# LUI: A scalable, multimodal gesture- and voice-interface for Large Displays

by

Vikraman Parthiban

M.S. Electrical Engineering, The University of Texas at Austin (2016)
B.S. Computer Engineering, The University of Texas at Austin (2014)
B.A. Liberal Arts, The University of Texas at Austin (2014)

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

~~August 2019~~ [September 2019]



Signature redacted

Author .....
Program in Media Arts and Sciences, School of Architecture and Planning
August 9th 2019

Certified by ..... Signature redacted
V. Michael Bove
Principal Research Scientist
Thesis Supervisor

Signature redacted

Accepted by .....
Tod Machover
Academic Head, Program in Media Arts and Sciences

# LUI: A scalable, multimodal gesture- and voice- interface for Large Displays

by

Vikraman Parthiban

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning
on August 9th 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

## Abstract

With the rise of augmented and virtual reality, new interactive technologies are incorporating immersive user interfaces that leverage gesture and voice recognition in addition to existing controller inputs. However, the state-of-the-art interfaces are quite rudimentary and not widely accessible for the user. Such interfaces require significant amount of sensors, extensive calibration, and/or high latency in the gestural commands. LUI (Large User Interface) is a scalable, multimodal interface that uses a framework of nondiscrete, free-handed gestures and voice to control modular applications with a single stereo-camera and voice assistant. The gestures and voice input are mapped to web UI elements to provide a highly-responsive and accessible user experience. The menu screen consists of an extendable list of applications, currently including photos, YouTube, etc, which are navigated through the input framework. This interface can be deployed on an AR or VR system, heads-up displays for autonomous vehicles, and everyday large displays.

Thesis Supervisor: V. Michael Bove
Title: Principal Research Scientist

This thesis has been submitted and approved by the following committee members:

Thesis Supervisor ..... Signature redacted

V. Michael Bove, PhD
Principal Research Scientist
Program in Media Art and Sciences

Thesis Reader .....

John Underkoffler, PhD
CEO, Oblong Industries

Thesis Reader .....

Zachary Lieberman
Co-Founder, OpenFrameworks
Parsons School of Design

# LUI: A scalable, multimodal gesture- and voice- interface for Large Displays

by

Vikraman Parthiban

The following person served as a reader for this thesis:

Thesis Reader Signature redacted

*John Underkoffler, PhD*
*Oblong Industries*

# LUI: A scalable, multimodal gesture- and voice- interface for Large Displays

by

Vikraman Parthiban

The following person served as a reader for this thesis:

Signature redacted

Thesis Reader____ _____

*Zachary Lieberman*
*Parsons School of Design*

# Acknowledgments

I would first like to thank my advisor V. Michael Bove and Sunny Jolly for giving me a home at the MIT Media Lab. Leaving my career, family, and warm weather in Texas was a difficult moment in my life, but the Media Lab provided a warmer welcome. Michael, you encouraged me to look at challenging projects at the Media Lab where I started my work in the holographic video space. This led to my work on LUI, because I commonly found myself frustrated when working with new poorly designed interfaces for augmented and mixed reality.

I want to thank John Underkoffler, whom I was fortunate to meet during a conference and lunch at the MIT Media Lab and inspired me to start building LUI. John, you taught me the history behind gestural interfaces and encouraged me to not give up. And of course, I would not be here if it were not for our mutual colleague Michael Klug. Michael, your advice while I was a full-time engineer at Magic Leap was instrumental. You told me that I would not regret coming to the Media Lab, and it's definitely true.

I also thank Zach Lieberman. Zach, you constantly offered up time via office hours to help and motivate students in artistic areas of their career. Your work in computation and design encouraged me to instill similar qualities in LUI.

And of course, I would be remiss if I did not thank my wonderful UROP students, Ashley Lee, Claire Tang, Kathryn Jin, and Jeremy Ma who committed their time and energy to mold LUI into what it is today.

To my MIT Hyperloop II team, you all gave me an outlet to work on a project so fun and MIT-worthy. The experience of building an award-winning electric Hyperloop vehicle in 6 months (and later 3 weeks) was incomparable.

I want to thank my family. Mom, Dad, Pradeepan, and Kayal, I am blessed to have

you in my life, and I promise I will call more often. And to my dear Zainab, you gave me a reason stay in Boston. We met at such a unique moment at MIT, and I would not have it any other way. I love you so much :)

Lastly, I thank God for giving me the strength and ability to get through these tough years at MIT. It was probably the most challenging two years of my life, and I'm excited to see where I'm going next.

# Contents

# List of Figures

# Chapter 1

# Introduction

Large 4k and 8k TVs have been entering the market to help visualize immersive datasets and media. However, these displays do not leverage the screen real-estate provided to the users but rather rely on pointers or controllers to manipulate the content. Multimodal interfaces create new ways of interacting and visualizing content on displays which are otherwise static. In this thesis, we specifically look at the combination of free-handed gesture and voice input on a large 2D display. Research has shown that gestures can be used with speech to provide additional information or meaning [8].

Previously working in the augmented reality industry, I was frustrated with how we still interact with content and media. The hand and gesture tracking currently available on AR headsets are limited in functionality. We still use remotes to view content on large TV screens or controllers for augmented and virtual reality environments. Our current methods to interact with such content creates a barrier between information and the user. LUI, or Large User Interface, came out of this frustration and a desire to remove that barrier. I envision a new "visual AI" that would be natural to the user's sensory inputs including gestures and voice. One day I hope to build the interactive, holographic Jarvis from the movie Iron Man.

## 1.1 What is LUI?

LUI is a scalable, multimodal web-interface that uses a custom framework of nondiscrete, free-handed gestures and voice to control modular applications with a single stereo-camera and voice assistant (see Figure 1-1). The gestures and voice input are mapped to ReactJS [13] web elements to provide a highly-responsive and accessible user experience. This interface can be deployed on an AR or VR system, heads-up displays for autonomous vehicles, and everyday large displays.

Integrated applications include media browsing for photos and YouTube videos. Viewing and manipulating 3D models for engineering visualization are also in progress, with more applications to be added by developers in the longer-term. The LUI menu consists of a list of applications which the user can "swipe" and "airtap" to select an option. Each application has its unique set of non-discrete gestures to view and change content. If the user wants to find a specific application, they can also say a voice command to search or directly go to that application.

Developers will be able to easily add more applications because of the modularity and extensibility of this web platform. Most of the gestures are discrete actions followed by a UI response instead of a continuous action that changes UI in real-time. The space of multimodal gesture- and voice- interfaces is more limited, and LUI hopes to create a more seamless experience for the user with technology that is readily available and easy to integrate.

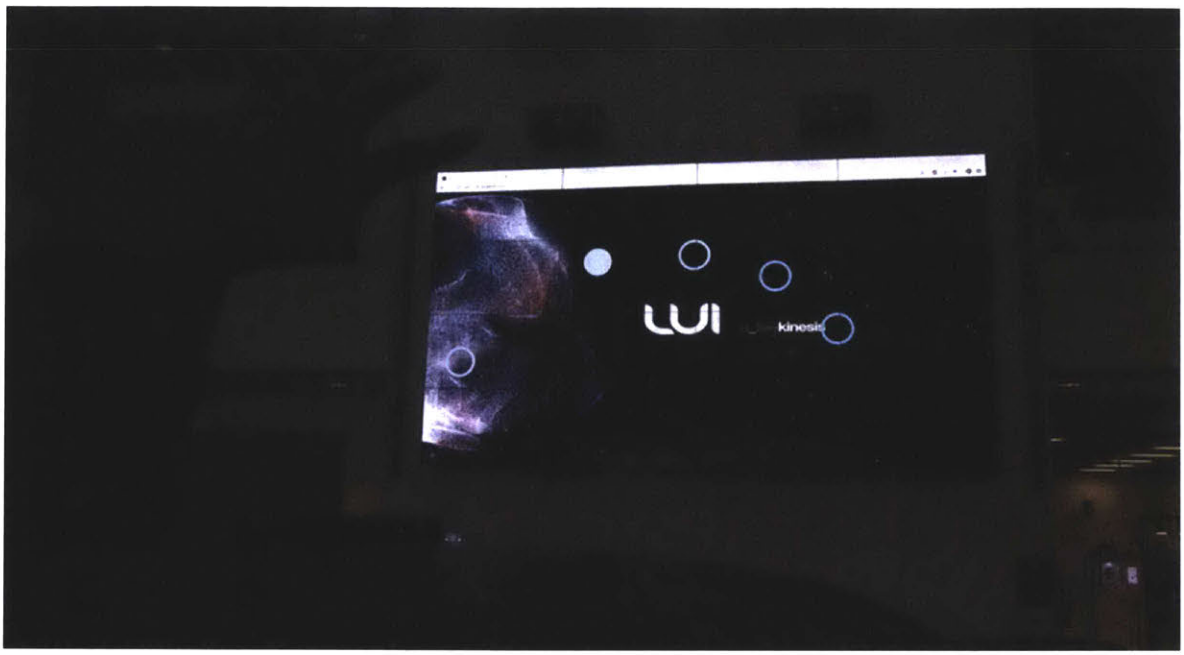



Figure 1-1: LUI demo on 17 ft diagonal display

## 1.2 Motivations

During my research, I had the opportunity to interview one of my readers, Dr. John Underkoffler, about his pioneering work in the space of gestural interfaces. His invention, g-speak [12], was part of his dissertation work at the MIT Media Lab. He mentioned that existing tools didn't have a great way of expressing space and time, so he built a general purpose software stack to represent just that. G-speak's gestural schema relied on "partially quantizing finger flex, where each finger can be in one of many positions, such as closed, curbed, or extended." There was a "time-slider" pose and "camera pic snapshot" pose. The hardware required a pair of gloves that was based on 6DOF pose and identity and required little dots and retroreflective marks on each finger.

G-speak never went through a formal user study, but it did achieve mainstream Hollywood popularity through the movie "Minority Report." Much of the development happened in house and there were often debates about finger markers on finger tips or

further down. G-speak didn't take off as a company or product for several reasons. The technology was expensive (about $150,000 to $200,000) since it required a sequence of cameras also facing the user. The goal was to provide high fidelity at high Field of View (FOV), but hurdles included standardizing vocabulary and the necessity of wearing the gloves at all times. At the same time, it was commonly agreed that the public wasn't ready for such a system yet.

Given this insightful conversation from John, I hoped to make a new gestural interface that would be low-cost, easily accessible, and developer-friendly which is the top of my design guidelines in Chapter 3. I hope that in the next 5 years, we won't have to stick with poor desktop based UI experiences for the future of storytelling, interaction, and holographic displays.

# Chapter 2

# Prior Art

There has been much work done in the space of gestural interfaces, but many require significant amount of sensors, extensive calibration, and/or high latency in the gestural commands. Most of the gestures are discrete actions followed by a UI response instead of a continuous action that changes UI in real-time. Below is a list of related work in the field of multimodal, gesture, and voice interfaces. Some use pen-input or touch-input, but the UI designs contain unique qualities the we hope to extend in LUI.

## Put-That-There (1980)

In 1980, Richard Bolt, Chris Schmandt, et. al. from the Architecture Machine Group at the Massachusetts Institute of Technology revealed the first voice- and gesture- interface called Put-That-There [2]. The system utilized word-by-word speech-recognition and hand-pointing as inputs for controlling a large graphics display (see Figure 2-1). The hand tracking sensor used was called ROPAMS (Remote Object Position Attitude Measurement System), which was based on "measurements made of a mutating magnetic field. The item had a small cord and could be mounted to the finger or wrist." The speech recognition sensor was the DP-100 manufactured by the Nippon Electric Company.

Though Put-That-There was ahead of its time, the gestures were limited to point-and-click and there were a limited framework of actions to enable more functionality. The UI elements were delegated to static icons and figures to demonstrate the functionality of this interface. Ultimately, the user was able to instantiate objects on the screen, such as squares, rectangles, and circles as well as placing them in respective areas of the screen by pointing at the location. This interface helped abstract the specific actions required by a mouse and keyboard and provided the user the ability to point-and-click on maps to move labels and icons.

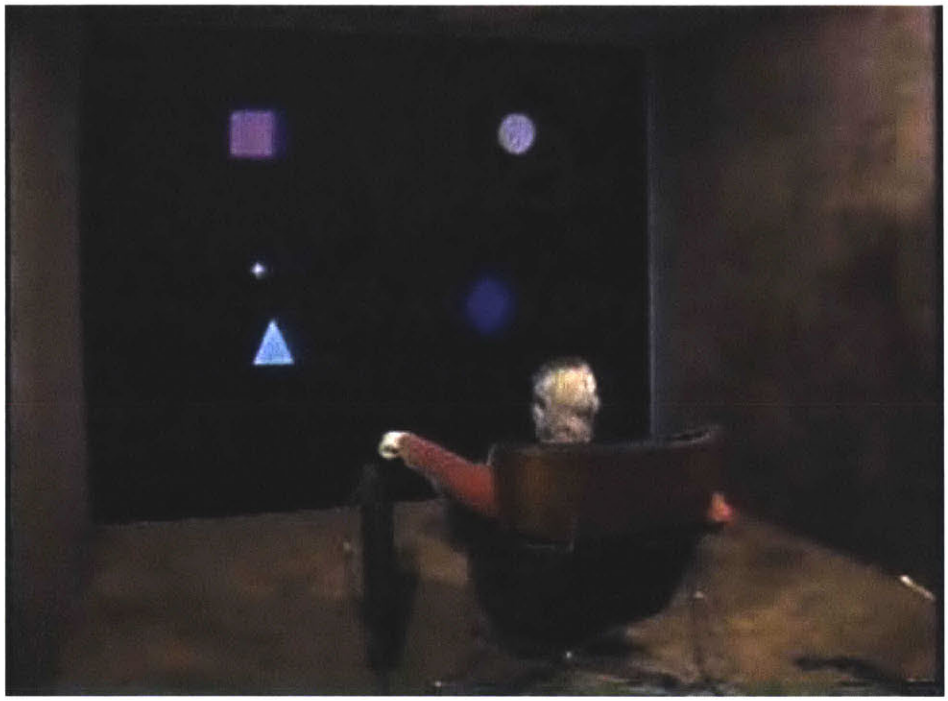


Figure 2-1: Put-that-there

## BumpTop (2006)

In 2006, Anand Agarwala and Ravin Balakrishnan developed a virtual desktop interface called Bumptop [1] at the University of Toronto. The goal was to reimagine the personal desktop interface hierarchy and enable a "piling"-first instead of a "filing"-first approach. This meant that instead of applications and documents hiding inside

folders, they would be piled on top for easier visibility and access. The interface used a pen input and a physics-based simulation to make the UI icons more playful and responsive to the pen inputs. One of the interesting features of their interface was how UI elements took the form of cards which the user throws away or keep. The discarded cards would crumple up and exit to the corner of the display. The prioritized cards were piled closer to the user and enlarged to show significance. We hope to extend this playful methodology in the gestural and voice commands.

## WUW (2009)

In 2009, Pranav Mistry, Pattie Maes, et al. from the Massachusetts Institute of Technology demonstrated a wearable gesture interface called Wear Ur World (WUW) [10]. The idea was to move away from screens and displays and make the physical world a desktop. The user wore a projector around their neck to display the interface onto surfaces and used computer vision to detect hands. These hand gestures, detected via color markers on the user's hands, were postures such as "Namaste" or "pointing" that resulted in altering the UI state. They enabled the user to zoom-in or zoom-out on a Maps application, but only tracked four fingers, index fingers and thumb. Though this work was innovative in its field, it still required the user to wear a projector and labels on the hand.

## G-stalt (2010)

In 2010, Jamie Zigelbaum, Alan Browning, et al. from the Massachusetts Institute of Technology presented a chirocentric (hand-centric) interface, called g-stalt [14], for interacting with video on a large 2D screen. This system was built on top of the g-speak gestural sensing platform by Oblong Industries [12], which used a Vicon motion capture system along with a passive IR retroreflective dots arranged on the back of the hand, the thumb, index, and middle fingers of simple nylon gloves (see Figure 2-2). The actual interface consists of a cubical arrangement of media such

as photos and videos which the user could sort through, play, and reorganize the structure into meaning arrangement. This cube of media could be further rotated and zoomed using a set of hand motions. The gesture set involved actions such as two-handed pinch, telekinetic actions, stop all, lock, and unlock. Though the methods and technology worked well, it came with a price of multiple IR cameras, projectors, and expensive hardware that required calibration.

Figure 2-2: G-speak by Oblong Industries

## Data Mining (2011)

In 2011, Christian Holz from Hasso Plattner Institute and Andrew Wilson from Microsoft Research created Data Miming [4] a system which formed images such as boxes, tables, or office chairs directly from gestures used to describe such objects. The study consisted of two parts: observation of how people naturally use gestures to describe physical objects and a prototype consisting of a Microsoft Kinect camera to capture such gestures. This free-handed gesture detection is similar to what LUI proposes but does not track all necessary fingers and palm vectors. The Data Miming system would compare the captured gestures to a stored list of objects to confirmation

the accuracy. Though the application worked pretty well, the gestures were discrete actions, which caused latency in the response, making the overall experience slower and not real-time.

## SpaceTop (2013)

In 2013, Jinha Lee et al. introduced SpaceTop [9], an integrated 2D and 3D spatial environment on a see-through LCD display to better support spatial memory. The system enabled a more immersive experience for the user, providing the ability to do document editing or 3D modeling without being restricted to a 2D interface. The sensors consisted of one Kinect camera pointed towards the user's head to enable motion parallax and another Kinect camera pointed down at the user's hands to detect position and pinch gestures.

One key interaction method revolved around the position of the user's hands. When the user lifted her hands, the 2D display would fade out or slide up to reveal a 3D interface. Conversely, when the user placed her hands on the surface, the display would revert back to 2D. Similarly, in LUI, the interface only recognizes the hands when they are lifted about the sensor and remains locked when the user removes them. The preliminary user study of SpaceTop showed that users felt comfortable sifting through a pile of documents with one hand while the other was focused on the main task. However, this interface also confused some users who had a hard time switching between the 2D and 3D modes.

## Magic Leap (2018)

Magic Leap [3] is the first consumer-grade Mixed Reality headset to co-locate 3D visualizations with an input framework of depth cameras, eye-tracking, and electromagnetic six degrees-of-freedom (6DOF) spatial tracking. Though the wearable device can detect the hand, all of the interactions occur with the 6DOF remote controller. Though

the controller provides critical feedback when working with the content, gestural and voice interfaces will ultimately provide a key experience if wearable platforms hope to one day just become a pair of glasses. One of LUI's stretch goals is to integrate the gestural and voice system into AR wearables such as the Magic Leap wearable to evaluate its function and usability in augmented and mixed reality.

# Chapter 3

# LUI Design Guidelines

While designing LUI, we came up with several criteria to make the system easily-accessible and scalable, and exhibit low latency. LUI requires only 1 Leap Motion [11] camera sensor (for gestures) and a Google Assistant-enabled [7] smartphone or smartspeaker (for voice) to operate. One could imagine the mobile phone ultimately operating as both the gestural and voice sensor for LUI. As technology progresses, we see depth camera sensors and voice recognition being embedded into large displays and TVs.

## 3.1 Accessible

In most prior art, the gestural interfaces require a specific arrangement of cameras, sensors, and platform-specific software to install and run. From the very beginning, we wanted to incorporate their gestural interface on the web to make it easily-accessible. Other options were developing on a local desktop platform such as Unity, but this requires users to install the software on each device. But with a web application that could be quickly accessed via an URL, the user could interact with the media and content immediately. This could also be quickly deployed in any display connected to the Internet.

For the input camera sensor, LUI requires 1 Leap Motion sensor which could be

plugged in via USB (see Figure 3-1) and a Google Assistant enabled smartphone or smartspeaker (see Figure 3-2). The Leap sensor allows for free-handed gestures (no gloves or markers on hand needed) and is much more precise and cheaper than the Kinect motion sensing or Intel Realsense platform. One of the drawbacks is that the Leap Motion does not provide as much field-of-view (FOV) as the aforementioned devices, but our use case doesn't require high FOV because it is single-user focused. The sensor can be placed on a podium or stand about 8 feet away from the display to mark the placement of the interaction box.

The extremely simple boot up experience of LUI sets apart the interface from past gestural interfaces which require gloves or finger tags, multiple camera calibration, and custom equipment space. In contrast, this interface only needs access to a web browser, an $80 sensor, and a smartphone.

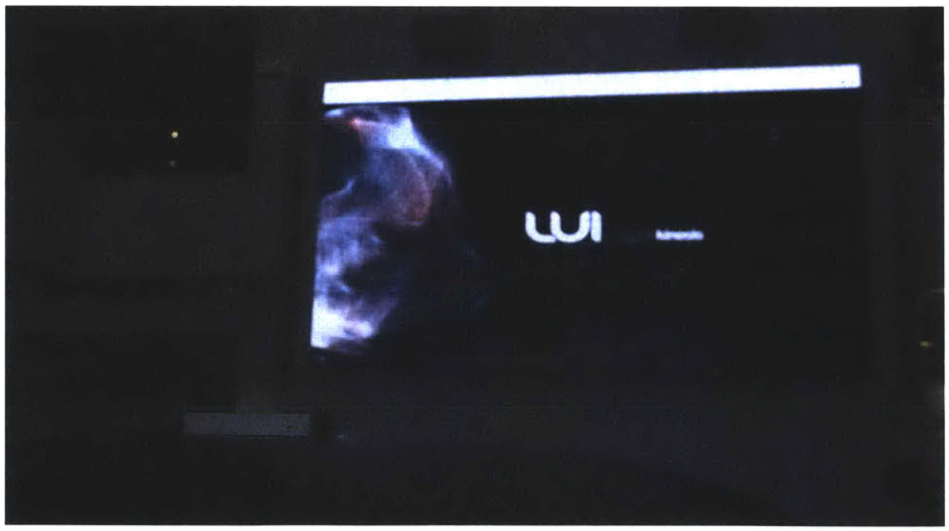


Figure 3-1: Leap Motion

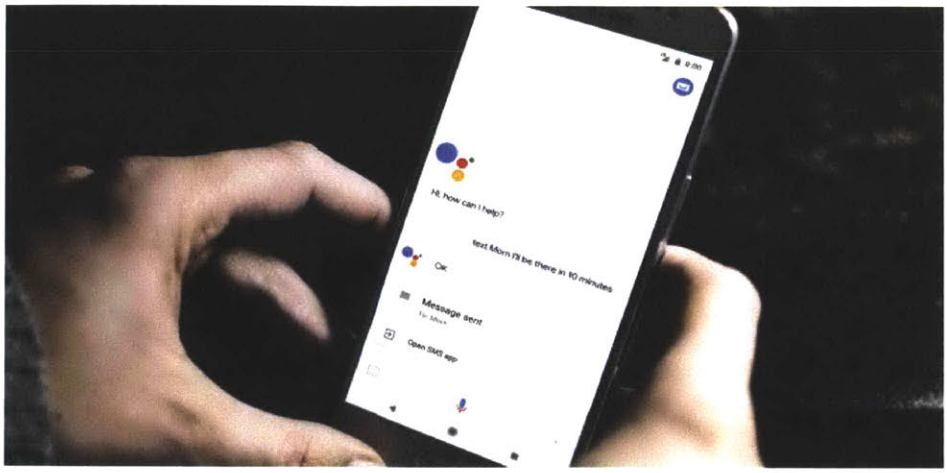



Figure 3-2: Google Assistant

## 3.2 Extensible

The initial design was a static web app where all of the HTML, CSS, and JavaScript was custom. Realizing this would not scale with new gestural and voice applications, we move to the ReactJS framework. The modularity of ReactJS allows the user to add extensions without worrying about the underlying structure of the codebase. ReactJS enables us to create a dynamic single page application, where the hierarchical structure and the modularity of the application allows extend implementation for the developers. User can easily add, delete, and modify individual applications in a few simple steps. Also, each application development cycle is independent from each other so any bugs or issues with one application do not affect the rest of the environment. The web application can be easily converted to any iOS or Android devices using React Native framework.

## 3.3 Non-discrete gestures

Most of the gestural interfaces from prior art are based on discrete actions, where the user makes a gesture and then the computer reacts only after the gesture is completed. This creates latency which makes gestural interfaces not real-time and frustrating to work with. The Leap Motion sensor provides a real-time hand tracking solution

that outperforms every other sensor on the market. With this device, the we can customize the gestures to work off of the continuous finger coordinates instead of discrete gestures.

# Chapter 4

# Software/UI Architecture

LUI consists of a front end ReactJS web interface with dynamic UI elements and animations, a framework of gesture and voice inputs, and a list of real-world applications. The backend processes only the voice inputs via the Google Assistant Firebase database. The system is as modular as possible so more applications could be easily integrated to this platform.

## 4.1 Navigation Schema

This section first describes the navigation schema on how a user would interact with the web application using gestures and voice. As a reminder, LUI requires 1 Leap Motion[11] camera sensor (for gestures) and a Google Assistant-enabled[7] smartphone or smart-speaker (for voice) to operate. Below is the following procedure for booting up LUI. More details of how the gesture and voice pipeline actually work are in Chapter 5.

1. Install Leap Motion v2 software to computer
2. Connect Leap Motion to computer via USB
3. Connect computer to large display via HDMI
4. Ensure Leap Motion is about 8 feet away from the large display

5. Ensure Leap Motion is at least chest height to the user

6. Open LUI website, https://lui-medialab.firebaseapp.com/

7. Connect Google Assistant-enabled smartphone to same Wi-Fi as LUI

8. Login Google Assistant to lui.medialab@gmail.com

The latest version of LUI consists of the following UI tree: Lock Screen, followed by Main Menu, and finally the applications. LUI also fully functions with a mouse. The interface always maps the fingers of the hand directly to the screen via circular markers to provide real-time location feedback. The index finger is considered the default, but it can be changed to user preference.

Since each application is modular and independent from the main menu and other applications, the gesture handling functions can be either largely similar to the handlers in other applications or perform app-specific tasks.

### 4.1.1 Lock Screen

Lock screen is a gentle introduction to the system (see Figure 4-1). It uses particles.js to add in tiny moving particles to the lock screen to add a layer of interaction on top. Simple swiping up motion unlocks the lock screen and leads the user to the menu interface.

When the user extends their hand over the sensor, their fingers are mapped to circular UI elements that hover on the display and follow the fingers. The feedback produces responses in the interface to show that the user is pointing at a specific location on the display. The system currently uses the index finger as the cursor to allow the user to pick an application.

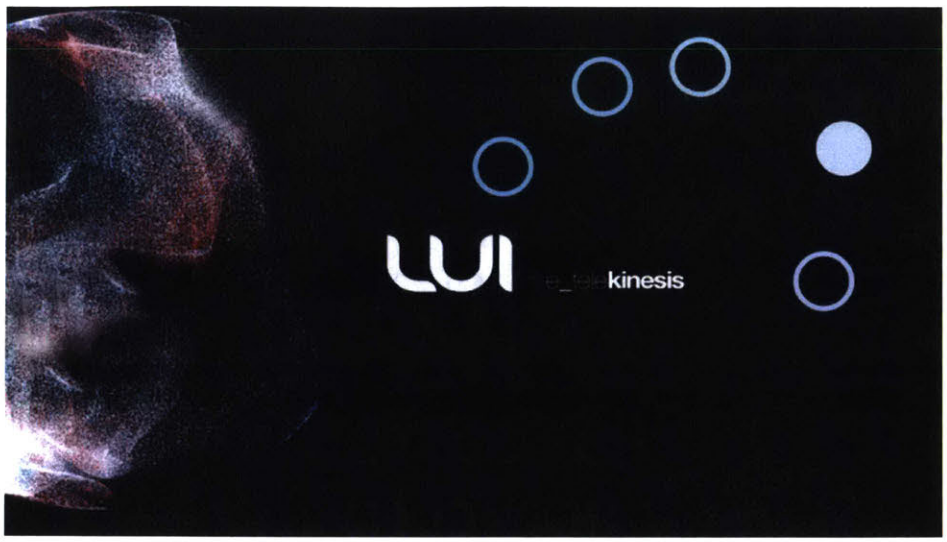



Figure 4-1: Lock Screen

### 4.1.2 Main Menu

The main menu is the page where the user can view all applications integrated into LUI (see Figure 4-2). The current applications integrated are Photos, Videos, and 3D Models, but near future applicatons include Augmented Reality mode, Gesture Keyboard, and Games. The user hovers over specific application to determine which one to enter and air tap an application to view the application in full screen. User can also use swipe up gesture to return to the lock screen. As more applications are added, users will be able search through the list of applications by swiping left and right. The voice command used to open an application is "Go to [Name of Application]."


Figure 4-2: Menu screen

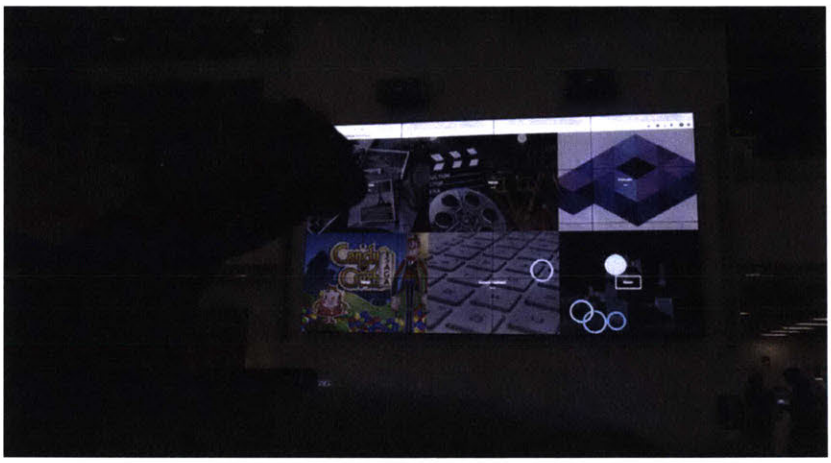


Figure 4-3: Menu screen with hand gesture

In the main menu, hovering the hand over the input sensor changes the state of each app that is visible (see Figure 4-3). Specific apps highlight as the fingers hover over the icon. Whenever there is a gesture action or voice command, LUI provides real time feedback by not only updating the UI elements, but also adding animation. For example, when the user unlocks the lock screen or enters an application from the main menu, the backdrop slides out and the UI zooms into the new page. Applications expand and collapse when user opens up the applications with an air tap and closes with a swipe up motion.

### 4.1.3 Photos

Photos[1], a gallery like UI, is an application core to LUI (see Figure 4-4 and 4-5). It consists of a carousel of photo pages that the user can swipe and select using hover and airtap. As the cursor hovers over the photo, the specific photo enlarges slightly. This application was the first developed for LUI to understand how gestures can browse media content.

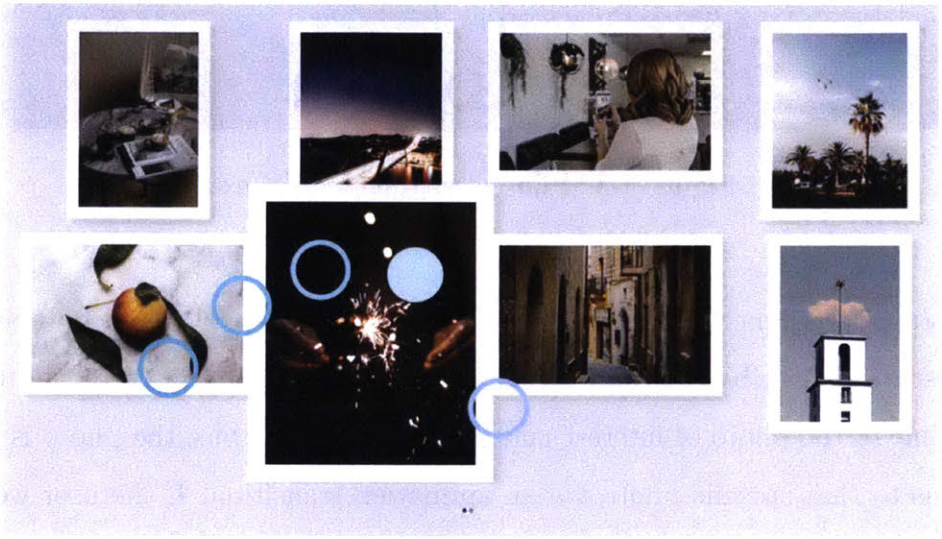


Figure 4-4: Photos

[1]special contributions made by UROP students Ashley JiEun Lee, Kathryn Jin

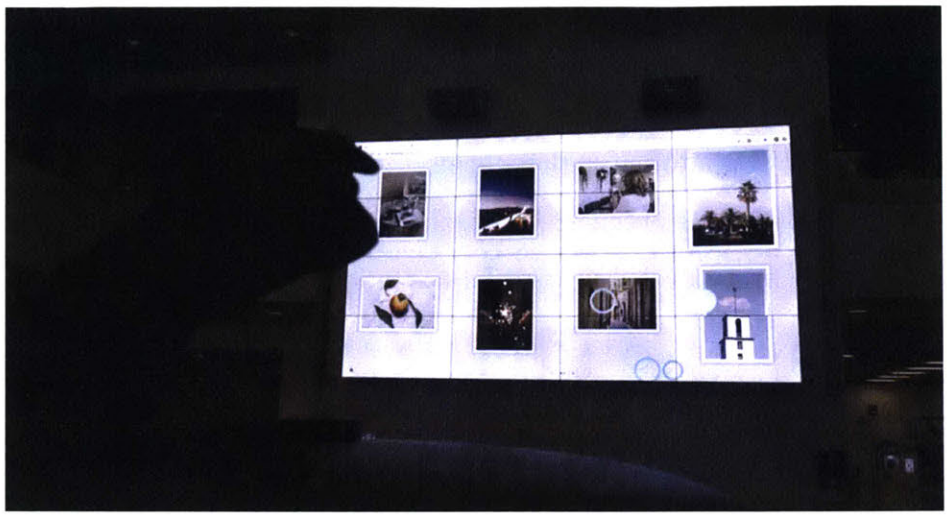


Figure 4-5: Photos with hand gesture

Each photo can assume full screen view by a further airtap or voice command. The airtap gesture is decribed in more detail in Chapter 5. Alternatively, the user can simply point to the photo of interest and say "Open this." Once the photo is selected, LUI zooms to that specific photo via an animation transition. If the user would like to change the photo, they can swipe left or right (see Figure 4-6). To exit the photos app, the user can swipe up or say "Go back" which will trigger LUI to go to the main menu.

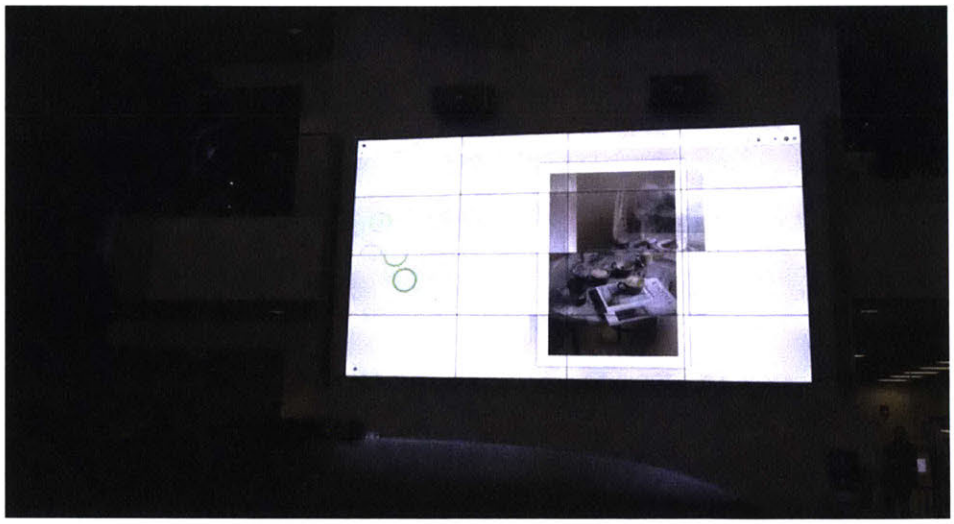


Figure 4-6: Photos single view with hand swipe

The left hand can also be used to change various aspects of the photo such as brightness, contrast, saturation, etc. This is an experimental feature but reveals what can be accomplished with this interface ((see Figure 4-7 and 4-8).

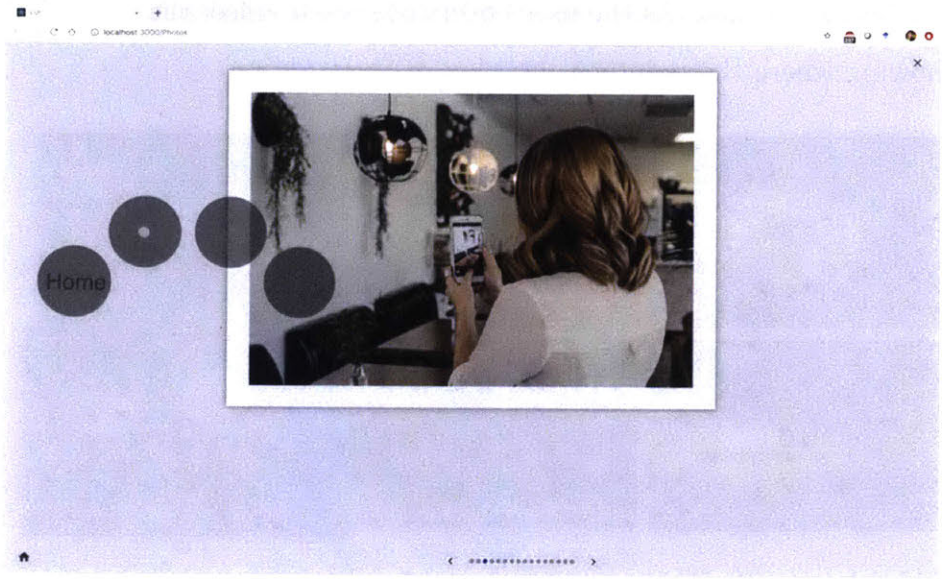



Figure 4-7: Photos option 1

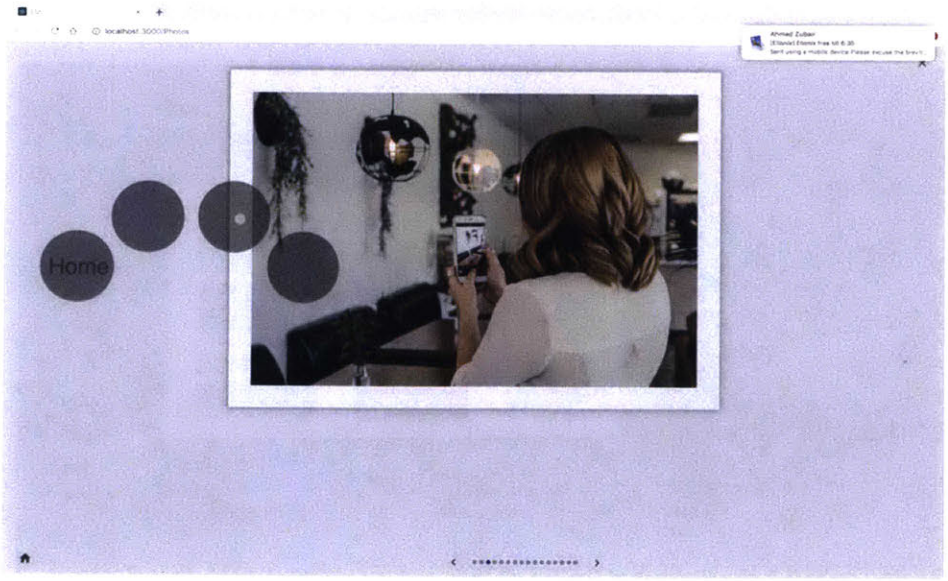



Figure 4-8: Photos option 2

### 4.1.4 Videos

The Videos[2] application is the second application created for LUI (see Figure 4-9 and 4-10). Similar to Photos application, this app also uses a carousel approach to swipe left or right between pages. As the user hovers over each video, the video will slightly enlarge showing where the pointer is at.

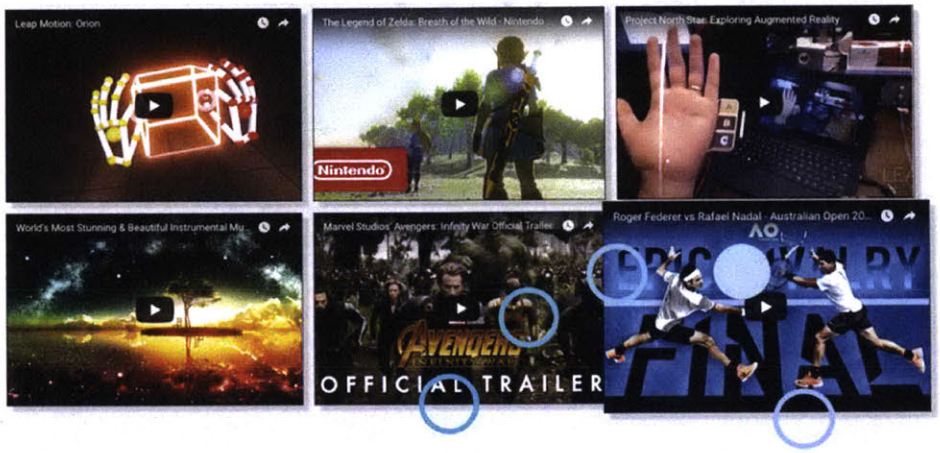



Figure 4-9: Videos


Figure 4-10: Videos with hand gesture

[2]special contributions made by UROP student Claire Tang

To select a video to full screen mode, the user can do an airtap (see Figure 4-11). To move to the next video, the user swipes left or right. To exit video full screen, the user swipes up. To exit the video application entirely, the user must swipe up again.


Figure 4-11: Videos single view

A unique component of the videos app is the abilty to use the left hand to control the volume of each the videos. We added this feature to understand how additional commands could be visualized, besides hovering and selecting, on large displays. While in the single video view, the user rotates the left hand clockwise to increase the volume or counterclockwise to decrease the volume. The visual feedback takes the form of a light blue-colored filled circle that indicated where the left hand was located. The volume feedback is given by a curved slider that increases or decreases in length right above the circle (see Figure 4-12 and 4-13).

Figure 4-12: Videos volume minimize (CCW)

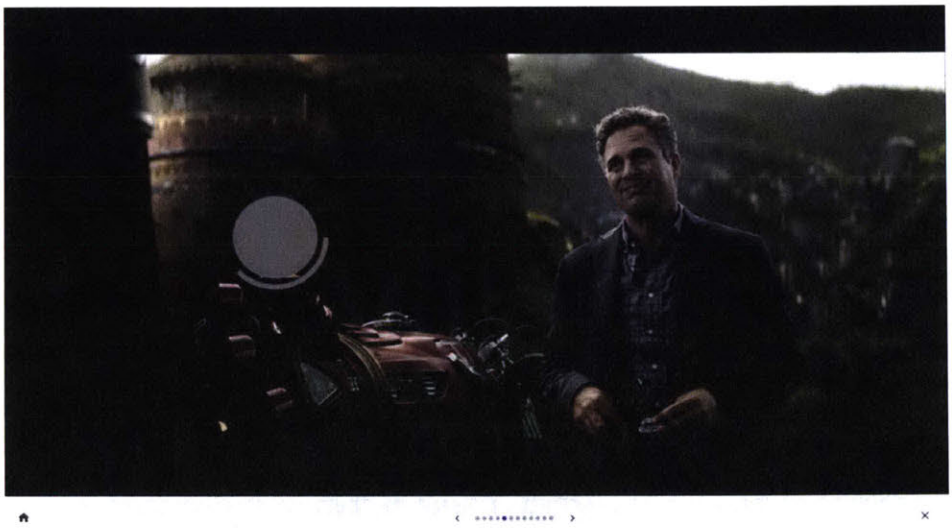


Figure 4-13: Videos volume maximize (CW)

### 4.1.5 3D Models

The 3D Models[3] app is the third application for LUI. The purpose of this app is to understand how to manipulate and view 3D models using gestures. This is the

[3]special contributions made by UROP student Claire Tang

most complex application because it not only uses two hands, but also each hand had a specific function. The content being rendered has multiple axes of orientation making the interaction more difficult to accomplish than the Photos or Videos app.

The models app v1 relies on an open source Javascript library called three.js. Three.js uses WebGL to render 3D models such as FBX fies onto the browser. We integrated this library into our ReactJS platform to give users access to models using gestures. Below is our version 1 attempt to control and view a series of blocks using free handed gestures (see Figure 4-14).

Figure 4-14: Models v1

Figures 4-15 and 4-16 reflect how the right hand interacts with the 3D model. As the hand moves closer to the Leap sensor, the app zooms outs of the model. Inversely, as the hand moves away from the Leap sensor, the app zooms into the model.

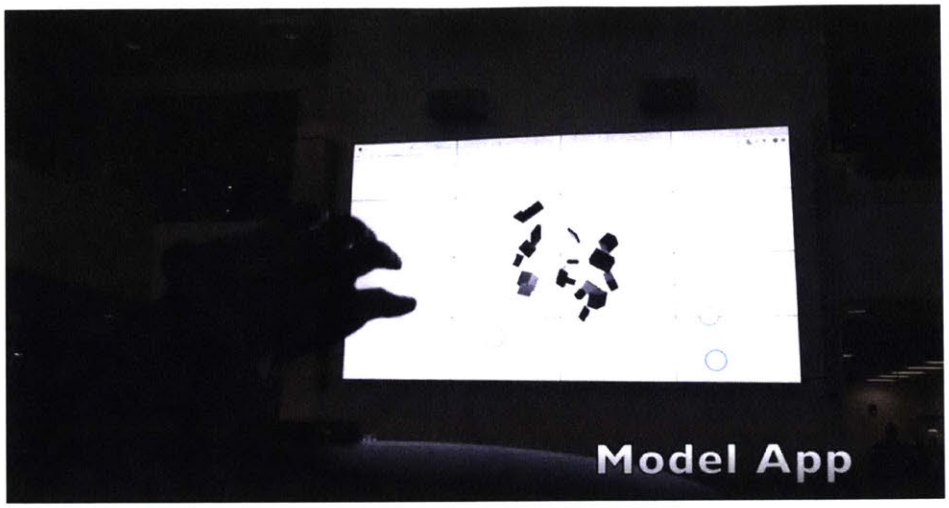



Figure 4-15: Models App Right hand to zoom out

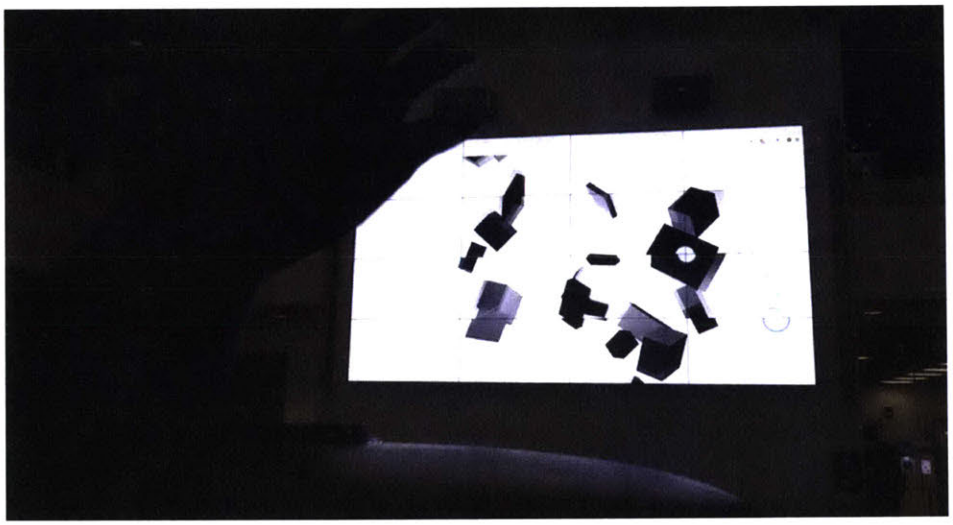


Figure 4-16: Models App Right hand to zoom in

The left hand plays a different role from the right. In the case of the models app, the left hand allows the user to pan around the scene (see Figure 4-17). This provides an additional layer of interaction besides zooming in and out. As the models interaction was being designed, we needed to estabilish a method of interaction which was not immensely involved or required detailed instructions to learn. As a result, we avoided the use of individual fingers to trigger options and rather focused on overall movement of hands (i.e. distance between hand and sensor to zoom and relative distance from

left hand to right hand to pan).

Figure 4-17: Models App Left hand to pan

One of the difficulties of the Models app was determining when LUI should stop recognizing the hands. As long as the sensor sees a pair of hands, it will trigger the interface to operate. One of the ideas moving forward was to create a start command (either using voice or gestures) to begin an action such as browsing content, zooming in, panning, etc).

### 4.1.6 Augmented Reality mode

One of the most exciting aspects of LUI is that it can be ported to any augmented reality or virtual reality environment since the platform is all web-based. Below is an example of the user opening LUI on a Magic Leap wearable device browser[4] (see Figure 4-18). In the current version, the Magic Leap controller takes the place of a gestural input.

[4]special contributions made by UROP student Jeremy Ma

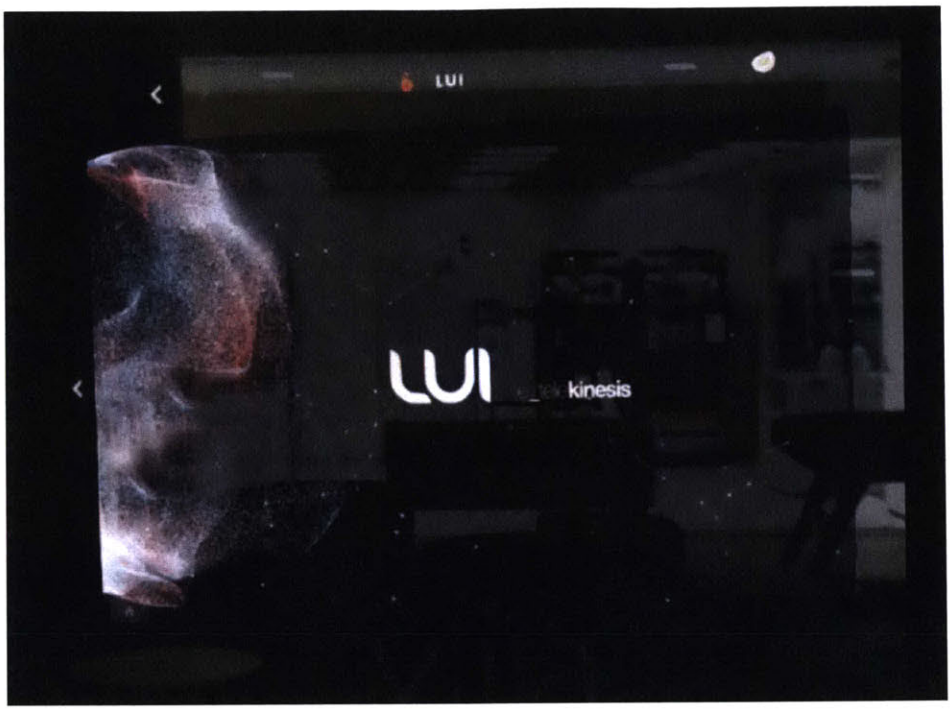



Figure 4-18: LUI in Augmented Reality

The main page is also clearly visible in augmented reality. The hand control currently points to the Photos application (see Figure 4-19).

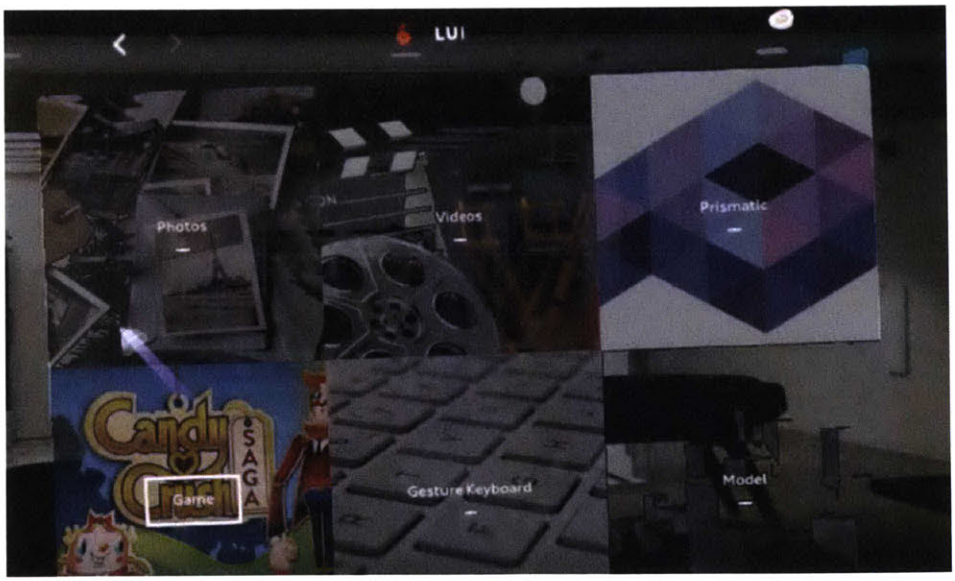



Figure 4-19: LUI Main Menu in Augmented Reality

Prismatic is a Javascript framework exposed by Magic Leap for viewing models in

augmented reality on the web browser. We did a brief integration of this library within the LUI React framework. In this current implementation, we can access a 3D model and animation of a whale by simply navigation to the "Prismatic" app (see Figure 4-20).


Figure 4-20: LUI 3D model in Augmented Reality

## 4.2 ReactJS Framework

ReactJS [13] runs many of the popular websites today, such as Facebook, Instagram, BBC, and Netflix. It allows the user to build complex UIs from small isolated parts of code called "components." For LUI's development, each application consists of its own component (see Figure 4-21). Properties (or props) are used to pass data between each component. There are three main javascript files in the source directory that determine the functionality of LUI. They are index.js, Pages.js, and App.js which are described in the following sections[5]. The other two critical files are leap.js and

[5]special contributions made by UROP student Ashley JiEun Lee

dialogflow.js, which describe the gestural interaction and voice interaction respectively. These last two files will be covered in Chapter 5.

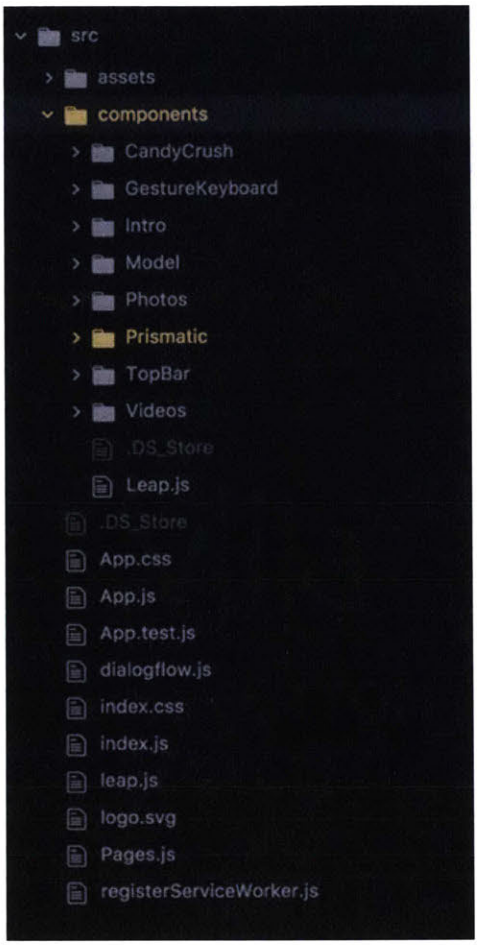



Figure 4-21: LUI source directory

### 4.2.1 Index.js

Similarly to any simple HTML page which contains an index.html page, when you start a ReactJS project, you must have an index.js file to specify the root path to render. In the case of LUI, we first render Pages.js through our index.js file:

```
ReactDOM.render(
    <Pages />,
    document.getElementById('root')
);
```

### 4.2.2 Pages.js

Pages.js will then route all links to each app developed on LUI. In this case, the apps that will be rendered are the Intro menu, Photos menu, Videos menu, GestureKeyboard menu, CandyCrush menu, Model menu, and Prismatic menu. Each menu will have it subsequent index.js.

```
class Pages extends Component {
  render() {
    return (
      <Router>
        <Switch>
          {/* <Route exact path='/' component={Intro}/>
          <Route path='/Home' component={App}/> */}
          <Route exact path='/' render={(props) => <Intro {...props}
              page={"intro"}/>}/>
          <Route path='/Home' component={App}/>
          <Route path='/Photos' component={Photos}/>
          <Route path='/Videos' component={Videos}/>
          <Route path='/GestureKeyboard' component={GestureKeyboard}/>
          <Route path='/CandyCrush' component={CandyCrush}/>
          <Route path='/Model' component={Model} />
          <Route path='/Prismatic' component={Prismatic} />
          <Route path='*' component={App}/>
        </Switch>
      </Router>
    );
  }
}
export default Pages;
```

### 4.2.3 App.js

Menus will include the the lock screen, main menu, and submenus containing each application. Before jumping into the detail of each menu, we must describe how the menus themselves must function. For that, there is another App.js file in the root

folder (complementing index.js and Pages.js) which will do exactly that. This file dictates how the gestures and voice will interact with each app.

```
const firebaseConfig = {
  apiKey: "AIzaSyDjM37_DSv2RvPQzl5YiVzmgRHfpd4rJFU",
  authDomain: "lui-medialab.firebaseapp.com",
  databaseURL: "https://lui-medialab.firebaseio.com",
  projectId: "lui-medialab",
  storageBucket: "lui-medialab.appspot.com",
  messagingSenderId: "247289397118",
  appId: "1:247289397118:web:eb2bcb0076d4bb4d"
};
if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig);
}
var database = firebase.database();
var currentRef = database.ref('voice');


class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      cards: [],
      hovered: "",
      clicked: "",
      page: "home"
    };
  }

componentDidMount() {
    currentRef.update({"current":"home"});
    const cards = [this.refs.card1, this.refs.card2, this.refs.card3,
       this.refs.card4, this.refs.card5, this.refs.card6];
    this.setState({
      cards,
```

```
    exit: false
  })
  //firebase cloud (sockets)
  var something = this;
  currentRef.on('value', function(snapshot) {
    let appClicked;
    var db = snapshot.val();
    var name = db.goto;
    if (db.update){
      if (name === "photos") {
        appClicked = "card1";
        something.setState({ clicked: appClicked });
      } else if (name === "videos") {
        appClicked = "card2";
        something.setState({ clicked: appClicked });
      } else if (name === "prismatic") {
        appClicked = "card3";
        something.setState({ clicked: appClicked });
      } else if (name === "game") {
        appClicked = "card4";
        something.setState({ clicked: appClicked });
      }else if (name === "gesture keyboard") {
        appClicked = "card5";
        something.setState({ clicked: appClicked });
      } else if (name === "model") {
        appClicked = "card6";
        something.setState({ clicked: appClicked });
      } else if (name === "landing") {
        something.setState({ exit: true });
      }
      currentRef.update({"update":false});
    }
    if(db.clicked&&db.hovered!=null){
      something.handleClick(db.hovered);
      currentRef.update({"clicked":false});
    }
```

```
    if (db.back){
      something.setState({ exit: true });
    }
  });
  //voice end
}

handleHover = (card) => {
  // console.log("HOVER", card);
  this.setState({ hovered: card })

}

handleClick = (card) => {
  // console.log("CLICK", card);
  this.setState({ clicked: card })
}

handleExit = () => {
  console.log("Exit to Intro");
  this.setState({ page: "intro" })
  // localStorage.setItem("page", this.state.page);
}

handleUnlock = () => {
  console.log("Unlock to Main");
  this.setState({
    page: "main"
  })
  // localStorage.setItem("page", this.state.page);
}

handleSwipeUp = () => {
  this.setState({ exit: true });

}
```

```
render() {
  const { classes } = this.props;

  if (this.state.exit) {
    console.log("EXITING")
    return <Redirect from="/Home" to="/" />
  }

  return (
    <Wrapper isMounted={this.props.isMounted} exit={this.state.exit}>
    <div>
      <Leap
        cards={this.state.cards}
        clicked={this.state.clicked}
        handleHover={this.handleHover}
        handleClick={this.handleClick}
        handleUnlock={this.handleUnlock}
        handleSwipeUp={this.handleSwipeUp}
        page={this.state.page}
      />

      <Grid className={classes.mainContainer} container>
        <Grid className={classes.rowContainer} container>
          <Grid ref="card1" item xs={4} onClick={() => { this.setState
              ({ clicked: "card1" }) }}
            onMouseEnter={() => { this.setState({ hovered: "card1" })
                }} onMouseLeave={() => { this.setState({ hovered: ""
                }) }} >
            <Photos isMounted = {this.state.clicked === "card1"}
                hovered={this.state.hovered === "card1"} clicked={this
                .state.clicked === "card1"} />
          </Grid>
          <Grid ref="card2" item xs={4} onClick={() => { this.setState
              ({ clicked: "card2" }) }}
```

```
        onMouseEnter={() => { this.setState({ hovered: "card2" })
            }} onMouseLeave={() => { this.setState({ hovered: ""
            }) }} >
          <Videos hovered={this.state.hovered === "card2"} clicked={
            this.state.clicked === "card2"} />
        </Grid>
        <Grid ref="card3" item xs={4} onClick={() => { this.setState
          ({ clicked: "card3" }) }}
          onMouseEnter={() => { this.setState({ hovered: "card3" })
            }} onMouseLeave={() => { this.setState({ hovered: ""
            }) }} >
          <Prismatic hovered={this.state.hovered === "card3"}
            clicked={this.state.clicked === "card3"} />
        </Grid>
      </Grid>

      <Grid className={classes.rowContainer} container>
        <Grid ref="card4" item xs={4} onClick={() => { this.setState
          ({ clicked: "card4" }) }}
          onMouseEnter={() => { this.setState({ hovered: "card4" })
            }} onMouseLeave={() => { this.setState({ hovered: ""
            }) }} >
          <CandyCrush hovered={this.state.hovered === "card4"}
            clicked={false} />
        </Grid>
        <Grid ref="card5" item xs={4} onClick={() => { this.setState
          ({ clicked: "card5" }) }}
          onMouseEnter={() => { this.setState({ hovered: "card5" })
            }} onMouseLeave={() => { this.setState({ hovered: ""
            }) }} >
          <GestureKeyboard hovered={this.state.hovered === "card5"}
            clicked={this.state.clicked === "card5"} />
        </Grid>
        <Grid ref="card6" item xs={4} onClick={() => { this.setState
          ({ clicked: "card6" }) }}
```

```
                onMouseEnter={() => { this.setState({ hovered: "card6" })
                    }} onMouseLeave={() => { this.setState({ hovered: ""
                    }) }} >
                <Model hovered={this.state.hovered === "card6"} clicked={
                    this.state.clicked === "card6"} />
              </Grid>
            </Grid>
          </Grid>

          {/* <Intro page = {this.state.page}/> */}
          {/* <DelayedComponent isMounted={this.state.page === "intro"}
              page={this.state.page} handleUnlock={this.handleUnlock} />
              */}

        </div>
        </Wrapper>
      );
    }
}
export default withStyles(styles)(App);
```

**render()**

The render() function is critical in ReactJS as that dictates what is drawn on the frontend UI. In this App.js, we create a grid which stores information about each card and its gesture and voice status. We also have a  for passing the Leap motion (gesture camera sensor) parameters to the front end.

The states that are passed include "cards," "clicked," "handleHover," "handleClick," "handleUnlock," "handleSwipeUp," and "page" status. handleHover passes information on whether the user is hovering over a certain card. handleClick passes information on whether the user has clicked (or airtapped) a certain card. handleUnlock determines when to set the state back to Main menu. handleSwipeUp is a boolean for determine if a swipe up action has occured.

### Cards

As noted above, we designate a "card" for every app. Each card lists a separate app name and handles a series of commands such as "handleHover," "handleClick," "handleExit," and "handleUnlock." These commands are properties passed from the leap.js file containing the LEAP Motion camera sensor information, which will be described further in the gesture and voice chapter. For example, if a user's hands hover over a card, the "handleHover" property is active and triggers a response on the application UI.

### firebaseConfig

The firebaseConfig sets up the interaction Google Home smartphone or smartspeaker. More info on this will be covered in the gesture and voice chapter.

# Chapter 5

# Gesture and Voice Pipeline

LUI maintains a limited list of gestures and voice to keep the interaction scheme simple. This chapter contains two sections: gesture protocol and voice protocol. For gesture protocol, we first explain how the Leap sensor works, followed by an explanation of the Javascript codebase. Lastly we describe how the gestures were computed. The LUI voice commands currently include "Go to [Application]," "Go back", and "Open this." For this section, we first explain how the Google Assistant operates followed by how we store voice data for retrieval and LUI feedback.

## 5.1 Gesture Protocol

The input camera sensor is a low-cost, off-the-shelf Leap Motion Controller[11] which tracks all 10 fingers, including the palm vector and hand radius. This sensor is comprised of two cameras and three infrared (IR) LEDs. The interaction space is 2ft wide by 2ft long by 2ft deep, resulting in an 8 cubic feet volumetric space which takes shape of an inverted pyramid above the sensor. The current device promotes a 150-degree field-of-view (FOV), but our research was provided a newer model consisting of a 180 FOV for more interaction space in augmented and virtual reality applications. The device is plugged via USB to the TV or computer. We leverage the finger tracking API to create custom gestures mapped to LUI.

### 5.1.1 Leap.js

The Leap Motion archive provides extensive documentation and material to capture this data. The platform is developer friendly and it is quite simple to extract data via Javascript. More information at Leap Motion v2 Developer Archive. Our implementation ported the Leap frames into ReactJS and we computed all the UI elements locally[1].

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom';
import LeapMotion from 'leapjs';

const fingers = ["#9bcfed", "#B2EBF2", "#80DEEA", "#4DD0E1", "#26C6DA"];
const paused_fingers = ["#9bed9b", "#b1f0b1", "#80ea80", "#4ce14c", "#25
    da25"];

class Leap extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            frame: {},
            rightHand: "",
            thumb: "",
            indexFinger: "",
            hovered: "",
            clicked: "",
            pinch: "",
            pause: 4
        }
    }

    componentDidMount() {
        this.leap = LeapMotion.loop((frame) => {
            let hands = frame.hands;
```

[1]special contributions made by UROP student Ashley JiEun Lee

```
    let rightHand = "";
    for (const hand of hands) {
        if (hand.type === "right") {
            rightHand = hand;
        }
    }
    this.setState({
        frame,
        rightHand
    });
    this.traceFingers(frame);
});

this.timer = setInterval(() => {
    if (this.state.pause > 0) {
        this.setState({ pause: this.state.pause - 1 });
    }
    if (this.state.rightHand) {
        var { rightHand, thumb, indexFinger, hovered, clicked,
            pause } = this.state;

        // hovering
        hovered = this.checkHover();
        this.setState({ hovered });
        this.props.handleHover(hovered);

        let gestureDetected = false;
        if (pause === 0) {
            // swipe up
            if (rightHand.palmVelocity[1] > 400) {
                this.props.handleSwipeUp();
                gestureDetected = true;
            }

            // airtap
            if (indexFinger.vel[2] < -300 && (hovered)) {
```

```
                    this.setState({ clicked: hovered })
                    this.props.handleClick(hovered);
                    gestureDetected = true;
                }
            }
            // pause if gesture detected
            if (gestureDetected) {
              this.setState({ pause: 4 });
            }
        }
    }, 100);
}

componentWillUnmount() {
    clearInterval(this.timer);
    this.leap.disconnect();
}

traceFingers(frame) {
    try {
        // TODO: make canvas and ctx global
        const canvas = this.refs.canvas;
        canvas.width = canvas.clientWidth;
        canvas.height = canvas.clientHeight;
        const ctx = canvas.getContext("2d");
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        const { rightHand, pause } = this.state;

        if (rightHand) {
            rightHand.fingers.forEach((pointable) => {
                const color = pause > 0 ? paused_fingers[pointable.
                    type] : fingers[pointable.type];
                const position = pointable.stabilizedTipPosition;
                const normalized = frame.interactionBox.
                    normalizePoint(position);
                const x = ctx.canvas.width * normalized[0];
```

```
                const y = ctx.canvas.height * (1 - normalized[1]);
                const radius = Math.min(20 / Math.abs(pointable.
                    touchDistance), 50);
                this.drawCircle([x, y], radius, color, pointable.
                    type === 1);

                if (pointable.type === 0) {
                    this.setState({
                        thumb: { x, y, vel: pointable.tipVelocity }
                    })
                }
                if (pointable.type === 1) {
                    this.setState({
                        indexFinger: { x, y, vel: pointable.
                            tipVelocity }
                    })
                }
            });
        }
    } catch (err) {
        // console.log("ERR", err);
    }
}

drawCircle(center, radius, color, fill) {
    const canvas = this.refs.canvas;
    const ctx = canvas.getContext("2d");
    ctx.beginPath();
    ctx.arc(center[0], center[1], radius, 0, 2 * Math.PI);
    ctx.closePath();
    ctx.lineWidth = 10;
    if (fill) {
        ctx.fillStyle = color;
        ctx.fill();
    } else {
        ctx.strokeStyle = color;
```

```
            ctx.stroke();
        }
    }

    checkHover() {
        // calculate location of cards
        const cards = this.props.cards;
        const { x, y } = this.state.indexFinger;
        for (let i = 0; i < cards.length; i++) {
            if (cards[i]) {
                const dims = ReactDOM.findDOMNode(cards[i]).
                    getBoundingClientRect();
                if (x > dims.left && x < dims.right &&
                    y > dims.top && y < dims.bottom) {
                    return ("card" + String(i + 1));
                }
            }
        }
        return ("");
    }

    render() {
        const { classes } = this.props;

        return (
            <canvas className={classes.canvas} ref="canvas"></canvas>
        )
    }
}

Leap.propTypes = {
    cards: PropTypes.array,
    handleHover: PropTypes.func,
    handleClick: PropTypes.func,
    handleExit: PropTypes.func
};
```

As seen in the Leap.js file, there is a loop that processes all the Hand frames and we select which hand we want to use for our gesture detection. Currently, we assume the right hand is the dominant hand and we determine if gestures are detected via three main states: pause, hovered, clicked. If there are no gestures detected, the pause is set to 0. If a gesture is detected (airtap, swipe up, etc), pause is set to 4. This is to allow for a delay to prevent gestures being triggered immediately again. There are two other functions called traceFingers() and drawCircle() which are computing the locations and drawing the circles for the visual feedback of the fingers.

Currently, the leap.js is partially replicated within each component, making the codebase more repetitive that it should. For example, the Photos, Videos, Models, etc all each have their own leap.js file. One of the next goals that need to be implemented is to create shared components where the leap.js file is used across all applications.

### 5.1.2 Swipe gesture

The swipe gesture is used to transition between photos, videos, or submenus. Once the stream of data from the Leap Motion Controller is initiated as the user enters LUI, the application will constantly listen to the palm vector to detect swiping motion. If the magnitude of the velocity exceeds the set threshold at any given frame, the system registers a swipe motion (see Figure[2] 5-1), and dispatches an event so that the UI can get updated accordingly based on the direction of the vector. For example, as used in the Photos application, if there is a swiping motion detected in a positive x direction, the gallery will slide right in response. In the future, LUI can listen to the x, y, and z coordinates to make any tweaks to the gestures.

[2]illustrations made by UROP student Dominic Lim Co

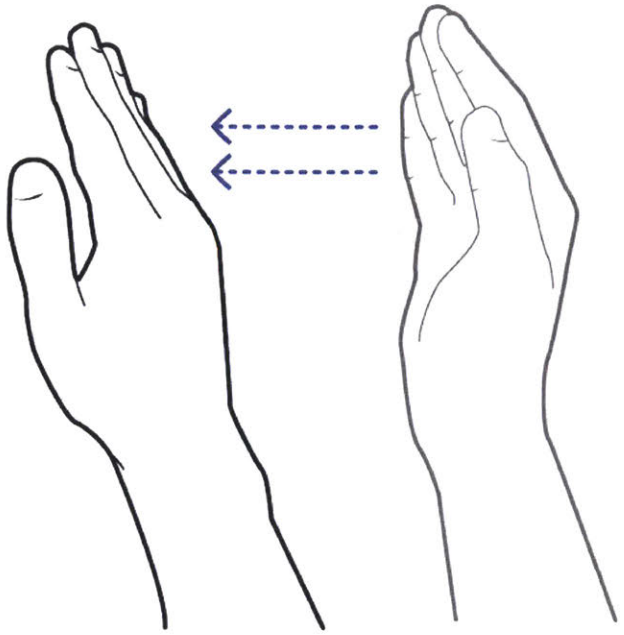


Figure 5-1: Swipe gesture

### 5.1.3 Airtap gesture

The airtap is used to enter a specific application from the Main Menu, "click" on a photo or video, "enlarge" a photo or video, etc. For the airtap gesture (see Figure 5-1), the system will listen to the vector of the index finger. If there is a movement in z direction with velocity above the threshold, the system registers air tap event. Based on the size of the UI and space covered by the leap motion controller, we can calculate the relative coordinate of the finger tip on the screen to figure out which element on the screen is being clicked. From the main menu, the user can air tap one of the applications to enter and explore the application in full screen.

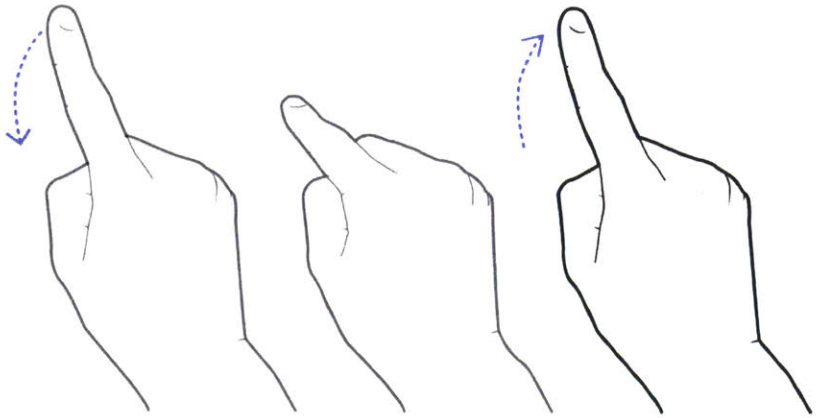


Figure 5-2: Airtap gesture

### 5.1.4 Bloom gesture

Once the user enters an application, the user can exit out of the screen and go back to the main menu with bloom gesture (see Figure 5-3). For the bloom motion, the system listens to the pinch strength of the palm at each frame. Pinch refers to the action of gathering all five fingers by closing the palm. Bloom event is dispatched as the user pinches and then opens up the palm and stretch all fingers in a short frame of time.

After conducting some user studies, many users were having difficulty with this gesture as it placed quite a bit of strain on the hand. Furthermore, the bloom readily got confused with the airtap gesture. As a result, this action was deprecated and replaced with a "swipe up" motion to exit apps.

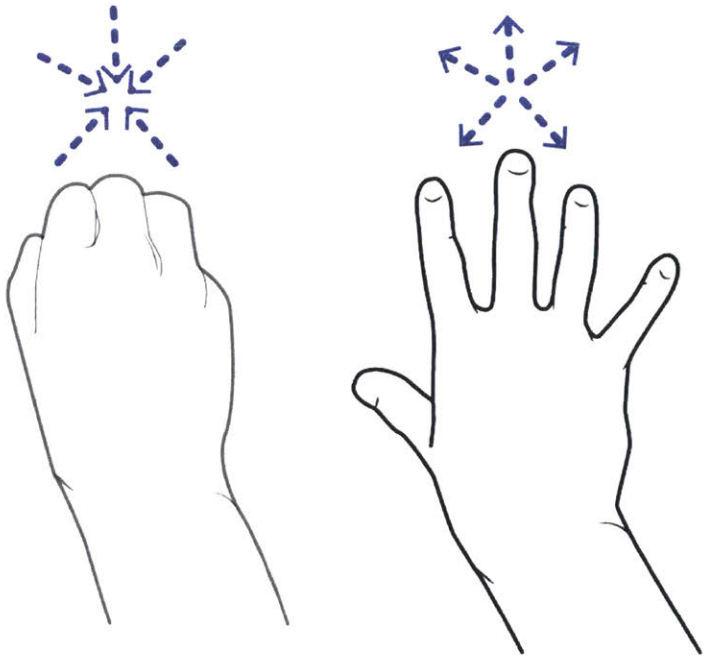


Figure 5-3: Bloom gesture

### 5.1.5 Second hand input

All of the gestures created so far only use one hand. However, when it comes to more complex commands or trying to visualize and control information beyond degree of freedom, it is essential that we have a second hand as well. In LUI, the second hand had multiple functions depending on what app it was in. For the Photos app, the left hand is change saturation, color, brightness. For the Videos app, the second hand was used to control volume up and down. For the Model app, the second hand was used to pan the 3D models. More information is described in Chapter 4.1 Navigation Schema.

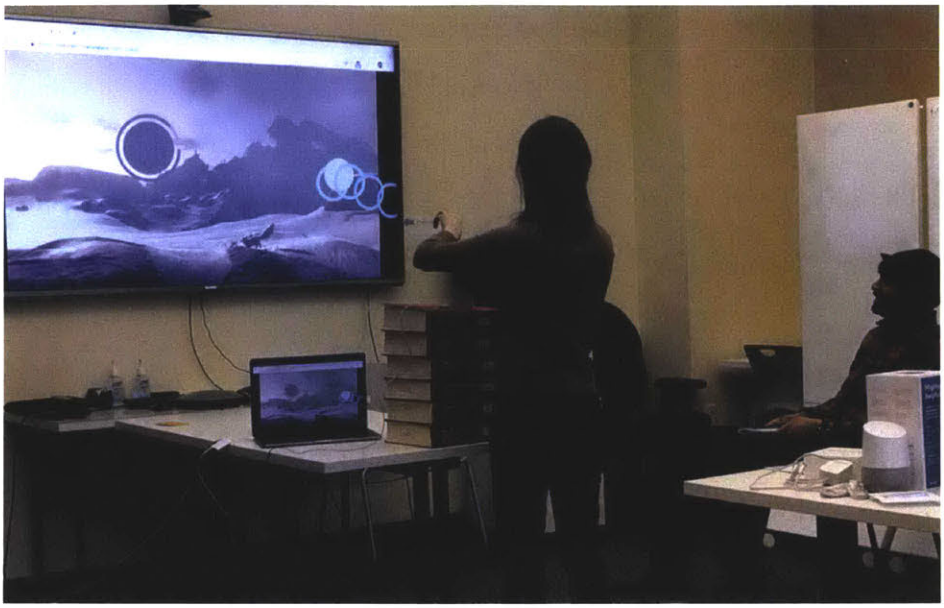


Figure 5-4: Volume control

## 5.2 Voice protocol

The voice application runs on a Google Assistant-enabled[7] smartphone or smartspeaker and a Firebase database[6] integrated to the ReactJS web app (see Figure 5-5). Upon first look, integrating voice seemed a very difficult challenge, but by leveraging the Google Voice Assistant and the Firebase database[3], we could focus on more of the interaction schema with LUI. Making our own voice engine was out of scope for this project.

### 5.2.1 Commands

Upon connection to the same WiFi network as LUI, the user must first tell the Google Assistant the following command:

"Talk to my test app"

The Google Assistant greets the users with "Welcome to LUI," and waits for voice inputs that specifies the name of the application to open. Expected voice intents include:

[3]special contributions made by UROP student Jeremy Ma

"Go to [Application]"

"Go back"

"Open this"

If the system heard the word "Open" but didn't register the actual name of the application, it prompts the users to specify the name of the application to explore.

### 5.2.2 Dialogflow.js

In order for the Google Assistant and LUI to communicate with each other, they leverage Dialogflow[5] (see Figure 5-5), which is Google's natural language processing (NLP) framework. Dialogflow offers a web interface to test if the voice commands being received are correct and provides methods to parse the data. The API matches intent (see Figure 5-6) to fulfillment as shown in code below. In other words, it matches the action being sent ("Go to Photos," "Open this," etc) to an output (in our case LUI). Since LUI requires asynchronomous fullfillments, we must employ "promises."

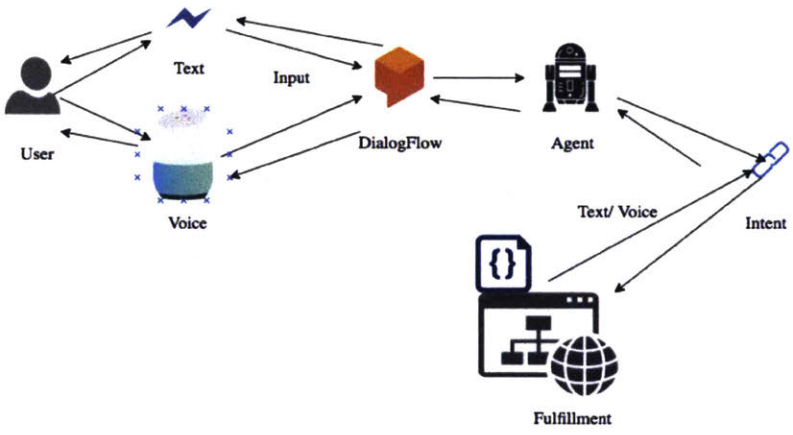



Figure 5-5: Voice pipeline

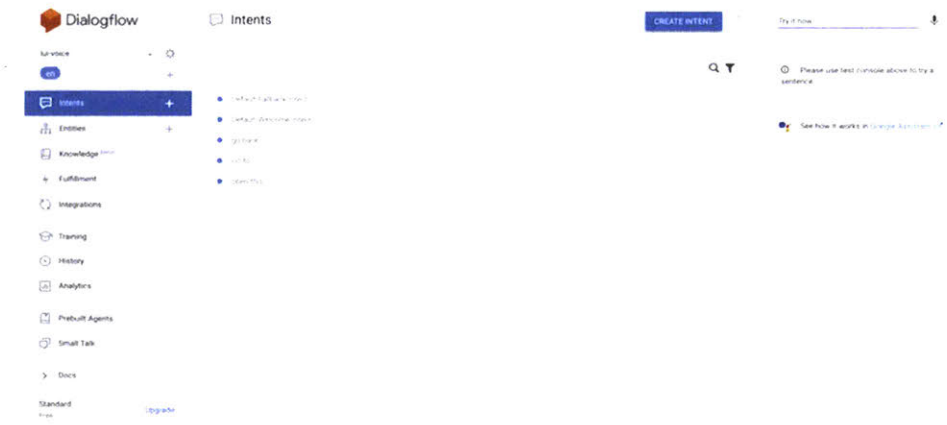



Figure 5-6: Dialogflow Intents

```
const functions = require('firebase-functions');
const {WebhookClient} = require('dialogflow-fulfillment');
const {Card, Suggestion} = require('dialogflow-fulfillment');
const requestNode = require('request');
const rp = require('request-promise');
const NUMBER_ARGUMENT = 'number';

process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging
    statements
exports.dialogflowFirebaseFulfillment = functions.https.onRequest((
    request, response) => {
const agent = new WebhookClient({request, response});
    console.log('Dialogflow Request headers:'+ JSON.stringify(request.
        headers));
    console.log('Dialogflow Request body:' +
    JSON.stringify(request.body));

    function welcome(agent) {
        agent.add('Welcome called');
    }

    function goTo(agent) {
        return new Promise((resolve, reject) =>{
        const options = {
```

```
    url: 'https://lui-medialab.firebaseio.com/voice.json',
    method: 'PATCH',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        "goto": agent.parameters.noun,
        "update": true
    })
};

//get stuff
rp('https://lui-medialab.firebaseio.com/voice.json')
    .then(function (body) {
        const apps = ["photos","videos","prismatic","game","
            model","landing","gesture"];
        var content = JSON.parse(body);
        if ((content.current == "home" && apps.includes(agent.
            parameters.noun))||(agent.parameters.noun == "home"
            && apps.includes(content.current))){
            //patch
            rp(options)
                .then(function (body) {
              if (agent.parameters.noun =="game"){
                agent.add("Game is in development");
              }
              else{
                    agent.add(`going to ` + agent.parameters.
                        noun);
              }
                    resolve();
                })
            .catch(function (err) {
                agent.add(`go to failed `);
                resolve();
            });
```

```
            }
            else{
                rp(options)
                    .then(function (body) {
                        agent.add('Path does not exist');
                        console.log(body);
                        resolve();
                    })
                .catch(function (err) {
                    agent.add('go to failed');
                    resolve();
                });
            }
        });
    });
}

    function open(agent){
    const options = {
        url: 'https://lui-medialab.firebaseio.com/voice.json',
        method: 'PATCH',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            "clicked":true
        })
    };
  rp(options)
    .then(function (body) {
    });
  agent.add('opening');
}

function back(agent){
    const options = {
```

```
            url: 'https://lui-medialab.firebaseio.com/voice.json',
            method: 'PATCH',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                "back":true
            })
        };
        rp(options)
        .then(function (body) {
        });
        agent.add(`going back`);
    }

    function fallback(agent) {
        agent.add(`Invalid command`);
    }

    let intentMap = new Map();
    intentMap.set('Default Welcome Intent', welcome);
    intentMap.set('Default Fallback Intent', fallback);
    intentMap.set('Go to', goTo);
    intentMap.set('open this', open);
    intentMap.set('go back', back);
    agent.handleRequest(intentMap);
});
```

### 5.2.3 Firebase

Once Google Assistant registers the voice input via Dialogflow, LUI extracts the name of the application to open, and saves it to Google Firebase database via PUT requests and websockets. The voice JSON packets can be viewed at `https://lui-medialab.firebaseio.com/voice.json` Any change in the database is detected by the system and the UI updates accordingly (see Figure 5-8 and 5-8).

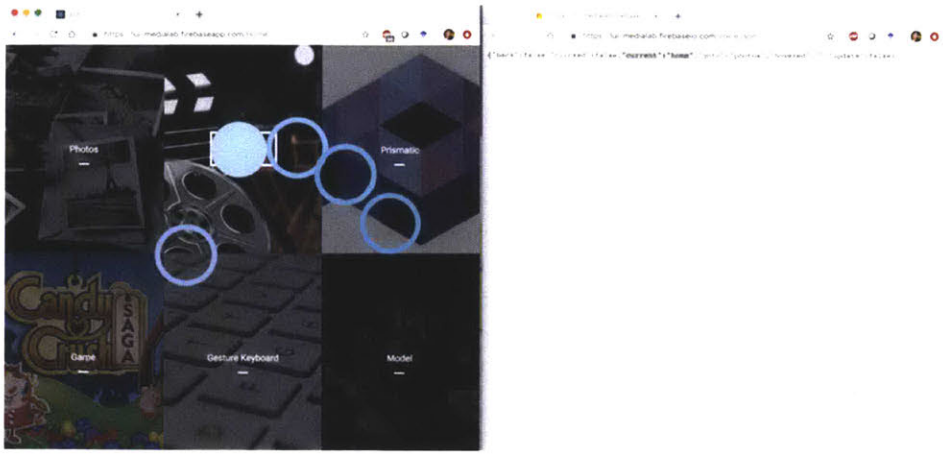



Figure 5-7: Voice current Home

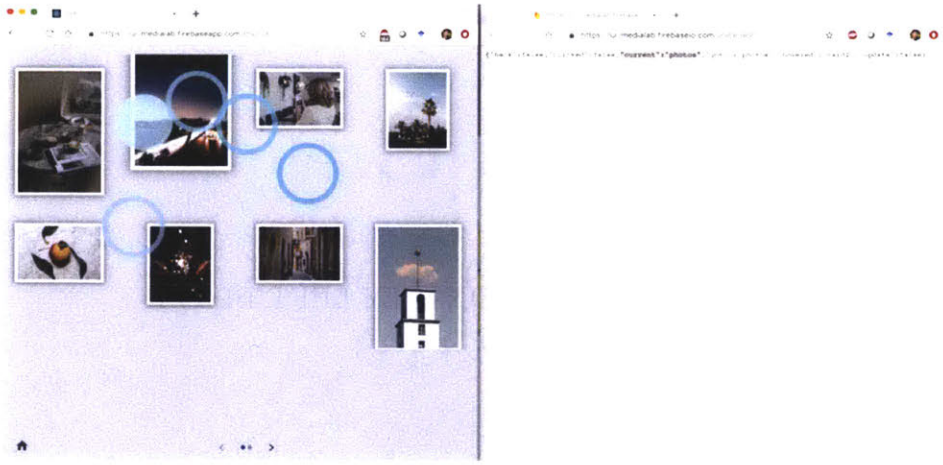


Figure 5-8: Voice current photos

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 6

# Ethics and Evaluation

LUI aims to be useful for people of all age ranges, from children to adults to seniors. At the moment it requires the user to be standing up facing a large display with hands able to freely move. LUI can also function while sitting in a chair as long as the Leap Motion sensor is placed above chest height. It is also important to consider the privacy of the users with a new technology like LUI. This interface requires a camera and voice speaker that constantly observes the user's hands and listens for commands. In our user study, we did not store any data of the hands or of the voice.

The purpose of the study was to discover new methods of how people interact with a large display using gestures and voice. By understanding how people use this interface, we can design novel applications that take into account user preferences, etc. The study took place in front of the display over 20 minutes. The user would engage with the display using their hand movements and voice commands. After the study, 10 minutes will be devoted to survey response.

On the research side, multimodal interfaces have become a core area of interest given the rise of new display technologies and interfaces. We hope this study can teach us about how the field of human-computer interaction is rapidly developing. We are familiar with the mouse and keyboard, but this study goes beyond that and tries to look at the combination of voice and gesture- input.

## 6.1 Setup

The LUI interface was deployed on an 8k large TV with a Leap Motion controller connected via USB and Google Home connected to the internet. The UI web page is accessed via an online web-link (more information in Chapter 4 Navigation Schema). Each subject was given the opportunity to play with the interface without any prior knowledge of how it works. They were then individually primed (shown how to do a swipe, air tap, etc) to navigate the screen using only their hand gestures above the Leap Motion sensor. When they saw their fingers mapped to the UI in real-time, they were then allowed to experiment with various gestures such as pinches and swipes.

## 6.2 Procedure

Below is the procedure used to conduct the user study on LUI.

1. Setup (done by organizer): A large 80inch 8k TV with a camera sensor and voice speaker hooked to the monitor. Note: The camera will not be recording the individual.

2. Setup (done by organizer): Ensure camera sensor is at chest height for user to wave their hands above

3. User approaches the 8k TV and stands 6feet away in front of the camera sensor table

4. User navigates the interface without instruction

5. User navigates the interface using a list of gesture and voice commands

6. User unlocks the screen with gestures

7. User enters an app with gestures

8. User browses the app with gestures

9. User exits the app with gestures

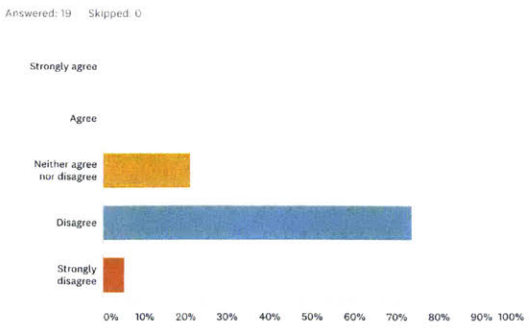
10. User navigates and explores another app with gestures
11. User enters an app with voice
12. User browses the app with voice
13. User exists the app with voice
14. User navigates and explores another app with gesture and voice together
15. User fills out survey (10min)

After the study, users were told they can use voice and given some commands to use. After spending some time getting used to the interface, each subject filled out a survey of questions corresponding to a Likert scale of 5 options ranging from strongly disagree to strongly agree. The total study takes no more than 20 minutes per individual: 10min for the experiment and 10min for the survey.

## 6.3 Survey and Analysis

We first surveyed an initial 10 subjects for beta testing. We did a subsequent study with 10 more subjects all in the same room in collaboration with Simmons University. The participants provided subjective responses via survey of ten questions give immediately after they learned how to use LUI. One of the observations made was that it was much easier to navigate LUI once a user showed the subject how to do it. The experience was akin to showing a person how to use a controller, mouse, or keyboard. The application became more intuitive once the intial list of commands were at least shown by example (swipe, airtap, etc).

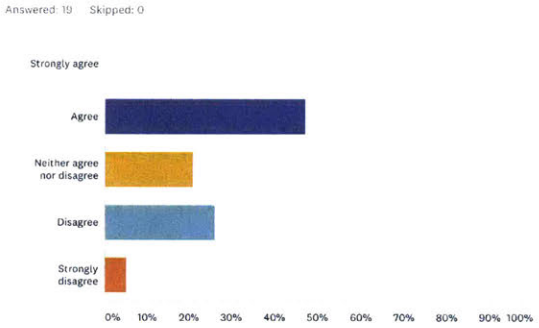
1. I found the system unnecessarily complex.




| ANSWER CHOICES | RESPONSES | |
|---|---|---|
| Strongly agree | 0.00% | 0 |
| Agree | 0.00% | 0 |
| Neither agree nor disagree | 21.05% | 4 |
| Disagree | 73.68% | 14 |
| Strongly disagree | 5.26% | 1 |
| **TOTAL** | | **19** |

Figure 6-1: Question 1

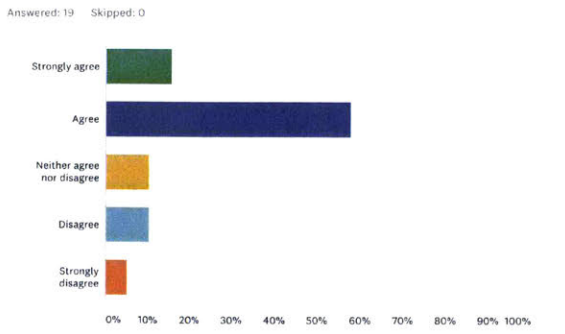
2. I found the system very easy to use.




| ANSWER CHOICES | RESPONSES | |
|---|---|---|
| Strongly agree | 0.00% | 0 |
| Agree | 47.37% | 9 |
| Neither agree nor disagree | 21.05% | 4 |
| Disagree | 26.32% | 5 |
| Strongly disagree | 5.26% | 1 |
| **TOTAL** | | **19** |

Figure 6-2: Question 2

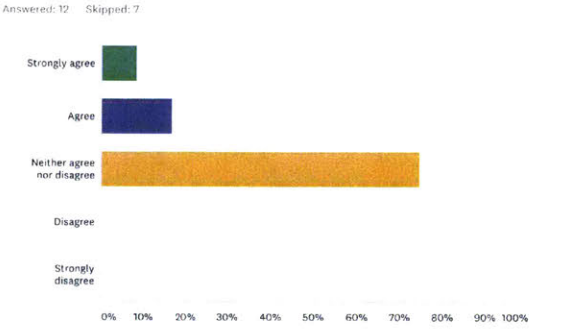
3. I like to use this interface for media browsing (photos, videos, etc).




| ANSWER CHOICES | RESPONSES | |
|---|---|---|
| Strongly agree | 15.79% | 3 |
| Agree | 57.89% | 11 |
| Neither agree nor disagree | 10.53% | 2 |
| Disagree | 10.53% | 2 |
| Strongly disagree | 5.26% | 1 |
| **TOTAL** | | **19** |

Figure 6-3: Question 3

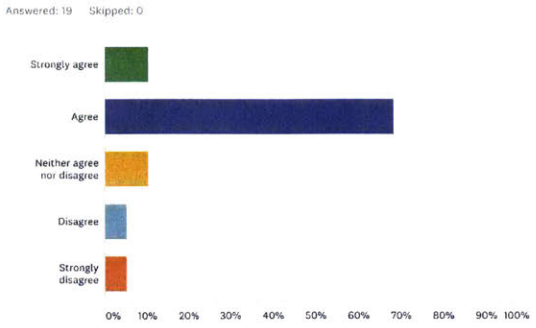
4. Did adding the voice help you navigate the interface better than gestures?




| ANSWER CHOICES | RESPONSES | |
|---|---|---|
| Strongly agree | 8.33% | 1 |
| Agree | 16.67% | 2 |
| Neither agree nor disagree | 75.00% | 9 |
| Disagree | 0.00% | 0 |
| Strongly disagree | 0.00% | 0 |
| **TOTAL** | | **12** |

Figure 6-4: Question 4

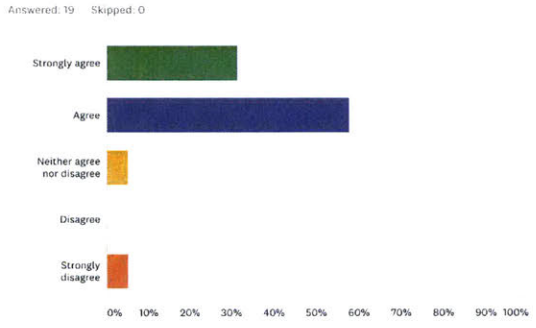
5. Most people would learn to use this system very quickly.




| ANSWER CHOICES | RESPONSES | |
|---|---|---|
| Strongly agree | 10.53% | 2 |
| Agree | 68.42% | 13 |
| Neither agree nor disagree | 10.53% | 2 |
| Disagree | 5.26% | 1 |
| Strongly disagree | 5.26% | 1 |
| **TOTAL** | | **19** |

Figure 6-5: Question 5

6. I would recommend using this kind of interaction for large displays.




| ANSWER CHOICES | RESPONSES | |
|---|---|---|
| Strongly agree | 31.58% | 6 |
| Agree | 57.89% | 11 |
| Neither agree nor disagree | 5.26% | 1 |
| Disagree | 0.00% | 0 |
| Strongly disagree | 5.26% | 1 |
| **TOTAL** | | **19** |

Figure 6-6: Question 6

7. What do you like about this system?

"finger gesture based"

"swipe movement is simple and natural"

"intuitive, responsive"

"don't need extra equipment or sensors"

"being able to use just my hands and not a keyboard and a mouse"

8. What do you dislike about this system?

"arm feels tired"

"Results feel a little unpredictable"

"the learning curve"

"less precise than a mouse"

"gestures could be trained better"

9. How would you improve this system?

"some instruction with swiping, etc to learn swiping"

"arm supports to hold arms up"

"Calibrate the system on a user-by-user basis"

"Let people pick associates gestures for events"

"Differentiate tap slightly, like require tap to be held for some time"

10. Narrate your experience as you would explain the interface to your friend. What you would recommend them to use it for?

"Hold your hand over a certain space and select options on your tv without the need for a remote"

"Swiping gestures to move left or right and a swipe up to exit the screen, and a pinch to expand"

"Gaming, 3D visualization for complex business meetings"

A user subject wrote, "What I enjoyed about this system is the advanced techniques that would be able to be employed by those with disabilities. Creating a hands free system would increase the amount of user possibilities as well as provide freedom to those who may not have the dexterity to use wired devices and accessories." This concept of hands-free gestures was one of the reasons why we made a significant effort in the original design of this interface.

## 6.4 Improvements and Next Steps

LUI is the only low-cost gesture- and voice-based interface that is easily accessible today. In the future, we believe that both the gesture and voice protocol will be supported by the smart-phone itself, thereby eliminating the need for a separate depth camera. We hope more developers will leverage LUI for integrating new applications beyond the scope of this work. We successfully demonstrated photos, videos, and models exploration with gestures and voice, but we are excited to see additional media, content, games, and visualizations developed on this interactive platform.

One could imagine browsing a company's latest products, portfolios, or data visualizations with LUI. Museums and art galleries can leverage LUI to allow spectators not only view but also interact with content. We also saw the power of augmented reality with LUI. By leveraging the web and frameworks already created for the web, we can make AR content easier to create. Since augmented reality is yet to reach mainstream popularity, we believe that LUI can become a platform that powers this future.

Based on our user study feedback, several users mentioned arm fatigue inherent to the system. This was important feedback because the sensor is required to be placed in front of the user asking them to extend their arms at all times. One next step is to make the UI more focused on voice and allow gestural commands only when necessary. Since voice was not well integrated in this study, users had to use gestures for all actions. This may suggest that that a gesture-only interface may not work as a scalable solution.

We have since developed a v2 of LUI based on the feedback of this first user study. Further fixes and updates have been made to the voice input and control, and the bloom gesture was replaced with a swipe up gesture to avoid fatigue. The 3D models application has been revamped to match the photos and videos UI. We now have a carousel of models where each model is selectable using gestures and voice. The user can not only expand or contract each model, they can rotate and pan as well. We

also have a Gesture Keyboard application where the user can paint onto the interface and the Firebase backend uses machine learning to determine what the user has drawn.

What still remains to be done is further optimization of each gesture and voice interaction for a smooth user experience. Due to interest of time another user study has not yet been scheduled. This work hopes to be a starting point for new ideas and development in the near future. One day, many of our interactions will become context-aware and not requiring a controller. Until we achieve telepathy, we believe LUI can support that future.

THIS PAGE INTENTIONALLY LEFT BLANK

# Bibliography


[1] Anand Agarawala and Ravin Balakrishnan. Keepin'it real: pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1283–1292. ACM, 2006.

[2] Richard A Bolt. *"Put-that-there": Voice and gesture at the graphics interface*, volume 14. ACM, 1980.

[3] Gary R Bradski, Samuel A Miller, and Rony Abovitz. Methods and systems for creating virtual and augmented reality, January 28 2016. US Patent App. 14/738,877.

[4] Christian Holz and Andrew Wilson. Data miming: inferring spatial object descriptions from human gesture. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 811–820. ACM, 2011.

[5] Alphabet Inc. Dialogflow | google cloud. https://cloud.google.com/dialogflow/.

[6] Alphabet Inc. Firebase. https://firebase.google.com/.

[7] Alphabet Inc. Google assistant is now available on android and iphone mobiles. https://assistant.google.com/platforms/phones/.

[8] Adam Kendon. The study of gesture: Some remarks on its history. In *Semiotics 1981*, pages 153–164. Springer, 1983.

[9] Jinha Lee, Alex Olwal, Hiroshi Ishii, and Cati Boulanger. Spacetop: integrating 2d and spatial 3d interactions in a see-through desktop environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 189–192. ACM, 2013.

[10] Pranav Mistry, Pattie Maes, and Liyan Chang. Wuw-wear ur world: a wearable gestural interface. In *CHI'09 extended abstracts on Human factors in computing systems*, pages 4111–4116. ACM, 2009.

[11] Leap Motion. Leap motion. https://www.leapmotion.com/.

[12] John S Underkoffler, Kevin T Parent, and Kwindla H Kramer. System and method for gesture based control system, October 6 2009. US Patent 7,598,942.

[13] Jordan Walke. React - a javascript library for building user interfaces. https://reactjs.org/.

[14] Jamie Zigelbaum, Alan Browning, Daniel Leithinger, Olivier Bau, and Hiroshi Ishii. G-stalt: a chirocentric, spatiotemporal, and telekinetic gestural interface. In *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction*, pages 261–264. ACM, 2010.