Tangible: Course Project Final Report

Asim Bhatia, Vaenthan Jeevan, Faiz Momin, Janakitti Ratana-Rueangsri, Anthony Wang

SE 101 — Dec. 2, 2019

1 Introduction

In 2018, Apple ranked fourth in global laptop shipments, capturing nearly 10% of the consumer market with its line of premium and portable notebooks [1]. However, even as many other manufacturers have moved to add touch-screen options to their laptop lineup, Apple has resisted this trend, to the disappointment of many users. In our group alone, only two out of five members had a touch screen computer. This motivated us to explore the concept of a "Universal Touch Screen." We wished to create a hardware add-on that could be installed on any monitor, which could enable touch-functionality. This would allow for the modernization of many old computing setups, without the need to replace all of the original hardware. This motivation drove us to develop Tangible.

2 Background Research

This problem has been explored in a variety of settings with varying outcomes. For example, the *AirBar* is a relatively successful commercial venture that uses infrared lasers to judge distance from a bar installed on the hinge of a 13 or 15 inch laptop [2]. Meanwhile, in academia, Yasin Nils Taib of the University of Gothenburg developed a universal touch screen controller to help interface between a touch screen sensor panel and the cursor of the host system [3]. In addition, several independent projects online combine these two ideas to create a DIY-style device that accomplishes the same task. A prototype by James Alliban was one such example consulted [4]. We note that none of the above approaches differs significantly in true practicality – even the commercial product. This inspired us to try to create something of similar viability, and assess where further improvements could be made.

To do this, we first decided on a sensor type, based on research into the specifications of common Arduino sensors [5, 6]. Ultimately, we elected to use `HC-SR04` ultrasonic sensors for their low cost, simplicity, suitability to distance measurements, and insensitivity to hindering factors [7]. After deciding on a sensor type, we referred to code segments by Dejan Nedelkovski of *How To Mechatronics* to learn how to read inputs from the sensors [8]. Once we could calculate the distance, the remaining challenge was to map these as touch inputs to the screen. These details are discussed in the Implementation section. To control the mouse in some of our features, however, we required the third-party `MouseTo` library by GitHub user `per1234` [9].

Finally, to demonstrate our device's capabilities, we also built several sample apps using the Unity engine. For this, we consulted the Unity User Manual in addition to online video tutorials regarding user interaction [10, 11, 12].

3 Implementation

3.1 Hardware

Tangible uses six `HC-SR04` Ultrasonic Distance Sensors arranged in a row along the top edge of the screen. The sensors are held in place using a cardboard frame that sits atop the monitor. Each sensor has a functional range of 2 cm to 400 cm, and a precision of 3 mm.

3.2 Control

The Tangible hardware is controlled by an Arduino Leonardo. The sensor code (`sensors.ino`) on the Arduino, written in `C++`, is responsible for activating the ultrasonic sensors, as well as reading and processing their inputs. The software is able to calculate the distance from a given sensor to the nearest obstruction (i.e. a finger) by measuring the time it takes for a pulse of sound to return to the sensor after firing. This time delay is multiplied by the speed of sound at room temperature and is divided by two.

Upon startup, the user places their palm at the bottom right corner of their display as the software takes an initial measurement of the user's screen height for calibration by activating the rightmost sensor and storing the measured distance. **Moreover, this calibration process generalizes for any screen height.**

3.3 Signal Processing

Depending on the current context, **the user's touch input is interpreted using either a discrete or continuous method**. For discrete touches, the sensor code divides the user's screen into a 4×6 array, with each of the six sensors being responsible for their respective column. The raw measurement of the user's vertical finger position measured by a given sensor is translated to a position in one of the four cells in a column, by converting that distance into a percentage of the recorded screen height. Distances greater than the screen height are ignored. **This process maps the user's finger position** to a two-dimensional, binary-valued `touchMap` array which is later used to determine coordinates for the cursor, or as virtual buttons to be mapped to app-specific key inputs. In the continuous touch mode, we store position information from the previous loop and compare it to that of the current loop. This allows us to calculate the position derivative $v_{\text{swipe}} = \frac{ds}{dt}$ and execute actions (such as scrolling, dragging, or drawing) in a manner **proportional to the velocity of the gesture**. As an additional preprocessing step, all distances are first normalized to the screen height.

3.4 Signal Noise Reduction

Several techniques were used to improve the reliability of our device and overcome undesired sensor behaviour. For discrete touches, we use Boolean logic on two `bool` arrays, `hasTouchInput` and `alreadyHasTouchInput`, to **determine whether a given input is intentional** or a false positive reading that was previously detected. This helped prevent multiple "phantom" touches, and allowed us to have precise control over when the device was actively looking for a new input.

For continuous touch and swipe gestures, such as velocity-based scrolling, we calculate $v_{\text{swipe}} = \frac{ds}{dt}$ using position information from two consecutive loops. However, the high frequency of loops executed by the Arduino (about 117kHz) resulted in $\frac{ds}{dt} = 0$ between each loop. At the same time, slowing down the refresh rate would cause the scroll movements themselves to appear choppy. Thus, we implemented a **modularly-conditional input scheme** using a variable `loopCounter` $\in \mathbb{Z}_{10}$ so that while the screen would always update, input measurements would only be made 10% of the time. This greatly enhanced the intuitive feel of the swipe gestures.

3.5 Software

To showcase the various features of our device, we **developed a set of custom applications** that are similar to ones found on a typical touchscreen mobile device. The following apps have been integrated into a single home screen interface called *TangibleOS*, in which users can quickly hop between apps and have Tangible **automatically adjust the way it interprets touch input**.

- *Asteroids* is an exciting arcade-style space shooter which allows the user to maneuver a rocket ship in a 2D plane with the objective to try to dodge and destroy asteroids using Tangible's 4×6 grid mapping to simulate on-screen buttons.
- *Jump!* is a minimalist platformer proof-of-concept that also uses a similar button-mapping technique. We worked on adjusting these controls to make gameplay feel more natural to the user.

- *Bin2Dec* is a binary to decimal number converter, and was designed as a convenient utility inside our operating system with a highly accessible input format.
- *TangibleGuitar* features a playable acoustic guitar that users can strum or pluck. The aim of this app is to demonstrate the accuracy of Tangible's continuous touch input on the vertical axis. The sounds for this app were created by recording the individual strings of an actual acoustic guitar.
- *TangibleInk* is a simple drawing app that uses a unique joystick-like input to direct the motion of the virtual paintbrush.
- *HighwaySurfers* is an arcade game in which the player steers their car to avoid crashing into other vehicles on the highway. This game showcases a special application of Tangible's continuous input: simulating a steering wheel. To determine the degree of rotation of this virtual steering wheel, the slope between the measured points of the two hands was calculated and translated into a speed and direction for the in-game car.

4 Group Member Contributions

4.1 Asim Bhatia

- Developed the *Bin2Dec* app
- Described software applications built for the Tangible operating system in report
- Organized the filming of the demo video

Signature

4.2 Vaenthan Jeevarajah

- Developed the *Asteroids* app
- Designed the *TangibleOS* home screen interface
- Edited the demo video using Adobe Premiere Pro and After Effects

---

Signature

4.3 Faiz Momin

- Developed the *HighwaySurfers* app
- Created and refined the steering wheel input mode for an intuitive driving interface
- Filmed the demo video

Signature

4.4 Janakitti Ratana-Rueangsri

- Formatted sensor input for computation and screen/key mapping
- Constructed the mounting structure for the sensors
- Developed the *TangibleGuitar*, *TangibleInk*, and *Jump!* apps

---

Signature

4.5 Anthony Wang

- Typeset prototype proposal
- Implemented presentation mode and vertical scrolling gesture with physics based on finger velocity
- Drafted and typeset final report

Signature

5 Final Product Evaluation

Overall, our final product performs quite impressively. The device is reliable, intuitive, and **supports a variety of gestures including taps, holds, swipes, and scrolls**. We were also able to effectively integrate all of these touch modes into a single "operating system", so that the device can **discern one type of input from another and respond accordingly**. This operating system allowed the user to conveniently access any of these touch modes with ease. One key success indicator is how "natural" the product feels, and often **we would find ourselves touching the screen to scroll a page even when the device wasn't enabled** – this is a clear sign that our device is refined enough to feel instinctive. Compared to our initial proposal, we exceeded all items of our evolutionary prototype plan. In addition, we fully solved two of the major challenges: calibration and cursor control. Nonetheless, the unexpected narrow range of the ultrasonic sensors prevented us from fully addressing the challenges of triangulation and sensor coverage. Due to the physical separation of the transmitter and receiver modules on each sensor, the range limitations of each sensor were compounded. This prevented us from triangulating the finger's precise position from a single row of sensors, and led us to explore the addition of a perpendicular row of sensors to enable triangulation. This is discussed in further detail in the Future Work section.

The ultimate effect of this range limitation was that our device managed an excellent vertical precision, allowing for seamless scroll gestures, but could only read inputs from a set of discrete columns across the horizontal width of the screen. The number of such columns was limited to the number of sensors we could install across the top of the screen; our final product had six. This means that in effect, we could achieve infinitesimal precision with vertical distances but only could only discern a finite number of horizontal values. Despite this limitation, we refined and perfected every aspect of our working method to maximise the precision of our calculations.

6 Design Trade-offs

Over the course of this project, we made several design trade-offs driven by both strategic and logistical considerations. On the strategic side, for instance, we elected to accept a small "deadzone" near the top of the screen adjacent to each sensor. Even though this could have been avoided and better coverage could have been achieved by translating the sensors away from the screen's top edge, we did not want the frame to

add too much bulk to the screen. This was an intentional decision, and a similar consideration must be made if a second axis of sensors is to be added along the side of the screen. There were also some trade-offs that were forced by logistical considerations. For example, we could not add this second axis on our final product due to a lack of pins on our Arduino model. Adding a second line of sensors would increase the analog input load from 6 to 10, exceeding the hardware limitations of the board. Another trade-off that had to be made was to fix the screen size. Originally, we had planned for a telescoping frame that could be resized to fit any screen. However, the angle limitations of the sensors and the lack of 3D printing resources of the required level of sophistication eliminated this option. Finally, in order to create a fluid user experience, we had to design our own "operating system" with a custom graphical user interface, incorporating larger buttons and controls most suited to our hardware.

7 Future Work

We are interested in two potential approaches to improving our product. The first, which is more evolutionary, is to perfect the ultrasonic sensor arrangement. This would mean adding a second line of sensors on one side of the monitor to enable **extreme precision in an extra dimension**. Currently, our device allows for excellent vertical precision, which lends itself very well to an impressively smooth scroll gesture. However, sideways motion of the finger is not as accurately determined, due to the range limitations of the sensors (which is far less than 15° in practice.) This means that we essentially have six discrete columns of touch area across the screen. Thus, adding sensors to the side of the screen would allow us to triangulate the exact position of the finger by using both vertical and horizontal distances to generate an (x, y) coordinate, so the input can be mapped with a **precision of just a few pixels**. However, the drawback to this approach would be to significantly increase the bulk of the device, as well as its hardware cost, rendering it less convenient.

Our second approach involves a more revolutionary enhancement. We would aim to replace our ultrasonic sensing framework with a **computer vision-based one**. The key observation here is that **no extra hardware would be required** - nearly every monitor already comes with a webcam! Instead of an array of cameras, **we could achieve a complete map of the screen with a single convex mirror**, installed over the top edge of the screen in such a way that the webcam is able to see the entire reflection of the screen. We would then be able to use a library such as OpenCV, in conjunction with a vector graphics library, to map the distorted reflection back onto a rectangle, and then calculate the position of a touch input (like a finger) relative to the four corners of the screen. This would allow us to triangulate the finger's exact position in a similar way, but without the hardware requirements demanded by the ultrasonic approach. This improvement is one which we are considering with great interest, as the user simplicity means that it could even be potentially **viable in the consumer market**.

References

[1] B. Lovejoy. Apple moves from #5 to #4 in global laptop shipments. *9to5Mac*, 2018.

[2] AirBar - Made Your Screen Come Alive. *Neonode, Inc.*, 2015.

[3] Y.N. Taib. Universal controller for resistive touch screens. *University of Gothenburg*, 2010.

[4] J. Alliban. Arduino-Flash range sensor experiment. *Wordpress*, 2008.

[5] Distance Sensor Comparison Guide. *SparkFun Electronics.*

[6] Sensor Components. *Arduino*, 2019.

[7] R. Burnett. Ultrasonic vs. Infrared Sensors. *MaxBotix*, 2017.

[8] D. Nedelkovski. Ultrasonic Sensor HC-SR04 and Arduino Tutorial. *How To Mechatronics*, 2015.

[9] `per1234`. `MouseTo` Arduino Library. *Github*, 2019.

[10] Unity User Manual (2019.2). *Unity Technologies*, 2019.

[11] Design and Deploy. Unity 5 - How to Make a Paint Program. *Youtube*, 2016.

[12] Jayanam. Unity Button Click Events. *Youtube*, 2018.