achieved using GRATF, a C# library that allows recognition and localization of glyphs in still images.

Figure 7.1 shows three raw images retrieved through the Surface SDK 2.0. The Microsoft PixelSense uses near-infrared sensors. Figure 7.1a, for instance, represents how the hand absorbs infrared light, rather than its heat values as one may think. Glyphs are $n \times n$ grids, with n from 5 to 8, where each cell is either black or white (see Figure 6.7b)

[8] MSDN. Printing Tagged Objects. `http://www.msdn.microsoft.com/en-us/library/ee804862(v=surface.10).aspx/`. [Online; last checked: 04/04/2014].

and minimum side length of 8mm, that can be printed using Glyph Recognition Studio [9]. Additional rules apply, such that that the border cells can only be white and the glyph cannot be symmetrical in any of its axis. The base of the HaptiQ is covered with white paper, so that the black tiles in the glyphs are recognised more easily (see Figure 7.1b). However, this is not sufficient for the glyph to be recognised at all angles. After testing glyphs of different dimensions and on different surfaces, I found out that extending the glyph with low absorption material (e.g. white foam board) improves the recognition of the glyph significantly (see Figures 7.1c and 7.2).

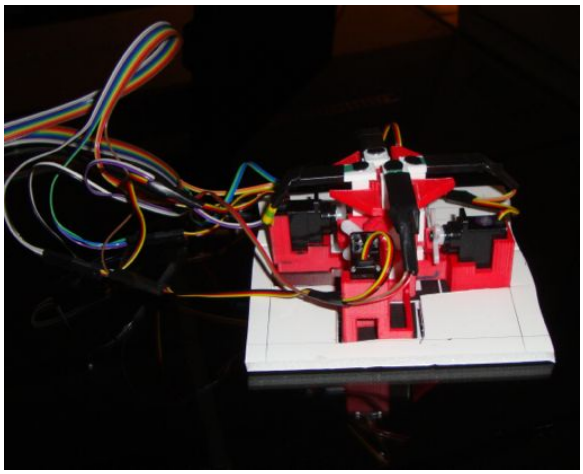


Figure 7.2: HaptiQ with extended white board base

7.2 The API

7.2.1 Input API

The Input API is the component used to retrieve the HaptiQ location on the tabletop. It follows that this is highly hardware dependent on the used hardware, especially on the type of interactive surface. The current targeted hardware comprehends the Microsoft Tabletops using the Surface SDK 2.0. Code to retrieve image data from web-cams is also provided. As explained in the design section, two main classes exist in the Input API: one to recognise Bytetags and one to recognise glyphs.

[9] Glyph Recognition Studio. Available at `http://www.aforgenet.com/projects/gratf/`. [Online; last checked: 04/04/2014].

Recognising the Bytetags is straightforward. This is done through the *TouchTarget* object of the *Microsoft.Surface.Core* library. On the other hand, glyphs are recognised by calling the *FindGlyphs* function, from the GRATF library, on a bitmap image. GRATF does not provide any functionality to estimate the position and the rotation of the glyphs. However, when a glyph is recognised, all its vertices are known. The position is estimated by calculating the crossing point between the two diagonals of the glyph. The rotation, instead, is estimated using the inclination angle of one of the edges of the glyph.

The API raises events of the form:

```
delegate void ChangedEventHandler(object sender, InputIdentifier
    inputIdentifier, Point point, double orientation, EventArgs e);
```

in order to notify the current location of a device, recognisable by a unique identifier (Bytetag or glyph). Then the HaptiQsManager, from within the HaptiQ API, handles such events.

It is important to note that Bytetags are supported for debugging purposes only. Working with the provided tabletop is time consuming. Allowing Bytetags to be used, it is possible to run, debug and test the system using the Input Simulator provided with the Surface SDK 2.0 using a normal machine.

7.2.2 HaptiQ API

The HaptiQ API, as previously stated, controls the device and coordinates its work with the other two main components: the Input API and the applications. The *HaptiQsManager*, the *HaptiQ*, and the *Actuator* classes play the major role in controlling the devices and ensuring the coordination between input locations, haptic objects and behaviours. These three classes are also dependent on the Phidget library [10].

On the creation of the manager, the configuration process is launched (see section 7.3 for more details about devices configuration) and HaptiQ instances are created appropriately. The manager assigns each device a unique ID, which can be used to find and access a device. Each HaptiQ keeps track of its own actuators, the connected pressure sensors, and the current behaviours to play. After an HaptiQ instance is initialised in the constructor, an

[10] Phidgets. Library available at `http://www.phidgets.com/`. [Online; last checked: 04/04/2014].

independent thread is initiated. This consists of a loop that runs all the current behaviours every 10ms. Reducing the frequency gradually decreases the smoothness of the played dynamic behaviours. On the other hand, increasing the frequency creates an unnecessary overuse of resources.

A behaviour consists of a dictionary mapping an actuator of the device to its wanted position. The position is expressed as a value between 0 and 1, where 0 indicates that the actuator should be moved to its minimum position, and 1 to its maximum position. The manager retrieves behaviours to be played, from each haptic object, and distributes them to the devices that generated them. Nonetheless, an HaptiQ instance allows behaviours to be explicitly added or removed, giving more flexibility to clients. Similarly to the HTP toolkit, behaviours can be combined by averaging the positions for each actuator. Actuators are independent from each other, therefore it is possible to exploit parallelism using a parallel for-loop as shown in Code 7.1.

The Actuator class wraps an *AdvancedServoServo* from the Phidget library, used to control a single servo motor, and keeps track of the current pressure applied by the user.

Code 7.1: Parallelising Actuators

```
Parallel.ForEach(actuators, entry =>
                    {
                        this.setActuatorPositionByPercentage(entry.Key, entry.
                            NormalisedPosition);
                    });
```

Pressure Events

The HaptiQ class defines two types of pressure events: pressure changes and pressure gestures. The first type of event is raised whenever the pressure registered by a sensor changes. The second type instead is raised on pressure gestures being recognised. The current API supports only one type of gesture: a *press*. A press is recognised whenever the pressure registered by a particular sensor increases over a certain threshold and then rapidly decreases within a certain time frame (see Figure 7.3).

Time-series analysis could allow the recognition of other gestures, but this is outside the scope of the project.

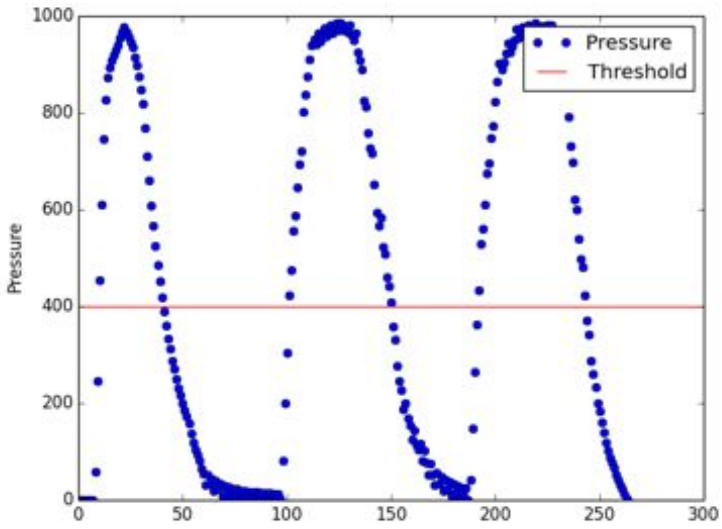


Figure 7.3: Three consecutive press gestures. Note that x-axis does not represent time.

Haptic Objects

Haptic objects are visual elements that client applications can use to represent information to the visually impaired. An Haptic object must implement the IHapticObject interface. This interface, as shown in Code 7.2, consists of three main methods:

- **handleInput**: called whenever a new input location is received from the Input API. This method returns a Tuple of three elements: a rule to apply for the returned values (*ADD*, *REMOVE*, *SUBSTITUTE*, *NONE*) and two behaviours. For instance, the method may add a new behaviour to the HaptiQ, or substitute the current behaviour with a new one. Since HaptiQs can combine behaviours, each HaptiQ object needs to keep track of what current behaviour is assigned to each HaptiQ.

- **handlePress**: called when a pressure gesture event is raised

- **registerAction**: allows clients to register custom actions to be executed on *handlePress*. For example, a *BasicAction* allows the client to specify text to be read out through the speakers when a pressure gesture is fired.

Code 7.2: IHapticObject

```
public interface IHapticObject
{
  Tuple<BEHAVIOUR_RULES, IBehaviour, IBehaviour> handleInput(HaptiQ haptiQ);

  void handlePress(HaptiQ haptiQ, PRESSURE_GESTURE_TYPE gestureType);

  void registerAction(IAction action);
}
```

The API provides the *HapticShape* abstract class, which implements *IHapticObject* and extends *Shape*, from the WPF Windows Shapes. Clients, therefore, have access to haptic objects usable on WPF applications, but can also create their own custom shapes, by implementing *IHapticObject*, for XNA applications.

The *HapticShape* class provides the implementation of *handleInput* and provides other useful functions, such as the possibility to add textual information to a shape. The implemented *handleInput* method ensures that states (*UP* or *DOWN*) and behaviours of a current HaptiQ, for a specific shape, are updated correctly. Classes extending *HapticShape*, such as *HapticRectangle* or *HapticCircle*, need to implement the method

```
IBehaviour chooseBehaviour(HaptiQ haptiQ);
```

which returns the behaviour that should be played by the given device, at a particular state (see Section 6.3 for the tactons used by the provided shapes).

Finally, objects extending *HapticShape* are renderable on WPF applications. These are rendered only when the object is added to a canvas or grid within the client application.

7.3 Configuration

The two prototypes created in this project have different characteristics. For instance, the second version has eight actuators, rather than four, and its servos have a certain action range that the first version does not have. The API allows devices with different characteristics to be used by serializing their configurations as XML files. On creation, the manager inspects the current directory for XML configuration files[11] and attempts to create HaptiQs using the deserialized configurations and the attached Phidget boards.

[11] Pattern used to name configuration files is: 'CONFIG_*.xml'

But even so, it could be the case that no configuration file exists for an attached HaptiQ. This is especially true the first time a device is used. In this eventuality a configuration form is prompted to the user (see Figure 7.4). The form has been created using *WinForms*, so that no dependency on WPF or similar library is added. The panel allows users to specify which actuators to use, their range of action, whether to use pressure input or not, and the fiducial marker to use.

Users are prompted with a configuration form until all attached Phidget boards are configured.

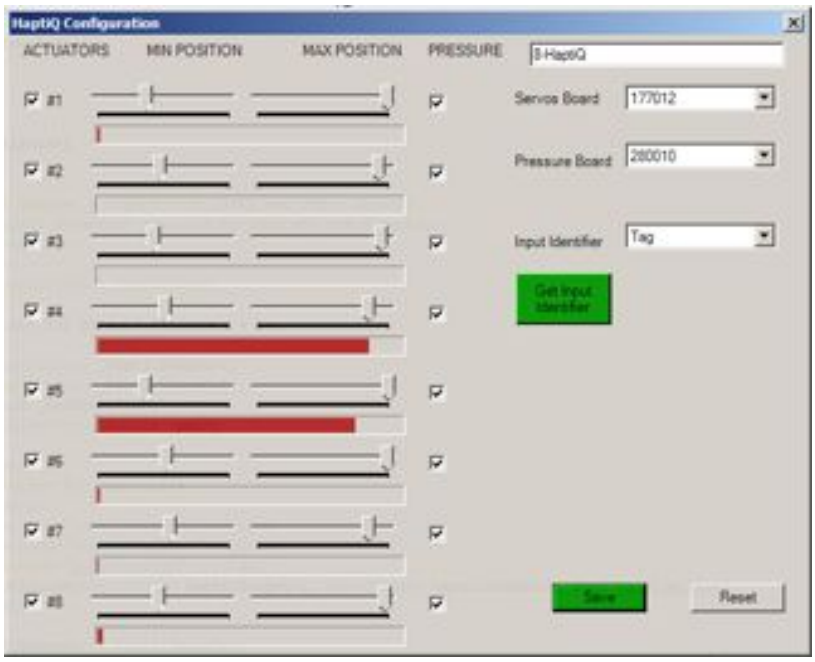


Figure 7.4: Configuration Form

7.4 Tactons

Tactons, or behaviours, represent the way the actuators of the HaptiQ move to represent shapes, edges, and directions. This section gives an overview of the behaviours implementation.

Each behaviour must implement the play method:

```
abstract Dictionary<int, double> play();
```

which returns a dictionary mapping actuators to their desired positions. This is the only requirements that has to be satisfied whenever a new behaviour is created. In addition, tactons can be dynamic. Therefore, the *Behaviour* abstract class provides an internal timer that behaviours should update at each call to *play*.

Being able to map an HaptiQ to basic behaviours, such as *Flat* and *Max*, is fairly simple because there is no dependence on previous states of the behaviour or the position and orientation of the device relative to a segment (e.g. Pulsation behaviour). The initial approach consisted in hard-coding all possible cases as shown in Code 7.3. This example is valid only for the 4-HaptiQ. It becomes clear that as the number of actuators increases, the code will also increases in size and eventually becomes unmaintainable.

Code 7.3: Tactons - Initial approach

```
private int[][][] directionMatrix = new int[][][]
                    {  new int[][]{                    // vertical
                        new int[]{0, 1, 0, 1, 0},
                        new int[]{1, 1, 1, 0, 0},
                        new int[]{0, 0, 1, 0, 1},
                        new int[]{1, 1, 0, 0, 1},
                        new int[]{0, 1, 0, 1, 0},
                        new int[]{1, 1, 1, 0, 0},
                        new int[]{0, 0, 1, 0, 1},
                        new int[]{1, 1, 0, 0, 1}},
                    new int[][]{                       // horizontal
                        new int[]{0, 0, 1, 0, 1},
                        new int[]{1, 1, 0, 0, 1},
                        new int[]{0, 1, 0, 1, 0},
                        new int[]{1, 1, 1, 0, 0},
                        new int[]{0, 0, 1, 0, 1},
                        new int[]{1, 1, 0, 0, 1},
                        new int[]{0, 1, 0, 1, 0},
                        new int[]{1, 1, 1, 0, 0}},
                    ...
                    };
```

The second approach generalises the initial approach, by representing actuators as bits, rather than integers within arrays, and applying bitwise operations (see Figure 7.5).

For example, to represent a vertical line in the 4-HaptiQ, we can just use the number five:

0101 (5) = 0001 (1) & 0100 (4)

Shifting numbers to right (or left) with carry allows behaviours to choose what actuators to use. The shifting offset is based on the orientation of the device compared to the segment it

has to represent. Finding this value consists into subdividing the HaptiQ in logical sectors. For instance, the 4-HaptiQ is divided in eight sectors (see Figure 7.6). Even sectors, in red, represent states that the device needs to represent by alternating actuators over time.

Say that 0101 is valid for the first sector, then when the device is in the third sector a shift by one only is needed: $0101 \gg 1 = 1010$ (with carry). For the case of even sectors, bits are not ended, but applied conditionally on the internal timer value:

0001 (timer mod 2 == 0)

0100 (timer mod 2 != 0)

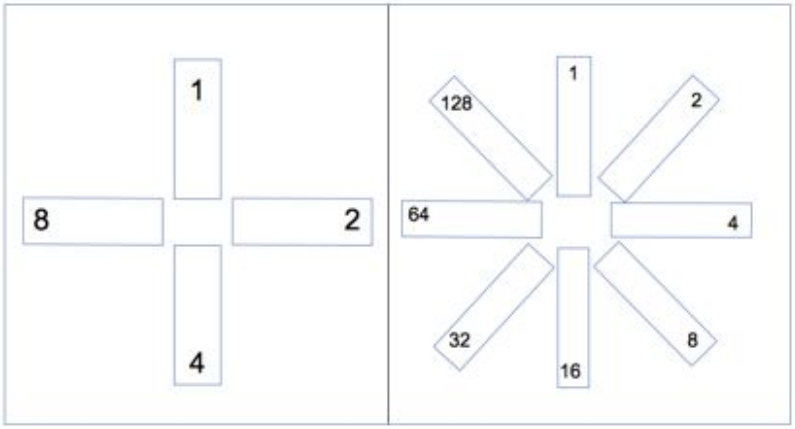


Figure 7.5: Actuators bits representation

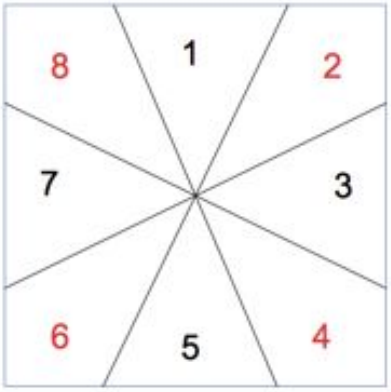


Figure 7.6: 4-HaptiQ sectors

Similar bitwise operations, as described above, are applied for the direction, pulsation,

and linear tactons. The notification tacton, however, takes a different approach since each actuator can be in different states (see Section 6.3.2). The solution used is to use a 2-D array with values $[position, direction]$ for each actuator, where the direction indicates whether the position should increase or decrease in the current iteration. The direction is inverted when the minimum or maximum allowed position is reached.

7.5 A basic example

In this section, I will show how to use the API using a simple example (see Code 7.4). This section is necessary to properly understand section 7.6, which discusses the provided example applications of this project.

The first step consists in creating a WPF application and adding a reference to the HaptiQ_API. In the design view, add a Grid named *container* to the main window. Import the HaptiQ_API in the code view of the application.

The call *HaptiQsManager.Create(windowTitle, <Input Class>)* initialises the HaptiQ device and the Input API, using the specified Input class (e.g. 'SurfaceInput' for Bytetags). Objects are automatically added to the system when they are created. These are rendered only when added to a WPF grid or canvas.

In the *OnClosed* method of the WPF application, it is important to call *HaptiQsManager.Instance.delete()* to ensure that the HaptiQs are detached correctly and all resources are disposed correctly.

Code 7.4: Basic API usage

```
1  using HaptiQ_API;
2
3  public partial class Application : SurfaceWindow
4  {
5    public Application()
6    {
7        HaptiQsManager.Create(windowTitle, ''SurfaceInput'');
8
9        HapticShape rect0 = new HapticRectangle(50, 50, 150, 200);
10       rect0.color(Brushes.Salmon);
11       this.container.Children.Add(rect0);
12
13       HapticShape rect1 = new HapticRectangle(150, 350, 200, 200);
14       rect1.color(Brushes.Orange);
15       this.container.Children.Add(rect1);
```

```
16
17          HapticShape rect2 = new HapticRectangle(550, 150, 100, 100);
18          rect2.color(Brushes.Green);
19          this.container.Children.Add(rect2);
20
21          HapticShape link = new HapticLink(rect2, rect1, true);
22          link.color(Brushes.White);
23          this.container.Children.Add(link);
24      }
25
26      protected override void OnClosed(EventArgs e)
27      {
28          base.OnClosed(e);
29          HaptiQsManager.Instance.delete();
30          Application.Current.Shutdown();
31      }
32  }
```

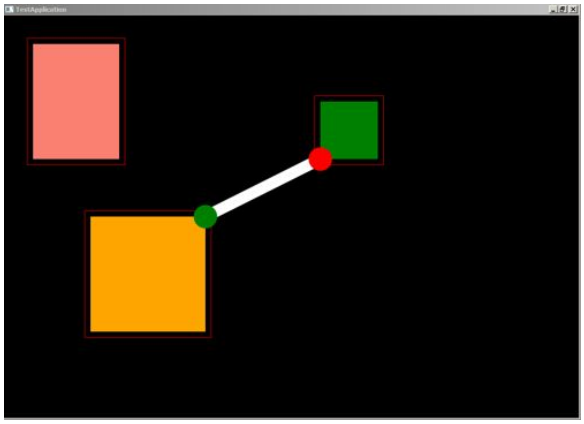


Figure 7.7: Basic API usage

Please refer to the HaptiQ manual for further details and additional examples on how to use the API.

7.6 Applications

This section will briefly explain some of the implementation details about the two example applications provided with this project.

7.6.1 Graph Visualiser

The graph visualiser application, as explained in section 6.4.1, provides basic functionality to build and visualise graphs that can be perceived using the HaptiQ. The main

user interface is created using Visual Studio to organize the WPF standard controls (i.e. buttons). A click handler is attached to each button. Two types of buttons exist in the application. The first type are used to create new haptic objects, while the second one to make the objects selectable or not.
When the first type of buttons is used, a new window is shown allowing the user to choose the properties of the haptic object. A create button in the new window adds the haptic object to the main window, as shown in Code 7.4. The second type, instead, iterates over all the haptic objects through the *HaptiQsManager* and make them selectable or not. This functionality is useful because we want objects to be selectable only when creating the graph and not when the user is sensing it.

7.6.2 Function Visualiser

The function visualiser application is self-contained in one simple C# class. A user can input basic functions (e.g. linear, trigonometric) via the interface provided by the application. The interface, similarly to the Graph Visualiser one, is created with the assistance of Visual Studio.
The application accepts only functions of the form: $y = f(x)$, with only one occurrence of x. Functions are evaluated using the NCalc library [12] at regular intervals over x. The pseudocode 7.5 shows the use of the NCalc library and the creation of the HapticPolyline.

Code 7.5: Function Visualiser pseudocode

```
var regex = new Regex(Regex.Escape("x"));
for(int i = 0; i < RANGE; i++)
{
  var function = regex.Replace(inputTextFunction, i.ToString(), 1);
  NCalc.Expression expr = new NCalc.Expression(function);
  var y = expr.Evaluate();
  points.add(new Point(i, y));
}

HapticShape polyline = new HapticPolyline(points);
polyline.color(Brushes.White);
container.Children.Add(polyline);
```

[12] NCalc. Library Available at `http://www.ncalc.codeplex.com/`. [Online; last checked: 02/04/2014].

Evaluation

The project can be considered to have been successful, with all of the primary and secondary objectives achieved. Additional requirements, not included in the objectives, were also achieved. The project has evolved quickly and future work is already planned as discussed in Chapter 9.

8.1 The Hardware

Designing and implementing the two HaptiQs has been a very challenging task. While the project is heavily inspired by the HTP [14], the device created in this project has been completely reinvented. The process of imagining 3D objects has been stimulating, but the low precision of the 3D printer transformed the entire process into a demanding one. This was particularly true for the first device, the 4-HaptiQ, due to the lack of experience with 3D modelling and printing.

The miniaturization process of the 8-HaptiQ, added additional challenges. Assembling the second device, for instance, requires more time and precision. Some pieces, therefore, are not as easy to assemble as wanted. Nonetheless, being the HaptiQ only a prototype the focus has been more on creating a proof of concept, rather than a final product that can be produced in a short amount of time. One of the issues to solve, for example, is the position of the pressure sensors in the 8-HaptiQ. Currently, the sensors are placed on the outside of the actuators making the use of the device uncomfortable.

No user study was conducted during this project. However, informal feedback on the first version of the device was provided by the supervisor of the project and the collaborators. A not fully working version of the 4-HaptiQ, with no pressure sensors, a Bytetag to locate

the position of the device on the interactive table, and a point-reference actuator was tested. The first note made after the test was about the point-reference actuator, which was disrupting the usability of the device. I also observed that even if the device did not work properly, it was still possible to follow object edges with acceptable accuracy. The fact that tactons can be learned even with little or no practice is a positive feedback.

I was unable to organize any other test sessions to get feedback about the 8-HaptiQ. However, I plan to pursue additional testing sometime in the near future to start working on the next iteration phase (see Chapter 9).

The HaptiQ, similarly to the HTP, is intended to be inexpensive. The two prototypes created in this project have an estimated cost of circa £300.00. Using cheaper printed circuit boards, however, can lower the price to less than £100.00. This is a significant improvement over the current commercial technology (prices are usually over £1000.00).

8.2 The API

The structure of the API has changed few times, based on how the hardware evolved, the challenged faced and the new requirements that were introduced during the project. Initially, the API was designed to support input location via Bytetags only. However, glyphs had to be introduced to increase reliability and accuracy. In the month of February I was also told that Régis (he is one of the two collaborators, see Section 5.1) would join the project in March and that he will then use an interactive table not supporting the Microsoft Surface SDK 2.0. Therefore, I removed any dependency from the Surface SDK from the HaptiQ API and created a separate API to retrieve the input locations of the devices.

The API has been designed and implemented to allow clients to easily use and possibly extend it. Régis has been using the API since the 18th March 2014, without any major difficulties. He also played the role of the client in the project, by submitting issues about the project through GitHub. Some issues are still not solved, because not relevant to the objectives of this project. Other issues are simply not solvable due to technology limitations. For instance, at the moment it is not possible to define when a device is currently on the table or not. This is due to some inaccuracy issues of the GRATF library in recognising the glyphs and/or on the table providing raw images of low quality. A possible solution could

be to classify a device as not being on the table whenever a glyph is not being recognised for more than a certain time threshold.

The API supports basic pressure gestures recognition. The only gesture currently recognised is a press: the user applies high pressure on one of the actuators and then releases the finger or the palm. A drawback is that gestures can be recognised for single actuators only. It is also possible to detect more complex and interesting patterns by applying time-series analysis or using an hidden Markov model, but this is was outside the scope of this project. Figure 8.1 is an example of a more complex gesture, where checking values over the threshold might be misleading.

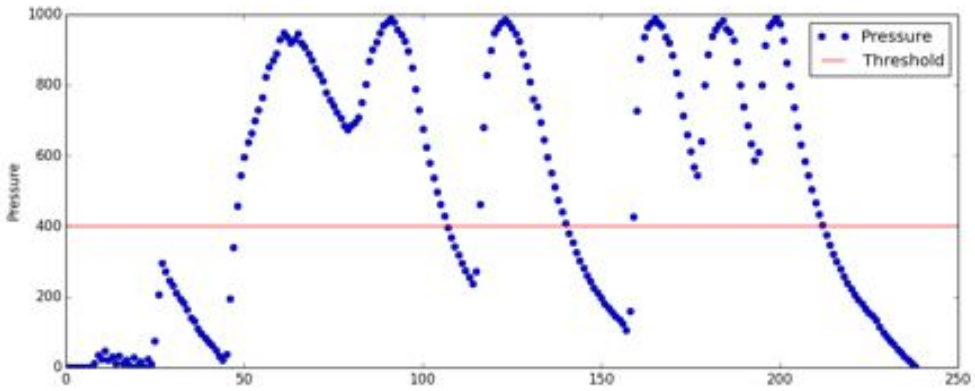


Figure 8.1: Complex pressure gesture

8.3 Tactons

To the best of my knowledge, this work is the first to explore dynamic vector-based tactons to exploit haptic cues of edges, corners and directions. The tactons proposed by Pietrzak et al. [16] are based on pin arrays. In his paper, it is interesting to note how the designed tactons evolved as user experiments were conducted. The tactons used for the HaptiQ, instead, are merely based on common sense rather than experimentation. Evaluating the current set of tactons and eventually producing a new improved set is part of the future work discussed in the next chapter.

During one of the meetings with the collaborators, we discussed a subset of the tactons presented in this report. The feedback was positive. The meeting after, during the test

phase of the HaptiQ, the behaviours were encoded efficiently. This cannot be considered a study of any kind, but the feedback was very valuable to define how the HaptiQ has evolved so far.

8.4 Applications

The applications presented in this project function as proof of concepts only. Therefore, the applications can be improved largely. The graph visualiser, for example, could support the deletion of haptic objects and the possibility to save and load graphs for later uses. The functions visualiser, instead, can be of great help to students taking courses in mathematics both at school and university level. The application performs very well already, but there are still some issues with scaling and zooming to be solved.

8.5 Summary

This project was very open-ended and consisted of many different components. Initially, I planned to conduct a user study to evaluate the HaptiQ. However, due to major ethical issues, I decided, with my supervisor, to focus more on the engineering aspects of the HaptiQ and its API. Nonetheless, an iterative process was used rather than a well structured one with well defined requirements. This allowed me to created two devices, with the second one completely re-engineered, and to evolve the API as needed.

As previously stated, no user study has been conducted to formally evaluate the device. The project, however, has been assisted and supervised throughout the whole design and implementation phases by Dr Miguel A. Nacenta, who is an expert in Human-Computer Interaction and haptic interfaces. The contribution of Dr Nacenta has been particularly fundamental to validate the design process of the hardware and the tactons.

Conclusions can also be drawn by comparing the device with the state of the art in haptic technology.

The HaptiQ, unlike MIMs [6] and inFORM [10], can be used on any interactive surface as long as it is possible to track the device. The HaptiQ, for example, can be used on a normal paper map assuming a top-down tracking mechanism is provided. This makes the HaptiQ

more flexible and less expensive, since it can be reused in different scenarios.
TeslaTouch [5] and REVEL [4] allow users to perceive tactile cues extrinsically or intrinsically. Both of these systems allow users to feel texture and to follow lines. The HaptiQ uses a variety of tactons to indicate not only lines, but directions as well. In addition, electrovibration or vibrational mechanical motors can be integrated with the HaptiQ to allow textures to be sensed.
UltraHaptics [8] and AIREAL [19] provide haptic feedback in a 3D environment. Accuracy, in these systems, is not constant in space. These solutions, in fact, tend to use tactile feedback for media applications to be used by the visually impaired, rather than by blind people.

Finally, a brief discussion about the objectives that were not achieved is needed.
Initially, one of the secondary objectives was to allow dynamic pointing on the interactive surface. But people with visual disabilities may encounter difficulties on keeping track of the relative position of a dynamic pointer, since no visual feedback can be perceived. Therefore, this objective was discarded.
The tertiary objectives included: enabling the haptic device to be used collaboratively and allowing the device to sense textures. With the focus of the project being on the tactile cues that the HaptiQ could enhance, I decided to give lower priority to any possible feature that could allow communication between two devices. However, the API can support multiple devices. Thus, clients can implement collaborative features on top of the API. On the other hand, the objective of adding the possibility to sense texture was not attempted because the frequency at which the servos, used for this project, work is too low [7]. A possible alternative could be to add a central static actuator to the devices, similar to the point-reference actuator, which could convey texture information to users via electrovibration [4, 5] or mechanical vibration [13].

[13]Example of vibration motor for haptic feedback available at `http://www.precisionmicrodrives.com/vibrating-vibrator-vibration-motors`. [Online; last checked: 04/04/2014].

Future Work

The HaptiQ aims to take the Haptic Tabletop Puck to the next level, by allowing the visually impaired to sense edges, corners and directions, and still keeping the hardware inexpensive. In the future, I plan to pursue additional work in collaboration with Régis Ongaro-Carcy, Saad Attieh, and Dr Miguel A. Nacenta. This chapter intends to highlight the potential work that could be planned for the near future, as well as the work currently pursued by Régis Ongaro-Carcy and Dr Miguel A. Nacenta.

One of the aims of this project is to distribute this work under the GNU General Public License. We believe that the HaptiQ can facilitate blind people to use interactive surfaces without having to spend a considerable amount of money and still providing a sufficient haptic-audio feedback quality. The next iteration of the HaptiQ will focus on improving the design and mechanics of the device. The current version is easy to print, but requires about 10 hours of assembling. It is important to redesign some of the pieces and part of the structure of the device, making the HaptiQ more modular and easy to access to people with no experience with 3D printing and models construction.

In this work no user study was conducted. While the HaptiQ seems to perform very well, it is still unknown how helpful the provided tactons are. A user study with people with visual impairments could help understand how the design, mechanics, and tactons of the HaptiQ perform. Whether a vector-based haptic feedback display provides more helpful cues to the blind and visually impaired when following a line, or on recognising a corner, is unknown. The current device supports modular actuators, so it is possible to compare the vector-based HaptiQ with a grid-based one. A study analysing the current HaptiQ will help to understand how well the current device performs and what is to be improved, added, or removed in the next iteration phase. For instance, it would be very interesting to compare

technology similar to VoiceOver [3], that uses audio feedback only, with the HaptiQ that uses both haptic and audio feedback (or haptic only). An improved version of the function visualiser application, for instance, would help understanding how well blind people can follow lines using the HaptiQ.

Future work will consist into exploring what new features should be added to the HaptiQ and its API. New types of haptic objects and behaviours could make the API more powerful, allowing client applications to describe more scenarios. The use of electro-vibration and/or mechanical vibrations could allow the HaptiQ to sense textures. This would be a powerful feature, since the aim of HaptiQ is to compensate for the flat and haptic-less interactive surfaces. Adding a break, like the HTP [14] does, would add a new dimension to the HaptiQ: friction. Providing resistance to movements parallel to the used surface can allow applications to represent certain types of textures. There are also other possibilities. For example, different meanings can be given to objects that exert different opposing forces to the movement of the HaptiQ.

Currently, Régis is developing a strategic game, based on geographical information, for visually impaired people. He is working under the supervision of Dr Miguel A. Nacenta and will conduct a user study with visually impaired people at the Université de Toulouse. While he is working on his research, I will provide support with the HaptiQ, being the main developer. I am also planning to actively participate in the design and development of the game, with the permission of Régis and Dr Nacenta.

Conclusions

Technology, in the last decade, has become increasingly ubiquitous. However, little effort has been spent on enabling the digital world to be used by people with disabilities. Blind people, in particular, have to face considerable difficulties when accessing textual and visual information. While screen readers are able to convey textual information with a good degree of success, the same is not true for data with a more graphical nature. Researchers are exploring haptic feedback technologies, allowing the blind to perceive shapes and textures, but low flexibility and high cost are still a main concern.

This project presents the HaptiQ, an inexpensive haptic-audio feedback device that the visually impaired can use on interactive surfaces to explore both textual and graphical information. The main focus of the HaptiQ is to facilitate the blind to sense edges, corners and directions.

All the main objectives of this project have been achieved. Two prototypes have been created, an API provides both low and high level control over the device, and two example applications show some practical uses. This work presents novel approaches in the area of haptic feedback technology. The HaptiQ, to the best of my knowledge, is the first vector-based haptic feedback display for the visually impaired. No user study has been conducted to formally evaluate the performance of the device. However, informal observations have given positive feedback. The device also provides audio feedback, adding an additional dimension that users can perceive.

The HaptiQ uses a new set of dynamic tactons to convey information to the visually impaired. Tactons are "structured, abstract, tactile messages which can be used to communicated information non-visually"[7]. The tactile encoding used by the HaptiQ is strongly driven by the design of the device itself. The main emphasis, on designing both the device

and the tactons as well, was to facilitate the recognition of edges, corners and directions by the visually impaired.

This work has some limitations that will be addressed in the next versions of the HaptiQ. The first issue to address will be to re-engineer the device in order to decrease its size. A smaller device could allow many users to collaborate using the same interactive surface. Friction and texture are two dimensions that have not been explored due to time constraints. Nonetheless, these properties could open new unexplored perspectives.

Moreover, the project opens to new areas of further research. A study targeting the usability and performance of the device itself would help the next development phase to produce a device closer to a final product. In addition, Régis Ongaro-Carcy and Dr Miguel A. Nacenta are currently working on building a strategic game for the visually impaired that makes use of the HaptiQ. A user study will evaluate how well the device performs when used collaboratively in a gaming environment. I have passively assisted the project by adapting both the hardware and the API to their needs. But, I am planning to actively contribute in the design and implementation of the game as well.

Finally, the biggest contribution of this project has been trying to bridge the digital divide gap for the blind. This work will be released under the GNU General Public License allowing the visually impaired to make their own haptic device and applications at home, with a low budget. The HaptiQ is not a final product yet, but I am planning on improving both the hardware and the software components as a personal side project.

Bibliography

[1] Freedom Scientific. `http://www.freedomscientific.com/`. [Online; last checked: 20/03/2014].

[2] HumanWare - Braille Displays. `https://www.humanware.com/en-united_kingdom/products/blindness/braille_displays`. [Online; last checked: 20/03/2014].

[3] VoiceOver. `https://www.apple.com/uk/accessibility/osx/voiceover/`. [Online; last checked: 22/03/2014].

[4] Bau, O., and Poupyrev, I. Revel: tactile feedback technology for augmented reality. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 89.

[5] Bau, O., Poupyrev, I., Israr, A., and Harrison, C. Teslatouch: electrovibration for touch surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology* (2010), ACM, pp. 283–292.

[6] Brock, A., Truillet, P., Oriola, B., and Jouffrais, C. Usage of multimodal maps for blind people: why and how. In *ACM international conference on interactive tabletops and surfaces* (2010), ACM, pp. 247–248.

[7] Brown, L. M., Brewster, S. A., and Purchase, H. C. A first investigation into the effectiveness of tactons. In *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint* (2005), IEEE, pp. 167–176.

[8] Carter, T., Seah, S. A., Long, B., Drinkwater, B., and Subramanian, S. Ultrahaptics: multi-point mid-air haptic feedback for touch surfaces. In *Proceedings of the 26th annual ACM symposium on User interface software and technology* (2013), ACM, pp. 505–514.

[9] D'Andrea, F. M. From carvings to computers:a history of tactile codes for people who are bling. *The Educator 21*, 2 (January 2009).

[10] Follmer, S., Leithinger, D., and Ishii, A. O. A. H. H. inform: dynamic physical affordances and constraints through shape and object actuation. In *Proceedings of the 26th annual ACM symposium on User interface software and technology* (2013), ACM, pp. 417–426.

[11] Harrison, C., and Hudson, S. E. Providing dynamically changeable physical buttons on a visual display. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2009), ACM, pp. 299–308.

[12] Ishii, H., and Ullmer, B. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems* (1997), ACM, pp. 234–241.

[13] Ledo, D., Nacenta, M. A., Marquardt, N., Boring, S., and Greenberg, S. The haptictouch toolkit: enabling exploration of haptic interactions. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (2012), ACM, pp. 115–122.

[14] Marquardt, N., Nacenta, M. A., Young, J. E., Carpendale, S., Greenberg, S., and Sharlin, E. The haptic tabletop puck: tactile feedback for interactive tabletops. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (2009), ACM, pp. 85–92.

[15] Massie, T. H., and Salisbury, J. K. The phantom haptic interface: A device for probing virtual objects. In *Proceedings of the ASME winter annual meeting, symposium on haptic interfaces for virtual environment and teleoperator systems* (1994), vol. 55, Chicago, IL, pp. 295–300.

[16] Pietrzak, T., Crossan, A., Brewster, S. A., Martin, B., and Pecci, I. Creating usable pin array tactons for nonvisual information. *Haptics, IEEE Transactions on 2*, 2 (2009), 61–72.

[17] Roberts, J. C., and Paneels, S. Where are we with haptic visualization? In *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint* (2007), IEEE, pp. 316–323.

[18] Shimada, S., Murase, H., Yamamoto, S., Uchida, Y., Shimojo, M., and Shimizu, Y. Development of directly manipulable tactile graphic system with audio support function. In *Computers Helping People with Special Needs*. Springer, 2010, pp. 451–458.

[19] Sodhi, R., Poupyrev, I., Glisson, M., and Israr, A. Aireal: interactive tactile experiences in free air. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 134.

[20] Ungar, S., Blades, M., and Spencer, C. Can a tactile map facilitate learning of related information by blind and visually impaired people? a test of the conjoint retention hypothesis. *Proceedings of Thinking with Diagrams 98* (2000).

[21] van der Vleuten, R. Blind Maps. `http://www.rubenvandervleuten.com/blindmaps.html`. [Online; last checked: 22/03/2014].

[22] Weiss, M., Wacharamanotham, C., Voelker, S., and Borchers, J. Fingerflux: near-surface haptic feedback on tabletops. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (2011), ACM, pp. 615–620.

[23] Xu, C., Israr, A., Poupyrev, I., Bau, O., and Harrison, C. Tactile display for the visually impaired using teslatouch. In *PART 1————-Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems* (2011), ACM, pp. 317–322.

[24] Yu, W., Ramloll, R., and Brewster, S. Haptic graphs for blind computer users. In *Haptic human-computer interaction*. Springer, 2001, pp. 41–51.