An API For Touch-Free Interfaces For Android Devices

Leeran Raphaely
Department of Computer Science & Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350
lZR@cs.washington.edu

ABSTRACT
There are a multitude of mobile-phone applications that could be made significantly more usable by providing users with an alternative, though similarly expressive, input method to the touchscreen. For example, medical clinicians may find themselves wishing to interact with an application while wearing contaminated gloves. However, with most current APIs used to interface with phone sensors, significant effort on the developer’s part, unrelated to the application’s core functionality, has to be exerted to facilitate a touch-free experience for the end user. In an attempt to ameliorate this problem, this paper describes an Android library created to provide easy, high-level access to basic touch-free gestures without requiring end-users to augment their phone hardware. Any smart phone equipped with a front-facing camera, microphone, and/or accelerometer is able to run applications utilizing this library. From the camera, the library can detect four motion gestures, corresponding to the directions up, down, left, and right. Clicks, the library’s equivalent to on-screen taps, can be recognized by clapping into the microphone, covering the camera, or creating a sudden movement to be picked up by the accelerometer. (Which of these is preferred would depend on external environmental factors.) Motion gestures and clicks can be combined to replicate the basic point and tap functionality of touch screens. Because the new input methods supported by this library replicate touch-screen input, with minimal additions to the code, developers can easily port existing touch-based interfaces into their touch-free alternatives.

Categories and Subject Descriptors
H.5.2 User Interfaces: Input devices and strategies, D.2.2 Design Tools and Techniques: Software libraries.

General Terms
Human Factors, Experimentation.

1. INTRODUCTION
Touch-screens have proven to be an extremely effective input method for smart phones. Using a minimal amount of device real estate, they can provide all the functionality of traditional mouse-and-keyboard input. Unfortunately, there are certain situations that make touching a phone directly unfeasible. One such situation, the very situation that instigated this project, is the case in which a medical clinician uses an app that gives instructions or offers diagnostic assistance during a medical examination or simple procedure. The application’s operator will often wear latex gloves that can make touching a screen difficult. Even if the screen and gloves are able to interact, contamination of both the phone from the gloves and the gloves from the phone is an issue. For such applications, touch-screen input is inadequate. The use of smartphone and tablet recipe applications while in the midst of food preparation is a more mainstream scenario that precludes direct contact with a screen. Messy fingers tainted by ingredients prevent users from scrolling through the steps of a recipe using the swipe-to-scroll interface employed by touch-screens.

Thankfully, smart-phones come equipped with a stockpile of sensors, including cameras, accelerometers, and microphones, that can be used to create alternative forms of interaction with the devices. The Android API gives programmers access to raw data from these sensors, but turning this raw data into application-level interactivity is no trivial task. The goal of this project was to devise a generic, gesture-based input method that, without augmenting a phone’s hardware, could provide a similar level of interactivity to touch-screens.

The central component of this library monitors image data from the front facing camera in real time and interprets four discrete gestures, up, down, left, and right. These gestures can be triggered by moving a finger, hand, or other object across the camera’s field of view, in the desired direction. The distance between the camera and the moving object is unimportant, but users tend to be most confident that their gestures are being recognized when they are within a few inches of the camera. The speed of the gesture is also recorded, which can be used to add further expressivity to the motion commands.

The library also supports “click” commands that can be triggered using the microphone, the accelerometer, or the camera. These clicks can be used to simulate taps on a touch screen, or any other functionality in which a select or action button would be used. Different click sensors can be utilized simultaneously, enabling a programmer to map different sensors to different functionality, however the primary reason multiple sensors are supported is because different environments may be more or less conducive to different sensors. The microphone click looks for sudden amplitude peaks, such as those created by a clap or a snap. This functions extremely well in quieter environments, but if loud noises are unavoidable, then the accelerometer can be used as an alternative click sensor. If the phone is lying flat on a surface, striking the surface near the phone can trigger a click by causing a vibration that is detectable by the accelerometer. This works well assuming the phone is lying on a flat surface that is both stable enough to be hit, but loose enough such that striking it causes the phone to move a detectable amount. Lastly, covering the camera can be used to signal a click as well. This is done by taking the average pixel color per frame and checking if this average is below a certain threshold. When this occurs, the lens is considered covered and a click is registered. This is an adequate click sensor in well-lit environments with non-black backgrounds.

Though developers can interface with the sensors directly to create new tailor made touch-free interfaces, the library also provides classes that help developers easily port existing touchscreen interfaces. One such class draws a cursor that
responds to camera gestures. A click command from one of the aforementioned click sensors will inject a synthetic screen tap into the application at the location of the cursor. Another helper class receives camera gestures as input and injects tap and drag gestures into the application, thus helping developers port touch-screen scrolling functionality into touch-free scrolling.

2. RELATED WORK

The concept of touch-free computing was brought to the public eye in the 2002 science fiction film Minority Report, which featured Tom Cruise’s character operating a computer using graceful, in-air gestures. Consumer devices, most famously Microsoft’s Kinect for the Xbox 360, have turned this concept into a reality for the masses. In addition, touch-free computing has been a topic of interest in the academic community for decades. However, the exact goals, restrictions, and methods of this particular approach have not been fully explored. This project’s emphasis is on simplicity, practicality, and usability, both for developers and end-users. Unlike many past efforts in the field, it was imperative to use only the basic functionality found in smartphones without augmenting them with extra hardware. This project’s methods also do not require any markers or extra props. In addition, this project is designed to be a usable, robust, open-source API, to be made readily available for developers to easily incorporate into any Android applications.

Gesture recognition based on marked gloves has been thoroughly investigated by the academic community. The paper “A Novel Gestural Input Device for Virtual Reality” [Maggioni 1993] details a system called ImageGlove that gleams three-dimensional positional data using a camera and a glove with a special dual-circle pattern on it. The image-analysis process looks for transformations in the pattern and uses that data to create an organic interaction experience for interfacing with computers in three dimensions. The paper “Finger tracking for interaction in augmented environments” [Dorfmuller-Ulhaas 2001] presents another glove-based system, this one using four markers on the index finger. Using their custom patterned glove and a camera, it can reliably derive the position, orientation, and pose of the index finger.

“In-air Typing Interface for Mobile Devices with Vibration Feedback” [Niikura et al. 2010], a paper written by researchers at the Ishikawa Oku Laboratory, describes a project with similar goals to ours. They designed a novel interface that can track a finger in real-time using a smartphone. Unlike the method presented in this paper, their method gathers continuous positional and distance information, allowing a user to click by moving their finger closer to the camera. To track a finger, they use the method detailed in “Fast Finger Tracking System for In-air Typing Interface” [Terajima et al. 2009], another paper from the same laboratory. This method works by “introducing a structural model of a hand and refines the search range by applying pose recognition of the palm and fingers in a stepwise fashion, which enables stable tracking.” Having seen videos of the interface live, this appears to be a very intuitive touch-free gesture system. It is so robust, in fact, that one can fluidly type on an on-screen keyboard without ever touching the screen.

This technique for gesture recognition did not fit the restrictions of this particular project, however. This is because extra hardware that is not especially cheap or readily available is required for its operation. Firstly, it requires a special high frame-rate camera on the phone capable of taking video at 144 fps. Secondly, even when testing on a fairly modern desktop, (Intel Core2Duo E6750 2.66GHz, 2GB memory), the approach was only able to achieve a processing speed of 0.4 seconds per frame, which would not be fast enough to recognize nearly any motion gesture, let alone the stated goal of being able to recognize typing. So, in addition to the external camera, they used additional image processing hardware as well.

Another more viable method of smartphone camera gesture recognition is the one proposed in, “Experiencing real 3D gestural interaction with mobile devices” [Yousefi et al. 2013]. Unlike the previous method, Yousefi’s does not require extra hardware and runs smoothly at least two modern smartphones, the Samsung Galaxy SII and HTC Sensation XL. Their approach “is based on finding the most suitable pattern from a large set of rotational symmetries of different orders that ensures a reliable detector for hand gesture.” This system can recognize the rotation of a hand in a pose in three dimensions, as well as recognizing its 3d position. This is definitely a more complex system than the one proposed here. It is not clear what its error rates are in actual practice.

Leap Motion, a San Francisco based hi-tech company, has already started selling pre-orders of their new device, The Leap, that essentially uses the same technology as Microsoft’s Kinect, but with about 200 times more sensitivity and focused on shorter range, namely, the approximately cubic foot of volume in front of a typical desktop computer screen. For around $70, consumers can purchase their product that essentially replicates all the features of a touchscreen interface.

3. APPROACH

The main code base for this project is contained within a single Android library called TouchFreeLibrary. TouchFreeLibrary’s classes are divided into two different packages: edu.washington.cs.touchfreelibrary.sensors and edu.washington.cs.touchfreelibrary.touchemulation. The sensors package contains lower level classes that provide clients with direct access to gesture and click commands. The touchemulation package contains higher-level classes that use the sensors classes to convert gesture and click commands into touch-screen events. If a programmer wants to create a custom interface specifically designed to take advantage of touch-free input methods, then the classes in the sensors package are all that is needed. However, if a preexisting touch-screen interface already exists, the developers may find it easier to use touchemulation classes on top of their original UI.

The CameraGestureSensor class in the sensors package interfaces directly with the front-facing camera and converts the real-time image data into four motion-based gestures: left, right, up, and down. Moving an object in front of the camera in one of these directions can trigger the different gestures. Any class wishing to receive messages from a CameraGestureSensor object must implement the CameraGestureSensor.Listener interface. To do so requires that the class implement the methods onGestureUp, onGestureDown, onGestureLeft, and onGestureRight. Each of these methods receives, as parameters, the calling CameraGestureSensor as well as the number of milliseconds the gesture took the user to make. Therefore, applications have the capacity to respond differently to
motion gestures depending on their speed. If they so choose, developers can cull extremely fast or extremely slow gestures.

The sensors package consists of three click sensors. Each of the click sensors derives from the abstract base class, ClickSensor. ClickSensor defines all the basic methods required to interact with a click sensor. It defines a Listener interface used by any class wishing to receive click messages from click sensor. Listeners can be attached to ClickSensor objects using the method addClickListener. A ClickSensor can have any number of listeners. This provides developers with the freedom to monitor and/or react to clicks from a single sensor in any number of objects. All classes implementing ClickSensor.Listener must define onSensorClick, which receives the caller object as a parameter. Through this parameter, objects can listen for messages from multiple ClickSensors and respond differently to each.

MicrophoneClickSensor, AccelerometerClickSensor, and CameraGestureSensor are the three subclasses of ClickSensor defined in TouchFreeLibrary. MicrophoneClickSensor recognizes sudden peaks in amplitude, such as those that would be caused by claps, snaps, or knocks. Users can adjust a MicrophoneClickSensor object’s sensitivity, which is defined as the minimum quotient for a click to be recognized when dividing the peak amplitude by the average amplitude. AccelerometerClickSensor objects recognize vibrations when a phone is lying flat on a table. For the time being, it provides no methods for further customization. Finally, CameraGestureSensor recognizes clicks when the mean pixel value of the image data gets below a user-definable threshold. With appropriate lighting and color, this triggers a click when the user covers or cups the camera. Notice that this is the same class that is used to recognize directional gestures from the camera. Because CameraGestureSensor objects take control of the camera, and process images every frame anyways, making them ClickSensors, as opposed to having two separate sensor classes for gestures and clicks that interface with the camera, structurally simplifies the library.

Developers designing new interfaces from scratch are encouraged to utilize the classes provided in the sensors package. One goal of this project, however, has been to facilitate porting touchscreen interfaces into their touch-free alternatives. The touchemulation package contains classes that employ the various sensors to create user interfaces that inject tap events into the application. One such class, GestureCursorController, creates a cursor that can be moved via camera gestures and can inject taps using click sensors. A screenshot of what this cursor looks like operating a test program is shown in Figure 1. A directional gesture causes the cursor to move with a constant velocity in that direction until either a click occurs or a gesture in the opposite direction is sensed. This fairly basic control scheme gives the user the freedom to select any point on the screen using only four discrete motion gestures. GestureCursorController defines a view that contains the single dot. Where the application attaches this view defines the cursor’s positional boundaries. A method, attachToActivity, is provided that sets the cursor’s view to the provided activity.

GestureCursorActivity is a helper class that is provided to simplify the process of attaching a gesture cursor onto an activity. To use this class, developers can simply extend their activities with the GestureCursorActivity, itself a subclass of the Android SDK’s Activity class. They must call initializeTouchFree and specify which type click sensor they want to use at some point, but other than that, GestureCursorActivity handles initializing OpenCV, creating sensor classes, and releasing the camera and other sensors when the activity is no longer in focus.

Finally, GestureScroller is a class designed to translate motion gestures into scrolling commands. GestureScroller stores four points on the screen: top and bottom points, for vertical scrolling, and left and right points, for horizontal scrolling. When it receives a motion gesture, it injects a tap and drag command with a speed proportional to the speed of the motion gesture, relative to two of the four aforementioned points. The coordinates for the points can either be set manually or automatically using the setPointsWithView methods. These methods take a view as a parameter and set the points of the GestureScroller to the edges of that view.

4. IMPLEMENTATION

The first challenge posed by this project was to determine how to best interpret the data from the camera. The idea of having four discrete, directional motion gestures was a requirement that had been in place from the project’s inception. Another requirement was that extra hardware or props were not to be used, or, if necessary, kept to a minimum. Doing full camera tracking without relying on a prop such as a patterned glove seemed complex enough that it was worth exploring a simpler method.
The algorithm we developed works by taking a gray-scale bitmap image from the front camera at every frame, and also storing a gray-scale bitmap of the previous frame. A new bitmap is generated consisting of the absolute value of the difference between the two aforementioned bitmaps. This third bitmap shows at which pixels movement occurred in the frame. The program then calculates the average coordinates of all the pixels where movement occurs, as well as the percentage of the bitmap in which movement is detected.

We are now left with a single coordinate of where movement is centered in the image and a percentage of the image in which movement occurs. The program looks for the moment that that percentage gets above a certain threshold, and considers that the start of a gesture. Once the percentage drops back below the threshold, the movement is considered completed. If a distance in one of the axes is covered that is above a threshold, a gesture message in the direction of the movement is sent to all listener objects. Though the algorithm is quite simple, both computationally and logically, it accomplishes all of the requirements of the projects while meeting the imposed constraints. Because the algorithm looks for movement alone, as opposed to color or shape, it is extremely flexible. Any object, including a human hand regardless of skin-tone or covering, can be used.

One way the algorithm’s flexibility becomes evident to the user is that, discounting the use of any non-anatomical apparatus, there is still a wide variety of viable gestures that can be performed, some of which function more consistently than others. For example, the whole arm could be used to perform gestures, but a hand or even single finger works as well. Users can be centimeters away from the camera, or they can be several feet. Having worked with the system for quite a bit, we feel that the simplest and most consistent mode of operation is to use a single finger about an inch or two away from the camera. It is as important to know when the gesturing object is inside the camera’s field of vision as it is to know when the object is outside. To make two gestures to the right, a user must position their object on the left side of the FOV, slide the object to the right, then while outside the FOV, move their object back to the left side of the camera, and slide to the right again. If the user tries to make two gestures to the right, and moves the object back across within the camera’s FOV, the system will register the input as being one to the right, one to the left, and one to the right. With a single finger close to the camera makes it very easy to tell when a gesture is being picked up by the system and when it is not.

Though the algorithm has many advantages, it is not without its faults. Arguably, the biggest of these is that it does not actually analyze the image data to the extent that it can tell which direction motion is happening if the motion occupies the entire screen. For example, if the object being detected is covering the entire frame and then moved upward, a person would easily be able to recognize the direction of movement, but without a part of the object leaving the frame, the system would not be able to discern a gesture from this data. Another disadvantage of the algorithm is that it does not give continuous movement data, which limits the expressivity of any interface programmed using this library.

None of the click sensors use especially novel algorithms, but they are proven to be functional enough nevertheless. MicrophoneClickSensor works by reading in audio samples in chunks, then averaging out the amplitude in each chunk. The average of the average amplitudes of the last eight chunks is updated after every chunk is read. As soon as a chunk with average amplitude eight times the average of the last eight chunks is found, the system monitors to see if the amplitude chunks return to a range somewhere within the previous average. If so, a click message is sent to all listeners. Fast Fourier Transforms (FFTs) were evaluated to try to isolate the frequencies of a clap, however these attempts functioned as well as or often worse than the current simpler approach.

AccelerometerClickSensor also works with an extremely simple mechanism that is nonetheless functional. The primary downside of the current method is that it only works if the phone is lying flat on a surface. To check that that is indeed the case, the algorithm scans the accelerometer data and will not send a click command unless the data coming from the x- and y-axes are within a very close range of zero. When that is the case, and the phone is at rest, the only force in the z-axis should be the force of gravity pulling downwards. If this force is disrupted, then we can assume some entity has shaken the phone vertically. This occurs, for example, when the phone is on a less than perfectly stable surface that is struck or tapped. Once this is recognized, a click message is sent to all listeners.

5. EVALUATION

To evaluate this project, we created an extensive teaching and testing application that allowed us to rapidly gather data about the project’s performance. Our test population was a diverse, twelve-person group that included friends, family, and fellow students. Upon running the application, the user is presented with a tutorial that teaches the basic concepts of the gesture control system. After the program gives them a chance to practice their newfound skills, it tests how well they grasped the concepts and utilized the system. These tests are presented in a game like fashion, which was originally done to coerce the user into operating the system to the very best of their abilities. Having witnessed a dozen people run the test, we believe that, for the most part, this resulted in worse, not better, performance. This is because many users tend to get anxious when playing games and felt a false sense of urgency while performing the tests, even when they were explicitly told that they were not being timed.

The initial tutorial section of the program presents the user with several animations depicting a hand performing the four directional gestures over the camera. See Figure 2 for a screenshot of one of these animations. Having seen beginners use the system in pilot tests, we made sure to emphasize the important fact that the hand should be out of the camera’s view unless the user intentionally wants a gesture performed. Each of the animations explicitly depicts a hand ducking out of the way of the camera until it is positioned to make a gesture.

Once the user has mimicked the motions to a satisfactory extent, they are brought to a practice round for the first test. In this test, they are presented with arrows pointing either left, right, up, or down. They must then make a gesture in the direction of the arrow. See Figure 3 for a screenshot of this mode in action. A score is kept showing how many times they were successfully gesture in the direction indicated by the arrow. Though we did record how long it took for each gesture to be made, there is no visual indication that the user is being timed. In fact, the program itself suggests that the user take as much time as needed since speed is not represented in any summary of the user’s score.
The results of the twelve person test has shown that users, on average, manage to perform 40.71 gestures correctly and 9.29 gestures incorrectly. This result was substantially worse than what we expected. The program does not store which results belong to which users, but I suspect that the lower than desired score may stem from utilizing several subjects very inexperienced with smart phones and user interface technologies different than a typical desktop. However, they could very well be more representative of our eventual end users, who will not necessarily have had a large amount of exposure to technology. We also timed each gesture performed during the test, and found that users spent an average of 1.533 seconds on each gesture. This is more in accordance with what we expected.

After this test, the user is introduced to the concept of clapping to trigger the click sensor. We decided not to include either the accelerometer clicking method or the color clicking method as we were convinced that the environments that the program would be tested in would be relatively quiet and could not begin to guess on the types of surfaces available. Thus, we also opted against using the accelerometer click-sensor. We have also found that the color clicking method is highly variable in effectiveness due to environmental factors of lighting and background. Therefore, the MicrophoneClickSensor seemed to be the best to test.

Immediately after being introduced to the clap-to-click technique, the user begins to manipulate the cursor. To practice their cursor maneuvering, they are asked to play a game in which they must move the cursor inside randomly placed boxes. Once they are properly in position, they receive a point as soon as they “click” by clapping. This simulates the act of tapping buttons on the screen using the cursor. The program keeps track of how many moves were required by users to get the cursor inside the box. A
move is defined as any completed gesture. For any one round, the user should be able to optimally click the boxes in two or fewer moves. The experiments show that users take, on average, 3.3 moves to maneuver into a box. We consider this to be a fairly successful result. Unfortunately, we have also found that it takes users an average of 7.66 seconds to click on a randomly placed box. Since any user can easily tap a button on the screen in under a second, it seems that using the gesture control scheme simply takes too long to be a viable replacement for anything but the most basic touch-screen applications.

Having observed each of the twelve participants, we must report that moving the cursor with gestures was a frustrating task for most users. However, simply making the gestures was fairly intuitive. For this reason, we believe that the cursor interface may be a little too difficult in its current state for the average user but shows promise if improvements are made.

6. CONCLUSION AND FUTURE WORK
The work on this project represents a solid first step towards the creation of a good API for touch-free computing that is both user-friendly as well as developer-friendly. Having used the library as a developer fairly extensively, we are quite confident that the overall structure of the system has and will continue to prove itself to be simple, flexible, and expandable. We tried throughout the creation of this library to develop a system that could be both powerful and simple to use, depending on how much control the user needs. While developing the testing application, requirements for the system were often challenging and several sub problems had to be solved to meet the demanding constraints on the system’s operation. One subtle way the library empowers programmers is that each of the sensors can be set to send their messages to multiple listeners. When developing the testing program, this was an invaluable feature, as it enabled us to create listener classes that reacted to the sensor messages, and other listener classes that just listened in and recorded data based on the sensors (very useful for developing our test applications).

Though this project certainly has had a measure of success, there are still some areas that could be explored further in order to greatly improve the user experience of touch-free interfaces. One technique that could be applied to this project with the potential for greatly improving the system is machine-learning. This could be used, for example, to determine exactly what makes a user’s clap unique and only send click messages when a matching sound is heard. They could also be used to adjust the response from the camera gesture system. For example, the system could notice when the user turns their head and use this information to optimize the detection process.

We are not fully convinced that the gesture interface is the best technique to emulate touch-screen functionality. In testing, users had a great deal of difficulty trying to reliably move the cursor. A further issue with the cursor is that it allows the user to click anywhere on the screen, however, most of the screen does not usually contain items of interest. A superior method for emulating the most commonly used touch screen functionality, namely, interacting with Views within an Activity, would consist of an overlay that highlights the views of an activity and allows the user to select one of these views using the left, right, up, and down gestures. When a click sensor triggers a tap, the system should react appropriately depending on the type of view. A button, for example, would just get clicked, either with an onClick or an injection. If the user selects a text box and clicks, however, then a custom keyboard would appear with a highlighted letter, which could allow the user to type short amounts of text. Under the current cursor system, typing, unfortunately is near impossible due to the poor maneuverability of the cursor.

Every class and all public and protected methods within the TouchFreeLibrary have been documented using Javadocs.

7. ACKNOWLEDGMENTS
I would like to thank each of the grad students who worked with me on this project, including Sidhant Gupta, Mayank Goel, and especially Nicola Dell, who answered my questions when I had them, met with me to make sure I was staying on track, and most importantly, gave me the chance to work very independently. Of course, I would like to thank my professor, Gaetano Borriello, who turned me on to the project to begin with.

8. REFERENCES