Aalto University
School of Science
Master's Programme in ICT Innovation

Wang Chen

SwipeLauncher:

Fast Application Switch for Mobile Multitask Optimization

Master's Thesis
Espoo, August 2, 2015

Supervisor: Professor Antti Oulasvirta, Aalto University School of Electrical Engineering
Instructor: Anna Maria Feit, M.Sc.(Tech)

Aalto University
School of Science

Aalto University School of Science Master's Programme in ICT Innovation	ABSTRACT OF MASTER'S THESIS

Author: Wang Chen

Title: SwipeLauncher: Fast Application Switch for Mobile Multitask Optimization

Number of pages: 69	Date: August 2, 2015	Language: English
Professorship: Communications and networking	Code: S-38	

Supervisor: Professor Antti Oulasvirta

Instructor: Anna Maria Feit, M.Sc.(Tech)

Abstract:

The growing of functions of smart phones brings the need of handling multitasking behavior on mobile devices. Nowadays, mobile multitasking takes quite some steps, especially while looking for a less frequently used application, which brings the need of optimizing mobile multitasking to make it smoother and faster.

In this thesis, I looked into the collaboration of interface design and prediction algorithm, aiming at providing a better mobile multitasking solution which is easy to control and fast to use. My work mainly includes two aspects: new gesture control method for reducing the operation steps while switching between multiple applications; new application recommendation algorithm to reduce the time and effort of reaching target applications.

I applied my findings to a mobile application launcher, SwipeLauncher, which defined a new gesture control method with fast access to certain applications. To validate my design, I first used model-based method for making design decisions, including the menu layout design and number of applications in the launcher.

Afterwards, I conducted an experiment with SwipeLauncher and Android home screen's performance for opening designated applications. The evaluation indicators include operation speed, accuracy, system complexity and user comfort. SwipeLauncher gets better performance in most dimensions: using SwipeLauncher is 34.2% faster than Android home screen, with lower error rate and ignore rate. SwipeLauncher can be used as an alternative of Android home screen in the future for mobile multitasking optimization.

Keywords: multitasking, mobile, gesture, model-based design

Acknowledgements

I wish to thank my supervisor Prof. Antti Oulasvirta for offering me the opportunity of working on this exciting master thesis topic; inspiring me for creative solutions of problem solving; and providing me all the helpful resources in time.

Thanks my advisor Anna Feit for guiding me through the actual operation process as well as the thesis writing process. She gave me lots of useful suggestions. I'm also grateful to Yunhe Guo for helping me during the developing process, and turning my prototype into a real working mobile application.

I would also like to thank the volunteers for participating in my experiment. They not only provided me with data for product evaluation, but also gave me valuable feedbacks for possible future work.

And thanks to my coordinators, Stefanie Schulz and Aino Roms, their help makes the thesis project process clearer and smoother.

Thanks to all the members at User Interface Research Lab for the inspirations during the group meetings.

Last but not the least, I appreciate my parents for supporting my study and caring every part of my life.

Espoo, August 2, 2015

Wang Chen

Contents

Chapter 1

Introduction

The way of using smart phones has changed a lot in the past few years, with their functions and interaction method disruptively developed. The huge amount of available applications greatly enriched the functions of smart phones, extending smart phone usage method while accompanied with problems of handling multitasking.

According to *Mobile Technology Fact Sheet* [16] , over 50% smartphone users download applications and use their smart phones for extended functions, besides basic functions like making calls and sending text messages. The most popular services of smart phones include surfing the internet, reading email, getting directions and listening to the music.

However, the current mobile multitasking experience is far from perfect. According to *Accenture Video Solutions Survey 2013* [2], only 6% users think multitasking on mobile devices is "very satisfactory" when accessing video services, 39% think it is "satisfactory", other 55% people think it is "not satisfactory at all" or "somewhat satisfactory". Therefore, there should be huge space for optimizing multitask interaction on mobile devices.

1.1 Mobile Multitasking

In my thesis work, "mobile multitasking" typically refers to controlling more than one application on mobile, including switching between opened applications and launching a new application. Here are two scenarios explaining the current mobile multitasking situation:

- User A is on the bus, looking at Google Map to ensure he will not miss his station. Meanwhile he is also listening to the music, while switching between Facebook to see friends' recent activities. When he switches

back to Google Map, he does not know why the application has been refreshed and he loses his previous route information.

- User B is using Adobe Reader for reading a document on her smart phone. Since there are some unfamiliar words in the document, she needs to use Google Translate to check the meaning of the words. For each iteration, she needs to go to application list, search for the Google Translate icon, open it and translate, then repeat the same steps to go back to the document. These steps take her quite some time, and each time when she returns to the document, she needs to look over for her paused point.

1.2 Launching Application on Mobile Devices

Current smart phones provide three solutions for opening an application:

- **Home screen**, in which users can choose certain applications and put them in customized order;
- **Application list**, which lists all applications in a fixed order (in most cases, by alphabet order);
- **Overview window**, which lists recent applications in usage order.

As we can see, opening an application inside application list always takes some time looking for an application among dozens of applications, though it is a "safe option" which guarantee the success of finding the application. The overview window is faster for reaching the most recent applications, but since it keeps floating all the time, it becomes uncomfortable when users need to switch between more than 2 applications, or finding an earlier application.

While compared with desktop solutions, since the need of desktop multitasking occurs much earlier, and more works have been done in this field, there are more completed solutions for desktop multitasking. Common desktop multitasking solutions includes task thumbnails in the task bar, using shortcut keys to switch between opened tasks, spliting the screen by multiple tasks, etc. However, directly copying desktop multitasking to smart phone does not work well, mainly for the following reasons:

- **Different Screen Size**
  Small screens limit the amount of displayed information, and also make it difficult to handle two or more tasks on the foreground at the same time, as spliting the screen will make the small mobile screen even

smaller. On the other hand, it still needs to keep a comfortable size of operation target, in order to get acceptable targeting time and click accuracy.

- **Different Interaction Methods**
  Interaction on desktop nowadays highly relays on keyboard and mouse, which is totally different on smart phone. Recent mobile interaction includes finger touch (also multi-touch), physical buttons, and sometimes voice recognition and sensor interaction. Different interaction methods will therefore leads to different multitasking manifestations.

1.3 Research Problems

Based on previous analysis, mobile multitasking problems can be concluded into three aspects, which are not really solved by current technologies:

- **Visual search**
  It takes time to look for an application icon, especially applications that are not often used.

- **Performance**
  Switching between applications is not comfortable enough, it always requires several clicks on different screen positions;

- **Ergonomics**
  Some applications (usually the left top ones on the screen for right-handed users) cannot be reached under single hand operation, especially on large mobile screens.

As a conclusion, the goal of the research is to design a new launcher, which makes it easier and faster to launch applications on mobile devices.

1.4 Scope of the Work

Based on the three current mobile multitasking interaction problems mentioned in section **1.3 Research Problems**, I am aiming at designing a new application launcher, which makes it faster and easier to switch between applications, by predicting the applications will be used next, thus providing quick access to them. My work mainly includes these two aspects:

- **A new gesture-evoked launcher**, which provides a more natural interaction experience, while shortening item selection time. This launcher is different with current launchers, but it is very easy to learn and works faster than existing ones;
- **A new mode for showing applications in the launcher**, which considers both the algorithm accuracy and user predictability. In particular, it is not only about the highest success rate of finding the target application in the launcher, but also for the users to be aware of which applications will appear in the launcher before open it.

1.5 Outline

In the previous sections, I stated the current limitation of mobile multitasking and design opportunities. The following sections will focus on detailed problem solving methods and approaches.

In chapter **2 Related Work**, I will analyze previous work in both launcher interface design and algorithm prediction. For launcher interface design, I will focus on operation gesture and layout design, which I want to optimize in my launcher; on algorithm prediction aspect, I will pick up one of the algorithms with highest accuracy for my launcher prediction.

Chapter **3 Design Foundation** introduces the most important elements I consider as the foundation for my design. I will analyze the key aspects for optimizing mobile multitasking behavior, which should be considered for my launcher design. I will also list the essential functions of the launcher, and arrange a full design space based on the functions. The design space provides a qualitative criteria for the following work.

Chapter **4 Design Process** is the core work of the thesis, in which I defined my design space as these aspects: 1) Visual Layout. Inculdes launcher visual expression and menu layout. 2) Gesture. Inculdes the gesture of calling out menu, exit menu and select an item among the menu; 3) Menu Parameters. Includes number of applications in the menu, which applications will appear in the menu and in which order. For Visual Layout and Gesture design, I will do several rounds of design sketching and design critique, and get the best design. For Menu Parameters, I will first analyze the interaction process for opening an application through my launcher with KLM model, and decomposed into serveal specific steps; then for each step, I will measure operation time through theoretical or experimental approaches, and compare total operation time under different menu parameters and select the best design.

Chapter **5 Modeling Based Design Decisions** focuses on the launcher layout decisions, finds out the best launcher design and compares my launcher with traditional Android home screen. In this chapter, I will discuss the menu layout designs based on previous chapters. The decision is based on the operation time modeling, which includes KLM modeling, visual searching time and pointing time prediction. Afterwards, the comparison between my launcher and Android home screen operation is aiming at providing the convenience of SwipeLauncher.

After decided final best design, I will evaluate my launcher in chapter **6 Experiment**, in which I did a user experiment to see how the launcher works in the real world and how it compares with existing launchers, in order to verify its usability.

Finally, in chapter **7 Discussion and Conclusion** I will talk about the strength and limitation about the new launcher, look into the expected usage scenarios and possible future works.

Chapter 2

Related Work

There are quite some works about optimizing multitask behavior, existing works can be roughly divided into the following three categories:

- **Algorithm Prediction**
  Focus on predicting user behavior, in particular the applications with highest possibility to be used next, and put these applications at positions which are easiest to reach. Since algorithm need certain amount of data to make precise prediction, it usually takes time to collect data and learn user behavior, the accuracy is relatively low at beginning.
- **Menu Layout**
  Focus on how to use least steps and shortest time to reach an application. These methods also consider how to organize menu contents on an interface, and how to switch between interfaces.
- **Gesture Control**
  Use specific gestures to control a menu. These methods always use the combination of several unique gestures to select an item in the menu quickly. Gesture controlled menus always need longer learning time for new users, but operation time always shortens significantly for advanced users.

2.1 Algorithm Prediction

Algorithm prediction solutions aim at "let application with highest usage possibility appears at easiest operation position". It uses previous user operation data for predicting following user behavior. There are mainly two kinds of prediction algorithms: **Context Aware Based**, and **Application Usage Sequence Based**.

2.1.1 Context Aware Based

Shin et al. [19] designed a new context model, which collects a wide range of contextual information on the smart phone, and makes personalized application predictions based on a Naive Bayes Model. Shin et al. found these three issues: last application, cell id, hour of day. They are the most important factors when predicting next applications to be used.

Based on this context aware model, Shin et al. developed *Dynamic Home Screen* application, which is a customized home screen launcher with the top possibility applications (predicted by algorithm). The applications on this home screen are ordered by predicted possibility. Among the applications, the one with biggest possibility increase is highlighted.

Since only a few applications will appear on the Dynamic Home Screen, it is relatively easier and faster to find an application among them. However, since the order of applications keeps changing, users need to adjust to a new menu each time. In this way, visual search time and target pointing time in the launcher will not decrease much with user learning.

Yan et al. [23] designed a system, FALCON, which uses contexts such as location and temporal access patterns to predictively launch applications before they occur, thereby reduces perceived delay for users. FALCON adaptively balances latency reduction benefit with energy launch costs. It represented the predicted applications as pure text, which is slower than icon (image) recognition, but does not support opening the application through widget.

FALCON appears as a smart phone widget, which displays the predicted applications. When users open an application which needs long loading time, opening time reduces significantly because FALCON has launched the application in advance. But for small applications, the optimization is not significant. On the other hand, users could not open an application directly through FALCON widget, but still need to open from application list. Figure 2.1 shows the screen shots for context aware solutions.

2.1.2 Application Usage Sequence Based

Parate et al. [15] mitigated long network content retrieval times by accurately predicting which applications will be used, and prefetching their application content in order to reduce operation time. They developed an Android widget, AppSensor, which not only provides the top possibility applications, but also pre-fetch them to shorten users' waiting time while opening new applications. Their algorithm treats each application as a "character" in a word, and an application usage sequence as a "word". The algorithm uses longest

preceding character (application) sequence to compute the conditional probability for the following character (application).

Similar with Dynamic Home Screen, the applications and their order keeps changing all the time, which takes users longer time to find a target application in the list.

Figure 2.1: Screenshots of Dynamic Home Screen, PREPP widget and FALCON

2.2 Menu Layout

A good menu layout design should be convenient to enter and exit, and easy to find an application in the menu. In particular, aiming at using least click and shortest time to open an application on mobile.

Android uses an overview window to solve this problem. [11] As shown in figure 2.2, an overview window provides fast access to the recently used applications. The applications are ordered by usage time, represented by an application icon and interface thumbnail. Since an interface thumbnail is much bigger than a single icon, it is faster and easier to select an application in the overview window rather than in the application list. What is more, the possibility of selecting a recently used application is always high, since the applications used together always have high relevance with each other.

However, since the order of applications in the overview window keeps changing according to application usage, users could not remember or predict which applications will appear in the overview window, which leads to high failure rate (of finding an application in the overview window).

Samsung provides a different solution by allowing two or more windows sharing a screen, of which each screen size is adjustable. [10] This solution is

especially suitable for switching frequently between two applications, since it provides two screen views at the same time.

Split the screen by multiple windows is a traditional desktop multitask solution. However, since mobile screen is relatively smaller, separating multiple windows on small screens makes each window even smaller, reduces content area size and increases operation difficulty. What is more, the operation is difficult under one-hand control mode.

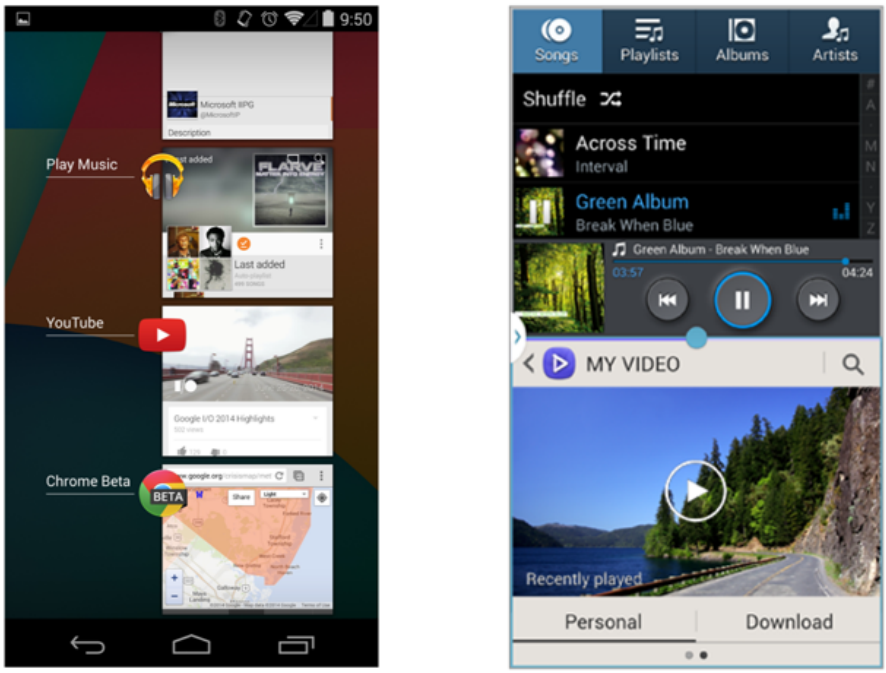


Figure 2.2: Android 4.4 on Nexus 5 (left) and Samsung TouchWiz Nature UX 3.0 on Galaxy Note 3 (right)

2.3 Gesture Control

Gesture control methods focus on how to select certain item by simplest gesture and least operation steps.

2.3.1 FastTap

FastTap [12] interface uses two fingers of one hand for menu selection. The thumb selects first level menu item, and forefinger selects secondary menu item. This menu works pretty fast for skilled users, and it also supports pressing two fingers at the same time, without expanding the whole menu, which even improves operation speed.

Figure 2.3 shows the interaction of *FastTap*. Before touching, the interface have a blank grid with triggering button. When evoked, the menu will be expanded for selection.

The limitation of this design is, it requires the second hand to hold the device for assistance. Always pressing thumbnail is also not quite comfortable.

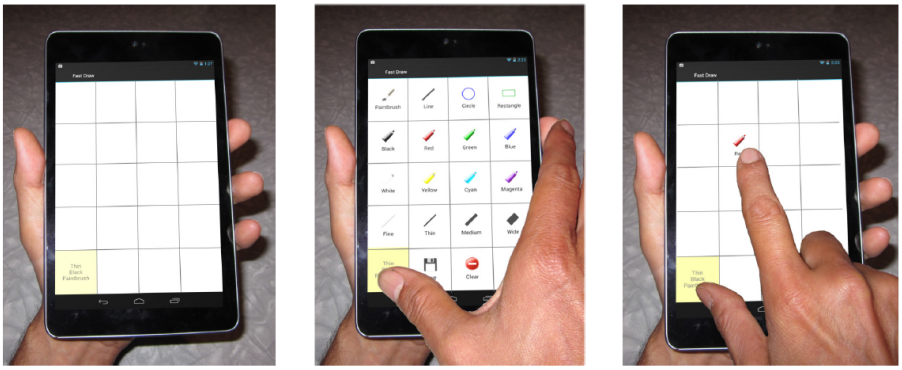


Figure 2.3: FastTap interface.

2.3.2 Finger-Count and Radial Stroke-Shortcuts

The highlight point of Finger-Count and Radial Stroke-Shortcuts [3] is the cooperation of two hands. As shown in figure 2.4, the left (non-dominant) hand is always used for first layer menu selection, while the right (dominant) hand selects an item within the corresponding menu. Finger-Count method controls by the gesture (finger swipe direction). With this interaction method, users could make very quick respond when they get familiar with the menu. Instead of pointing at a certain area for selection, users use a specific gesture at any touchable area instead.

However, this layout only fits menus with two layers, and items in each layer cannot be over 5. Same as FastTap, it also cannot work under single-hand mode.

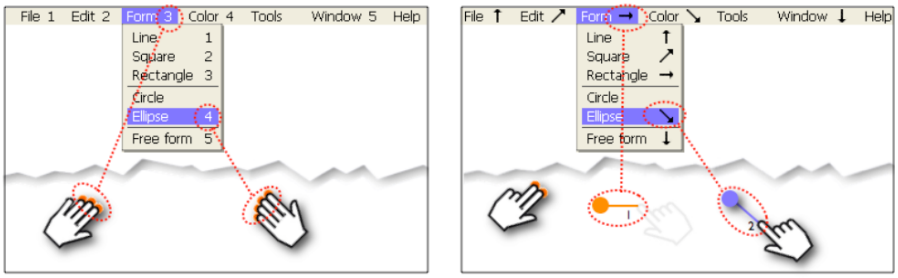


Figure 2.4: Finger-Count and Radial Stroke-Shortcuts.

Chapter 3

Design Foundation

This chapter introduces the most important elements that are considered as the foundation of design.

Section **3.1 Goals** will analysis key aspects for optimizing mobile multitasking behavior. Chapter **4 Design Process** and chapter **5 Modeling Based Design Decisions** will describe how to reach these design goals in details.

Section **3.2 Design Space** will list essential functions of the launcher, and arrange a full design space based on these functions. Design space provides a qualitative criteria for future work.

3.1 Goals

The goal of this thesis project is to design a gesture-controlled launcher to assist mobile multitasking. Mobile multitasking behavior can be optimized in the following aspects:

- **Fast adaption**
  Different usage frequencies of mobile applications not only depend on users' preference, but also affected by contextual factors. If the applications which will be used next can be precisely predicted and easily accessible, the operation time of reaching an application can be shortened.

- **Easy to find target application**
  Current application launchers are all "static", only arrange applications icon based on certain role, such as user definition, alphabet order or application usage, and always take long time when searching for an unfamiliar application. There are also some researches about algorithm

prediction of next applications in use, but that leads to the frequently updating of applications order, and thus makes visual search difficult. If an application list can adjusts to users' real-time need while ensures partially stabilized, then finding application should be much easier.

- **Simplified Interaction**
  Most launchers need at least two steps for opening an application: 1) click to open application list; 2) slide or scroll list to find the target application (optional); 3) click to open the target application. These steps could be promisingly simplified, so that users could use less steps and shorter time to open an application.

- **Support for single-hand operation**
  The launcher could be easily operated with one hand, because sometimes users prefer or have to use smart phone with one hand. Under single-hand mode, the thumb of the dominant hand is most often used [22]. By arranging all possible operations within thumb's comfortable area, the launcher can be used under single-hand mode as well, which guarantees efficiency and convenience of operation. Besides, it could also be customized for left-handed users.

3.2 Design Space

The launcher design should be minimized, so as to ensure operation efficiency and learnability. In this case, I simplified functions of the launcher in 4 aspects:

- **Enter Menu** and **Exit Menu**. The menu can be called out for application selection. Users are also able to exit the menu if they changed their mind.

- **Select Application**. Users can select a target among candidates provided by the launcher.

- **Full Application List**. If the launcher failed with predicting the next application, users can enter full application list quickly.

- **Go Back to Previous Application**. Since the possibility of going back to the last application is always high, it would be useful to have a short cut for going back to the last application without entering the full menu.

Based on these functions, a design space is arranged with following aspects:

- **Gesture**. More detail will be discussed in section **4.1 Gesture**.
  - **Call out menu**. The gesture of calling out the menu should be easy to learn, fast to operate, not overlapping with existing functions, and can avoid misuse.
  - **Exit menu**. The gesture of exit should be easy to find when the launcher is extended, but should not be considered as a high priority function.
  - **Select item**. Select an application in the launcher should be easy and fast, with low error rate.
- **Visual layout**, which will be discussed in section **4.2 Visual Layout**.
  - **Application arrangement**. The way of arranging applications in the menu, and the order of applications.
  - **Application visual expression**. The visual appearance of applications in the list. For example, with icon or thumbnail; with or without application name.
- **Menu parameters**. More details will be discussed in section **4.3 Modeling** and chapter **5 Modeling Based Design Decisions**.
  - **Number of applications**, which means how many applications will appear in the launcher. Allow more applications appear in the launcher will increase the possibility of finding the target application in the launcher, but meanwhile increases the visual selection time and target pointing time.
  - **Which applications**. The applications appear in the launcher should have high usage possibility, meanwhile the launcher should be as stable as possible.
  - **Order of applications**. The position of applications in the launcher will affect positioning time. Generally, the applications closer to the launcher triggering point are easier to find and reach. This parameter is highly bounded with "which applications". If the launcher shows the algorithm predicted highest possibility applications, then the applications will be ordered by possibility,

while the highest possibility application should be placed as easiest to find and easiest to reach; if the launcher shows the most frequently used applications, then they will be ordered by usage frequency, and so forth.

Figure 3.1 gives an overview of the design space.

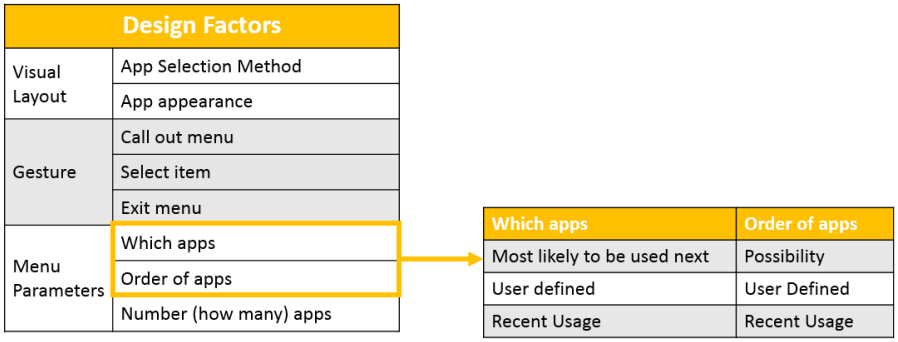


Figure 3.1: Launcher design space

Chapter 4

Design Process

The goal of this chapter is to figure out a unique interaction method, which could make mobile multitasking as fast and easy as possible. The design process is mainly divided into these three aspects: Gesture, Visual Layout and Modeling.

Section **4.1 Gesture** starts with three different initial sketches based on brainstorming, which lead to different interaction gestures and visual layouts. This section analyses the strength and limitation of each design, then selects an optimal one and refined by *design critique* [14] and *cognitive walkthrough* [5] methods.

Section **4.2 Visual Layout** looks into several menu layouts, then picks up a most comfortable and efficient layout based on multiple design factors.

Section **4.3 Modeling** builds up an evaluation model of design parameters of menus and analyses the impact of design parameters on total operation time. Further prediction will be discussed in chapter **6 Experiment**. The design will be verified empirically during chapter **7 Discussion and Conclusion**.

4.1 Gesture

During a brainstorming section, I explored various possibilities of launcher designs which focused on the **visual layout** and **gesture dimensions** of the design space. Afterwards, three typical solutions are selected, as shown in figure 4.1.

Solution 1: Multi-touch gesture

Use three fingers to touch the screen for menu evoking. Then swipe up and down to make item selection. Release finger at an application area to open it.

Vertical lists make item selection easier than horizontal lists, and ensures the application list is not overlapped by user's hand. This design uses multi-touch for menu control for avoiding gesture overlapping with existing Android interactions. But three-finger gesture is not the most comfortable way, and almost impossible to use under single-hand operation.

Solution 2. Dragging button control

Similar with iPhone AssistiveTouch button, the triggering button will always hovers on the screen. Press down the button to call out the application list. Drag this button to make item selection. Release at the selected application to open it.

Finding a button on the screen and pressing is the easiest solution. However, according to previous users' feedback about iPhone AssistiveTouch button, having a button always visible on top of the screen and covering some screen area is quite annoying. What is more, it is also a common case that users accidently touched the button and triggered something unexpected.

Solution 3. Swipe to choose

Swipe form left bottom corner to trigger the launcher, and release finger at an application to open it.

Swiping from left bottom to screen center is a quite natural gesture for right-handed users (for the left-handed ones, it's also easy to provide another vertical mirrored version). Another advantage of this design is, all applications are reachable by user's thumb, which means it could be handled easily under single-hand operation.

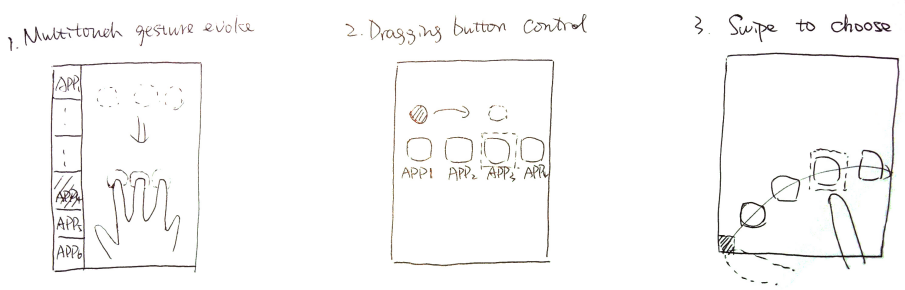


Figure 4.1: Sketches of launcher design solutions

I finally picked Solution 3. Swipe to choose, because it's easy to use, supports single-hand operation, and avoids distracting of general smart phone usage. A more detailed interaction mock-up is designed as shown in figure 4.2, with its detailed function list:

- By swiping from the left bottom corner to the screen center, the launcher will be called out with a list of applications, which is predicted by our algorithm. Each application is shown as an icon and application name. By releasing finger on the selected item, the application will be opened, meanwhile the launcher is folded.
- If users did not find the target application in the launcher, they can open full application list by releasing finger at "application list" button.
- If users triggered the launcher by accident, just release at left bottom corner to fold it.
- By double-tap the left bottom corner, users can go back to the previous application.
- The left bottom "sensitive area" is transparent, which means it won't overlap with any original screen space. The red rectangle in figure 4.2 is just used to announce the size and position of the area.

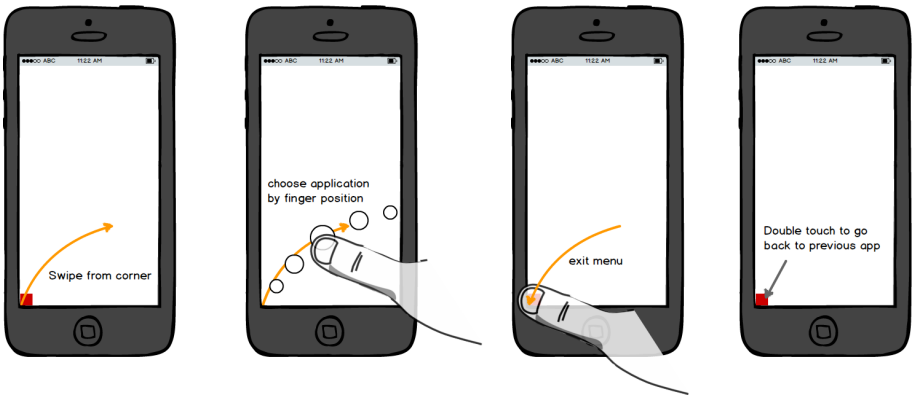


Figure 4.2: prototype wireframe for "Swipe to Choose".

For better indication of SwipeLauncher's interaction process, figure 4.3 shows a *state machine* [20] of all possible statuses of SwipeLauncher, and the interaction for switching between these statuses.

A state machine stores the status of a system, operates on input to change the status and causes actions or outputs.

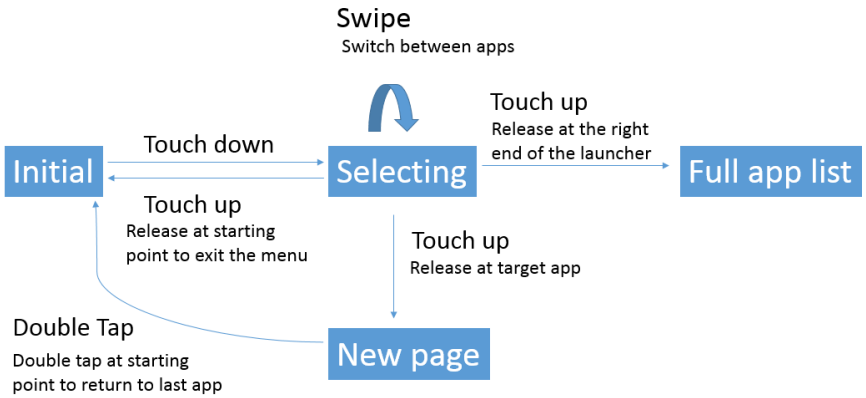


Figure 4.3: State Machine for "Swipe to Choose" prototype.

4.2 Visual Layout

Four different types are listed for the evaluation of different application selection methods, which are Horizontal, Vertical, Diagonal and Angle, as show in Figure 4.4.

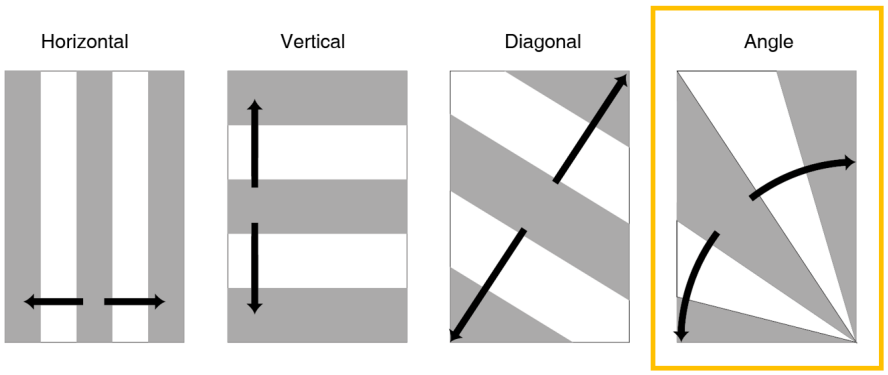


Figure 4.4: Different menu layouts of application selection methods

Vertical and diagonal solutions have wider selection areas for each application, which makes item selection easier. However, these solutions also bring the need of reaching a target on upper screen area, which is difficult under single-hand operation mode, especially for big screen smart phones. According to Wroblewski.L's research [22] on touch device comfort, the most

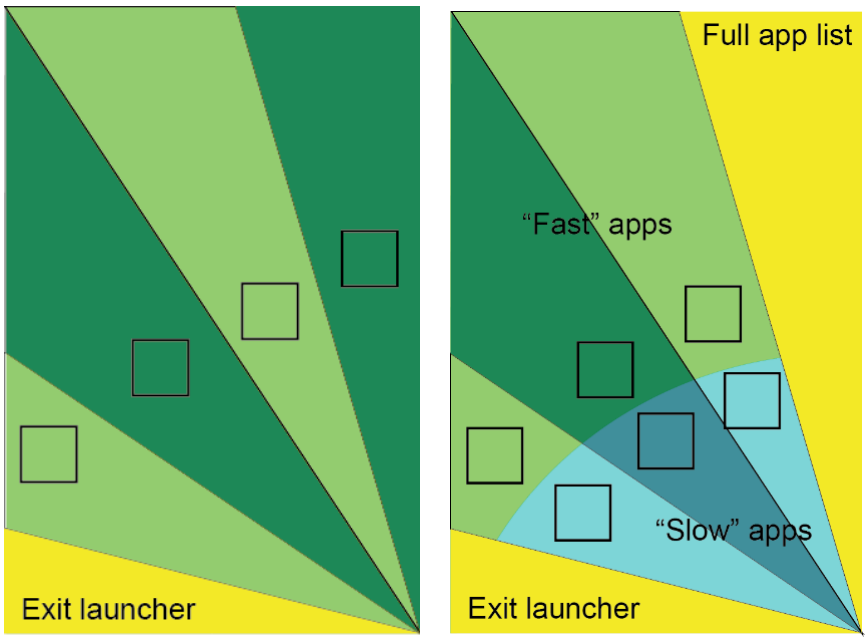
comfortable area for mobile phone is the left bottom area, as shown in figure 4.5. In this case, horizontal and angle design have the most comfortable selection area, as they have most selection space within the "easy" area, thus ensures all possible operations within thumbnail's available area under single-hand mode. What's more, angle design also provides bigger effective selection area for each application. The curved trajectory fits users' thumb movement trace. For these reasons, dividing selection area by angle is the most natural, ergonomic and efficient choice. Therefore, solution 3, "Swipe to choose", is selected as the final design for its easy assessment and natural interaction gesture.

Figure 4.5: Thumb functional area while using smart phone (Wroblewski 2012)

At this point, two possible menu layouts are also discussed within Swipe to choose. The two menu layouts are shown in figure 4.6. According to the design space, the difference between the two layouts are mainly about menu parameters: which applications, how many applications, order of applications. In particular, the first menu layout provides only one row of applications predicted by algorithm, the second menu layout provides two rows of applications, the first row is user's most frequently used applications, second row is algorithm predicted applications.

Menu layout 1

Release at green areas to open corresponding applications; release at yellow area at the bottom to exit the launcher, in case the target application is not in the list.



(a) (b)

Figure 4.6: Two possible menu layout design. (a) Menu layout 1. (b) Menu layout 2.

Menu layout 2

Release at green or blue areas to open corresponding applications. Here, the "fast" applications refer to algorithm predicted ones, which are highly related with user's recent behavior, and keep updating according to users' operation; the "slow" applications refers to most frequently used applications in general, which always stays the same and reflects users' long-term preference.

Section **4.3 Modeling** will build a model for menu parameters evaluation, including which and how many applications in each layout, in order to give a best combination of menu parameters and further select a best launcher design. The most important evaluation indicator is operation time for selecting target application.

4.3 Modeling

This section mainly looks into menu parameters: which applications and how many applications in the launcher.

4.3.1 Which Applications

Section **4.2 Visual Layout** already described two menu layouts under "Swipe to Choose", which are different in number of applications and which applications.

The two menu layouts both used algorithm prediction of "most possible applications to be used". In my design, I use *APPM* algorithm [15] for prediction.

APPM algorithm was implemented by Parate et al. in 2013, which predicts the next-likely applications to be used and the likelihood of an event to occur within some time interval. APPM uses the previous used applications to predict the following ones. Since the prediction is highly related on previous user data, the prediction is highly personalized for each user, and could adjust to user's current behavior quickly. Parate et al. used APPM algorithm for application usage prediction, and reached over 80% accuracy when predicting the top 5 ranking for the next application to be used. Afterwards, they also implemented an adaptive shortcut menu, PREPP, with top applications predicted by APPM algorithm. (More details are in Chapter 2.1, Algorithm Prediction)

However, if the applications in the launcher are only based on algorithm prediction, the applications will probably update frequently, thus user needs to adjust to a new launcher every time, which is not always comfortable. Shin et al. [19] also mentioned in his work, that dynamically changing home screen decreases user satisfaction, and users prefer certain icons to be static. In this way, I use a combination of "fast applications" and "slow applications". "Fast applications" are based on algorithm prediction of users' current behavior, and these applications keeps changing frequently; "Slow applications" refers to the applications with most usage over a long-term period, which makes the launcher more static. The combination of "fast applications" and "slow applications" not only ensures the accuracy of finding the target application in the launcher, but also makes the launcher more static.

4.3.2 How Many Applications

With SwipeLauncher, user could have a general impression of which applications will appear in the launcher before opening, but will not be able to predict exactly which application will appear where. So, users need to spend time searching for an application, and will face the risk of failing to find the application in the launcher, which leads to the two operation situations to be discussed:

1. **Find target application in the launcher**. Users opened launcher,

reach the target application and open it via launcher.

2. **Switch to application list**. Users opened launcher but did not find the target application, then they open application list via launcher. Since the launcher provides a shortcut to application list, users do not have to make any additional switch between screens for application list. Launcher provides a fast access into application list as well.

It's clear that opening an application within the launcher takes shorter time than switching to application list. However, with the increasing number of applications in the launcher, the possibility of finding the target application in the launcher increases, as well as the visual search time and pointing time. Figure 4.7 shows the accuracy and operation time relationship within the launcher, with different application number N in the launcher. The goal is to find out the best number of applications N for reaching a balance of launcher accuracy and selection time, so as to get the shortest expected operation time.

Chapter **5 Modeling Based Design Decisions** will describe the effect of N changing on accuracy, visual searching time and pointing time of an application.

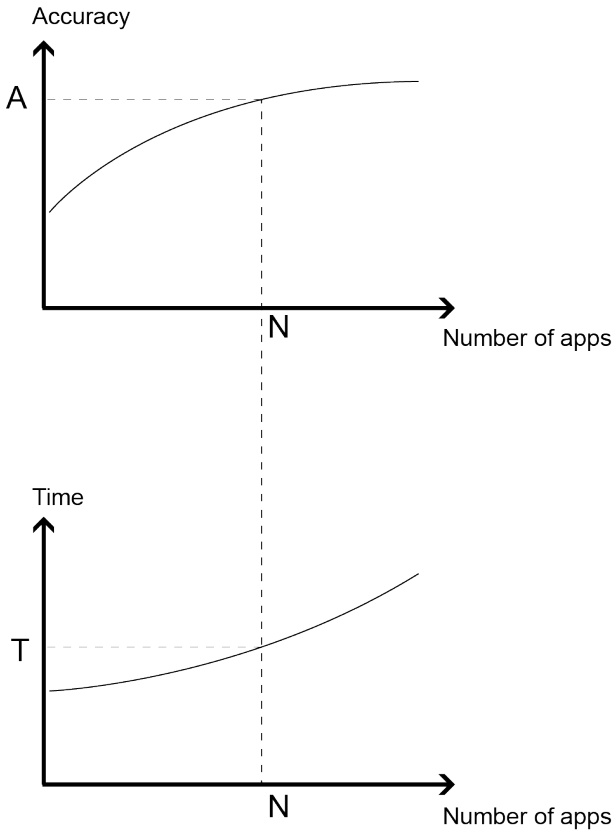


Figure 4.7: Relationship between launcher accuracy, selection time and number of applications N in the launcher.

Chapter 5

Modeling Based Design Decisions

This chapter focuses on layout decisions of SwipeLauncher. I will find out the best user interface design and compare SwipeLauncher with traditional Android home screen.

In section **5.1 SwipeLauncher Operation Time**, I will discuss the two menu layout designs came up in section **4.2 Visual Layout**, decided the best menu layout, as well as the best number N of applications within each menu layout. The decision is based on the operation time modeling, which includes KLM modeling, visual searching time and pointing time prediction.

In section **5.2 Android Original Solutions Operation Time**, I will discuss the typical solutions for launching applications provided by Android system, and pick up the most typical one as the baseline.

In section **5.3 Comparison**, I will compare the efficiency of SwipeLauncher with Android home screen to prove the convenience of SwipeLauncher.

5.1 SwipeLauncher Operation Time

5.1.1 KLM Analysis

Keystroke-Level Model (KLM) [8] is a method for predicting task execution time from a specific task scenario. KLM method is used to evaluate interaction process complexity of a design. The sequence of keystroke-level actions is listed for analyze the operational time and complexity.

Main keystroke steps are as following:

- K - Keystroke pressing a key or button on the keyboard;
- P - Pointing with mouse to a target on the display;
- M - Mental act of routine thinking or perception;

- W - Waiting for the system to respond.

On mobile devices, Holleis et al. [21] adapted a set of operators according mobile phone operation characteristic. Here are some that I used within modeling scope:

- K - The time needed to tap a button by finger;
- P - The time needed to move a finger from one position to another position.

For operation within SwipeLauncher, KLM steps are as following:

1. **Find it in the launcher**
   $T_1 : 2(M + P) + K$
   The first step is to trigger the launcher, which takes M (mental reflection of the operation gesture) + P (point at triggering area to open launcher); the second step is to open target application in the launcher, which takes M (visual search for target application) + P (point at target application location) + K (release finger to open the application)
2. **Did not find it in the launcher, and turn to application list**
   $T_2 : 2(M + P) + K + T_{AppList}$
   The first step is to trigger the launcher, which takes M (mental reflection of the operation gesture) + P (point at triggering area to open launcher); the second step is to open application list in the launcher, which takes M (visual search in launcher) + P (point at application list button) + K (release finger to open application list)

Opening target application from launcher takes less KLM steps and operation time than application list. In subsection **5.1.2 Operation Time Analysis**, I will build a more detailed model for evaluation of each KLM step's operation time. After working out the operation time of the launcher and application list with different number of applications N, the best number of N comes out with the best balance between accuracy and operation time.

5.1.2 Operation Time Analysis

The expectation of operation time T_N can be calculated with the following formula:

$$T_N = A(T_{P1} + T_{V1}) + (1 - A)(T_{V2} + T_{P21} + T_{P22})$$

- A: Algorithm prediction accuracy
- T_{P1}: Pointing time for an application in the launcher
- T_{V1}: Visual search time for an application in the launcher
- T_{V2}: Visual search time for an application not in the launcher
- T_{P21}: Pointing time for open application list in the launcher
- T_{P22}: Time for open an application from application list

Here, $T_{P1} + T_{V1}$ refers to situation 1 "Find target application in the launcher". The possibility of situation 1 depends on algorithm prediction accuracy A. $T_{V2} + T_{P21} + T_{P22}$ refers to situation 2 "Switch to application list". The possibility of situation 2 is $(1 - A)$.

The goal is to find the best N which results in minimum T_N and best performance.

In menu layout 2, T_{P22} is different from traditional solution of opening an application from application list. Based on KLM analysis, tradition solutions usually take 3 steps: 1) find and touch "home screen" button $(M + P + K)$; 2) find and touch application list icon on home screen $(M + P + K)$; 3) find and touch target application icon on application list $(M + P + K)$. Design 2 skipped the first and second step, replaced with a gesture of evoking SwipeLauncher. So in design 2, T_{P22} refers only the time of finding and touching target application icon on application list $(M + P + K)$.

Visual Search Time

According to Brumby et al.'s research [7] on visual search behavior, visual search time mainly depends on two factors: **distance between targets** and **color of target application**.

Distance between targets. Vertical separation between items affects distance between item visual visits. Figure 5.1 shows the duration and distance of item visits by condition. The bigger the gap is, the shorter visual search time users will spend on an item and the fewer items can be viewed as a group.

For the manipulation of menu layout, vertical separation between items was systematically classified into small gap (17 pixels, 0.40°visual angle), medium gap (35.5 pixels, 0.85°visual angle), and large gap (55 pixels, 1.3°visual angle).

Since the task is to open a known application, visual search within SwipeLauncher should be considered as known-item search. According to figure 5.1 (Brumby

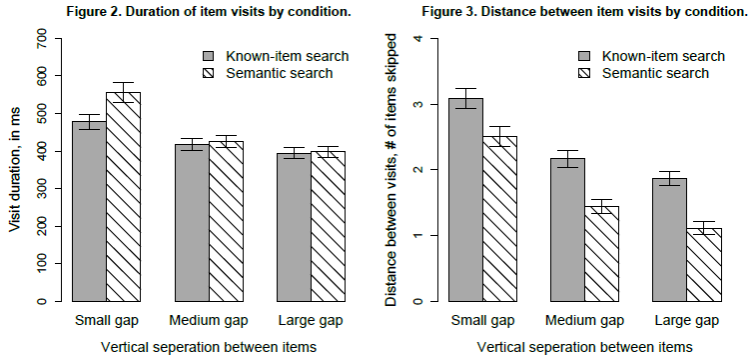


Figure 5.1: Left: duration of item visits by condition; Right: Distance between item visits by condition. (Brumby 2014)

2014), in menu layout 1 (with only one row of algorithm-predicted applications), when the number of applications in the launcher is from 3 to 6, the distance between application icons is bigger than 55 pixel. It is considered as large gap, and the visual search duration under this situation is 395ms. When there are 7 applications in menu layout 1, the distance between application icons is between 55 and 35.5 pixel, which belongs to medium gap, and leads to visual search duration 415ms. In menu layout 2 (with two rows of applications, the first row is users' most frequently used applications, second row is algorithm predicted applications), when there are 4-8 applications in the launcher, the virtual and horizontal distances between application icons are bigger than 55 pixel, which belongs to large gap, and leads to visual search duration of 395ms.

Color of target application. According to Kieras et al.'s research [13] on visual search influence factors, while searching for a known target, color is the most important distinguishing factor. People would first search for the targets with same color, then look into its shape, content and other details. I simulated the color of alternative applications through #Homescreen's statistical result [4]. #Homescreen is a tool for home screen sharing and application discovering. It collects data of most popular applications of its members, and orders them by usage statistics. The screenshot of most popular applications on #Homescreen is shown in figure 5.2.

Here is an example showing how visual search works: if users are looking for "YouTube", and assume the application candidates are first six applications in figure 5.2. So the gap between items belongs to "large gap", and visual search duration of each item is 395 ms.

The first step is looking for a red target (color of YouTube icon), this step

Figure 5.2: Top applications by #Homescreen users

takes 395ms. Since there is only one red icon in the menu (suppose the top 6 applications of #Homescreen's statistics), so the goal achieved when users' sight is located on the only red application (Youtube). The whole visual search time is 395 ms.

In another situation, if users are looking for Twitter, then in the first step of looking for blue icons, there is 50% possibility that the users' sight is first located on Twitter icon, and another 50% on Facebook icon, because their colors are similar. In the second case of users focusing on Facebook first, then users spend 395ms to distinguish that the icon not the right goal, and jump to the next blue icon. So the expectation of visual search time should be:

$$395 \times 0.5 + (395 + 395) \times 0.5 = 592.5ms$$

If the user is searching for an application not in the list, it takes longer time to look into all possible targets, because the decision is made after viewing each visual "unit".

Visual search time for the two menu layouts is shown in table 5.1, 5.2, 5.3, 5.4, figure 5.3, 5.4, 5.5, 5.6. In general, having more applications in the launcher will increase visual search time. However, visual search time not only depends on the number of applications in the launcher and their distance, but also the icons' color. In this way, it would be easier to search applications with special colors. That is why sometimes having more applications in the launcher does not always increase visual search time.

Table 5.1: Menu layout 1, when the target application is in the launcher

Number	Same color percentage	Gap	Visual search duration(ms)	T_{V1}(ms)
3	0	Large	395	395
4	50	Large	395	493
5	40	Large	395	474
6	50	Large	395	592
7	71	Medium	420	660

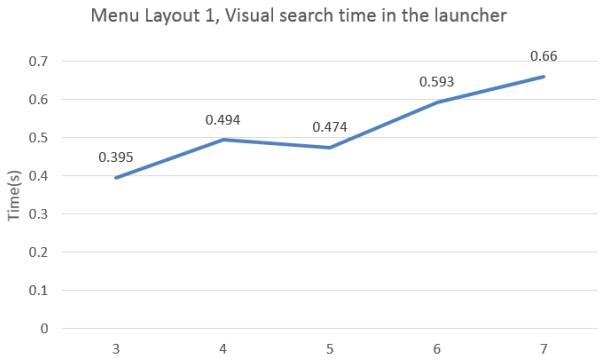


Figure 5.3: Menu layout 1, visual search time when the target application is in the launcher

Table 5.2: Menu layout 1, when the target application is not in the launcher

Number	Gap	Visual search duration(ms)	T_{V1}(ms)
3	Large	395	395
4	Large	395	790
5	Large	395	790
6	Large	395	1185
7	Medium	420	1260

Table 5.3: Menu layout 2, when the target application is in the launcher

Number	Same color percentage	Gap	Visual search duration(ms)	T_{V1}(ms)
2*2	0	Large	395	395
3*2	50	Large	395	493
4*2	40	Large	395	474

Pointing Time

For the calculation of pointing time, I built some prototypes using MIT APP Inventor [1], an interactive prototyping tool. Each prototype was built up with gesture-evoked control. The only difference from a real launcher is that,

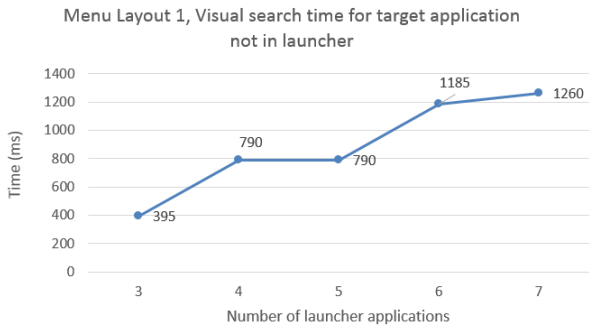


Figure 5.4: Menu layout 1, visual search time when the target application is not in the launcher

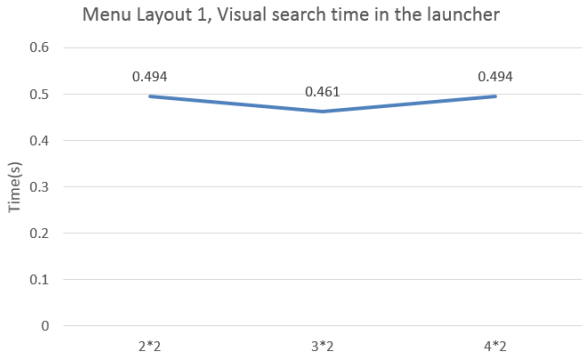


Figure 5.5: Menu layout 2, visual search time when the target application is in the launcher

Table 5.4: Menu layout 2, when the target application is not in the launcher

Number	Gap	Visual search duration(ms)	T_{V1}(ms)
2*2	Large	395	790
3*2	Large	395	1185
4*2	Large	395	1580

when pressed icons, it only shows a picture of the corresponding application, not open real application. I make this simplification because at this moment the goal is only to evaluate operation time for reaching an application.

Reasons of using prototyping method instead of Fitts Law for measurement of pointing time are: 1) Fitts Law model is used for pressing a target in a distance, while SwipeLauncher uses swiping instead of clicking. 2) The

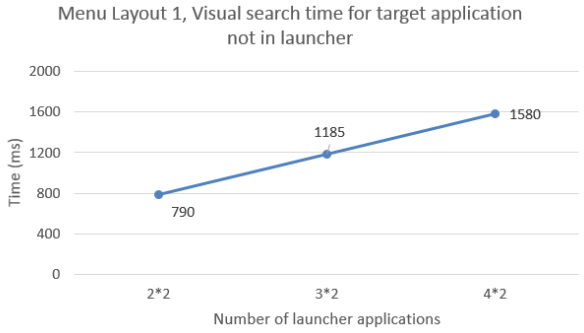


Figure 5.6: Menu layout 2, visual search time when the target application is not in the launcher

operation time for touching targets at different location on the screen is different, not only because of the distance from triggering point, but also affects by the gesture easiness.

With menu layout 1, the relationship of pointing time changing with number of applications N is shown in table 5.5.

Table 5.5: Pointing time for different N (number of applications) with SwipeLauncher menu layout 1

N	T_{P1}(s)
3	1.23
4	1.35
5	1.35
6	1.38
7	1.45

With menu layout 2, the relation of pointing time changing with number of applications N is shown in table 5.6.

Table 5.6: Pointing time for different N (number of applications) with SwipeLauncher menu layout 2

N	T_{P1}(s)
2*2	1.2
3*2	1.2
4*2	1.3

5.1.3 Accuracy

With menu layout 1, the accuracy (of finding the target application in the launcher) depends on algorithm prediction accuracy; with menu layout 2, the accuracy (of finding the target application in the launcher) depends on both algorithm prediction accuracy and most frequent used applications' usage percentage.

Algorithm prediction accuracy is based on APPM algorithm [15] prediction within LiveLab [18] database. APPM is an algorithm implemented by Parate et al., which predicts next-likely events and likelihood of an event to occur within some time interval. More detailed information about APPM is mentioned in section **2.1 Algorithm Prediction**.

LiveLab is a database that contains traces of 34 volunteers' application usage data on iPhone 3GS for a period of up to 14 months. The data used in my experiment includes app-usage traces and appusage.sql, which provide the start time and the duration of each application usage.

While using APPM algorithm to predict the application usage in LiveLab database, the accuracy data is shown in table 5.7.

Table 5.7: The APPM prediction accuracy changing with predicting number of applications

N (number of applications)	Accuracy
2	0.7007
3	0.7759
4	0.8302
5	0.8658
6	0.8873
7	0.9042

5.1.4 Expectation of Operation Time

Based on previous formula $T_N = A(T_{P1} + T_{V1}) + (1 - A)(T_{P1} + T_{V2} + T_{P2})$, time for opening target application with menu layout 1 and menu layout 2 under different numbers of applications are listed in table 5.8 and 5.9:

In menu layout 1, the best design comes when N = 4, and expectated task completion time is 2.207s.

In menu layout 2, the best design comes when N = 3*2 (3 "fast applications" and 3 "slow applications", expectated task completion time is 1.968s.

So, the best SwipeLauncher design is menu layout 2, with 3*2 applications in the launcher. Launcher layout is shown in figure 5.9.

Table 5.8: Operation time result for SwipeLauncher menu layout 1

N	A	T_{V1}(s)	T_{P1}(s)	T_{V2}(s)	T_{P2}(s)	T_N(s)
3	0.7759	0.395	1.23	0.395	3.6	2.432
4	0.8302	0.494	1.35	0.790	3.6	2.207
5	0.8658	0.474	1.35	0.790	3.6	2.350
6	0.8873	0.593	1.38	0.185	3.6	2.445
7	0.9042	0.660	1.45	0.260	3.6	2.362

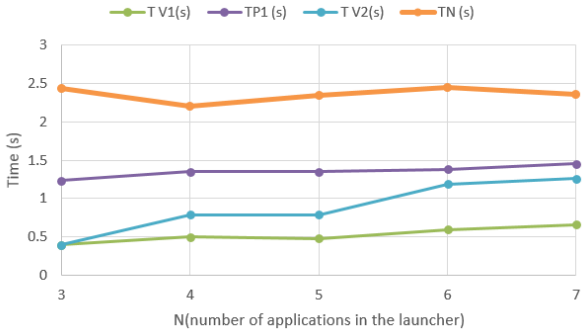


Figure 5.7: Pointing time for different N (number of applications) with SwipeLauncher menu layout 1.

Table 5.9: Operation time result for SwipeLauncher menu layout 2

N	A	T_{V1}(s)	T_{P1}(s)	T_{V2}(s)	T_{P2}(s)	T_N(s)
2*2	0.8302	0.494	1.2	0.79	2	2.084
3*2	0.8873	0.461	1.2	1.185	2	1.968
4*2	0.916	0.494	1.3	1.58	2	2.053

This "best layout design" does not lead to fastest operation speed, but the balance between operation speed and accuracy. It does not only consider the operation time within the launcher, but also considers the situation of switching to application list.

5.2 Android Original Solutions Operation Time

This section analyzes existing solutions provided by Android system, then selects the best combination among them. Afterwards, I will compare the best SwipeLauncher design with best Android solution's operation time.

I define the operation time through Android's solution as: start from a

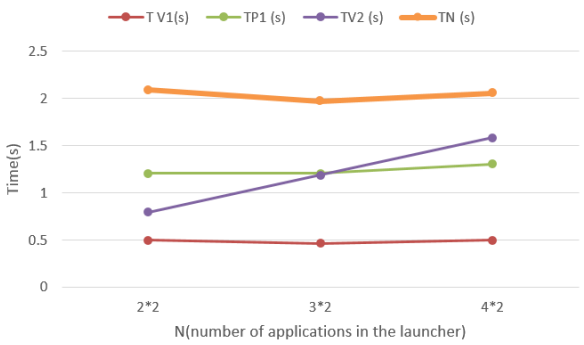


Figure 5.8: Operation time result for SwipeLauncher menu layout 2.

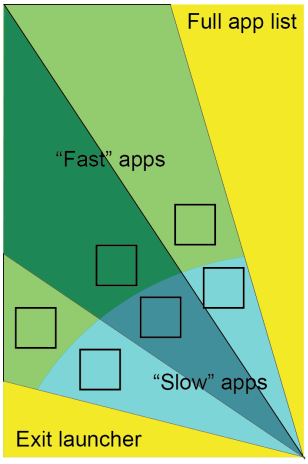


Figure 5.9: SwipeLauncher final design

random application interface and trigger an operation, to click the target application icon and open it. For example, in the "home screen" case, timing starts when the home screen button is pressed, ends when the target application is opened.

Figure 5.10 shows all possible launching methods provided by Android 5.0 system. Different Android operation systems are slightly different, and many manufacturers customized their own Android system. I am not going to discuss the difference between Android systems in this thesis. Here, I limited the range within the existing three solutions on Nexus 5, system Android 5.0 "Lollipop".

1. **Home Screen Shortcut.** Users can customize application shortcuts'

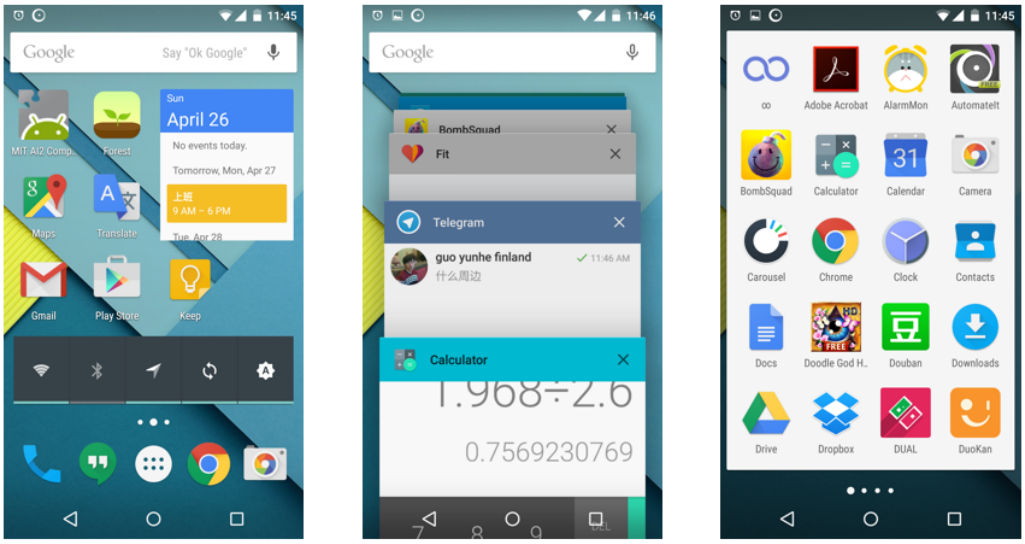


Figure 5.10: Android original solutions for launch applications

existence and position on home screen. The four applications and the shortcut of application list in the bottom appears on all home screen pages (if there are more than one), other applications only appear on a single home screen page. Here are two possible situations for opening an application on home screen:

1.1 **Find it directly on the home screen page.** Users press the "home" button to open home screen page, and press an application icon to open it.

1.2 **Switch to application list.** This is an optional situation. If users viewed all home screen pages but still did not find the target application, users can press the "application list" button to view all applications.

When users are familiar with users' own home screen applications, users can switch quickly to application list if he is sure the target application is not on home screen. This could help to reduce operation time for experienced users.

2. **Recent Task List.** Recent used applications are listed by recent usage in this list. Users can scroll up and down to view the applications in a flow menu. This solution is efficient for the most recent 2-3 applications, but the efficiency drops quickly while looking for more previous

applications, since users do not always remember which applications are used in which order some hours ago.

2.1 **Find it directly on the recent task list screen.** If an application is recently used, it will appear directly on the screen. The more recent it is, the larger space it takes.

2.2 **Scroll down and find it.**This is an optional situation. Users scroll down to view more previous application.

2.3 **Switch to application list.** This is an optional situation. If users did not find it in the recent task list, or feel tired of scroll down so many times, users should first go back to the home screen, then press "application list" button to view application list. This takes longer time than situation 1.2 Switch to application list on home screen, because in this case, users will not know the applications in recent applications list before opening, so should view all applications in this list to decide the target application is not here. What's more, because of the instability of recent task list, there is quite some chance of switch to application list under this solution.

3. **Application List.** This is the safest option, since all applications installed on the mobile will appear in the list. However, it is always slower to find an application here, because icons are arranged with high density. Since the application list is always static, users could roughly remember the location of frequently used applications, but not all applications.

   3.1 **Find it directly on the first page.**If the application appears on the first page on application list.

   3.2 **Switch to another page.**If the application is not on the first page, then users need to switch between pages for searching.

5.2.1 KLM Analysis

I also analyzed the KLM steps of launching an application for all the situations mentioned above:

1. **Home Screen Shortcut**

   $T_{1.1} = 2(M + P + K)$

   $T_{1.2} = M + P + K + xS + T_{AppList}$

2. **Recent Task List**

   $T_{2.1} = 2(M + P + K)$

   $T_{2.2} = 2(M + P + K) + xS$

   $T_{2.3} = M + P + K + xS + T_{AppList}$

3. **Application List**

   $T_{2.1} = 3(M + P + K)$

   $T_{2.2} = 3(M + P + K) + xS$

In the KLM steps of **3 App list**, S refers to a swipe and x is the number of screen swipe.

5.2.2 Android Original Solution Time Measurement

I also used prototyping method for measure the operation time of each situation, the operation time for all Android solutions are as following:

1. **Home Screen Shortcut**

   $T_{1.1} = 2.2s$

   $T_{1.2} = 2.2s + T_{AppList}$

2. **Recent Task List**

   $T_{2.1} = 3s$

   $T_{2.2} = 7s$

   $T_{2.3} = 7s + T_{AppList}$

3. **Application List**

   $T_{2.1} = 2.6s$

   $T_{2.2} = 2.6s + x$

   (Time for swipe a page $S = 1s$, x refers to the application location page, range from 0, 1, 2, 3 . . .)

5.2.3 Accuracy

For Home Screen, "accuracy" refers to the possibility of finding the target application on home screen. In order to get the best efficiency for home screen, I consider the most frequently used applications are put on home screen. According to comScore's U.S. Mobile App Report [3] about home screen application numbers, the average number of applications on a home screen is 12. According to Bohmer et al.'s previous work [6], the usage of top 12 applications is around 54%.

5.2.4 Expectation

The expected operation time of Android home screen + application list is 2.6 s. Calculated by:

$T = AT_1 + (1 - A)T_2$

A: Possibility of finding target application on home screen;

T_1: Time of finding target application on home screen;

T_2: Time of finding target application on application list.

Since **2. Recent Task List** shows longer operation time than home screen, also has high uncertainty, I will not put it into the baseline plan.

The combination of Home Screen and Application List is used as a baseline in the following, for comparison with SwipeLauncher.

Home screen can be highly customized by users. If users are familiar with their own home screen and put frequently used applications on home screen, there would be a high possibility of finding target application on home screen, and the operation should be faster. In addition, application list is used as a fallback solution. If users did not find the target application on home screen, they can always find it in application list.

5.3 Comparison

As a conclusion, best design of SwipeLauncher is menu layout 2, with 3*2 applications and "all applications" button in the launcher, expected operation time is 1.968 s. Best design of traditional solution is Home screen, expected operation time is 2.6 s. So, SwipeLauncher is predicted to be 0.632 s faster than traditional solution, increased operation time by 24.3%.

The operation time calculation in this chapter is based on theories. Chapter **6 Experiment** will use an empirical study section to compare the best design of SwipeLauncher and Home Screen.

Chapter 6

Experiment

6.1 Experiment Design

In previous section **5 Modeling Based Design Decisions**, I find a best SwipeLauncher design through prediction of operation time. In this chapter, I will use an experiment for evaluating the application opening process of the best SwipeLauncher design and Android home screen. Android home screen is used as a control group. During the experiment, participants were asked to open applications for 150 times on SwipeLauncher and Android home screen, include the situations of target application in the launcher and application list. Afterwards, participants fill in a questionnaire and did an interview for ranking SwipeLauncher and give further suggestions. The evaluation includes operation speed, error rate, comfort, etc.

6.1.1 Factors

The study was a within-subjects 2*2 factorial design. The factors and levels were:

- **Interface:** SwipeLauncher and Home Screen. SwipeLauncher is the new interaction method I developed. Home Screen is a function of Nexus 5, in which users can customize the shortcuts for certain applications, and order them freely on the screen.
- **Opening solution:** Open application through launcher or application list. Participants need to open the target application provided by experimenter. The target application could appear in launcher randomly.

6.1.2 Participants

12 participants are involved in this experiment, includes 7 females and 5 males, age range 22-28 (mean age 25.6 years old, standard deviation 2.121). All the participants are familiar with Android 4.0 or newer version system, and have Android phones for over 6 months.

All the participants are right-handed and not color-blind.

6.1.3 Software and Apparatus

The experiment is held in an enclosed room, with a camera recording participants' operation, a table and desk for the participants. The experimenter sits beside the participants during the experiment for observation.

The experiment device is Nexus 5, with Android 5.0 System installed. The phone already has 66 applications installed (which is the average number of applications installed on smart phone), with 12 applications on home screen and 6 in SwipeLauncher.

The number of applications in SwipeLauncher is based on analysis in chapter **5 Modeling Based Design Decisions**.

Since I want to make both scenarios as realistic as possible, I not only control the number of applications in each solution, but also which applications. The applications used in the experiment should be generally popular, so that participants could quickly recognize the target application, which helps decrease error rate and operation time. What's more, it also ensures that participants are searching by icon color and shape, instead of looking at application name.

The applications that appear during experiment are selected among the most popular applications according to comScore's U.S. Mobile App Report [9] in Figure 6.1. The statistics data is based on American people mobile phone usage in 2014, age 18-55+.

During the experiment, I fixed the SwipeLauncher "slow applications" to Facebook, Youtube, and Google Play, which are the generally most frequently used mobile applications, also according to comScore's U.S. Mobile App Report. The "fast applications" will always be selected among these 15 applications, which are the next most frequently used applications: Google Maps, Gmail, Instagram, Facebook Messenger, Twitter, the weather channel, Google +, Netflix, Snapchat, Amazon Shopping, Pinterest, Skype, Whatsapp, Evenote, Dropbox. The appearance probability of each application is based on its usage frequency, according to Figure 1, but removed the applications only available on IOS, such as Apple Maps and iTunes. An application

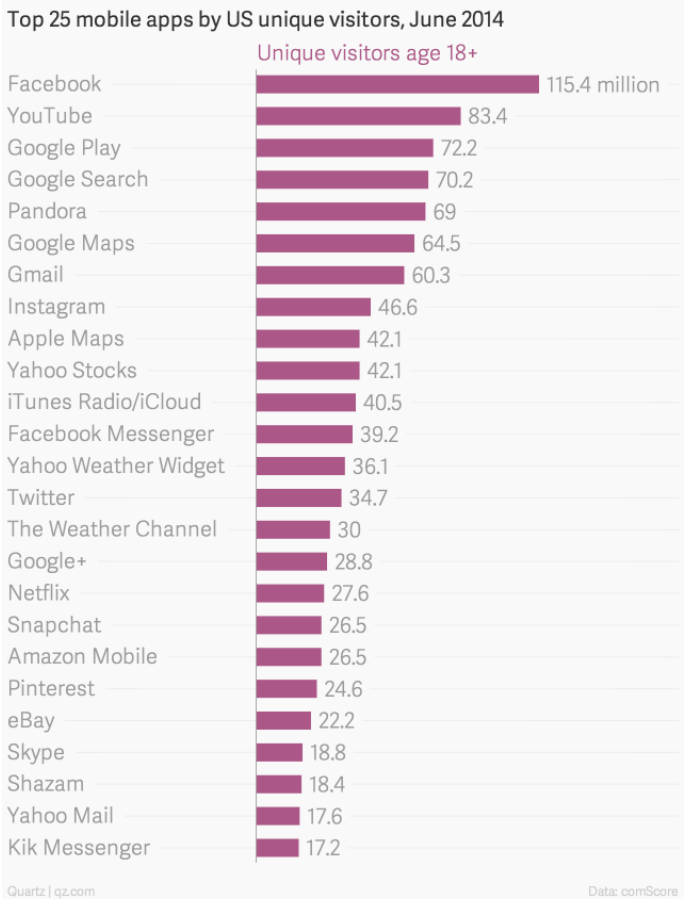


Figure 6.1: Top application usage (U.S. Mobile App Report, 2014)

with higher usage frequency in the real life have higher possibility of appearance in SwipeLauncher.

During the experiment, the applications on home screen are: Instagram, Facebook, Twitter, Google Maps, Facebook Messenger, Youtube, Whatsapp, Snapchat, Dropbox, Evenote, Skype and Gmail. They are the most popular applications on home screen, according to the statistical results on `http://homescreen.is/top-applications`. All the applications are on one home screen, which is also the only home screen on the experiment device. Figure 6.2 shows the final visual layout for home screen and SwipeLauncher used during the experiment.

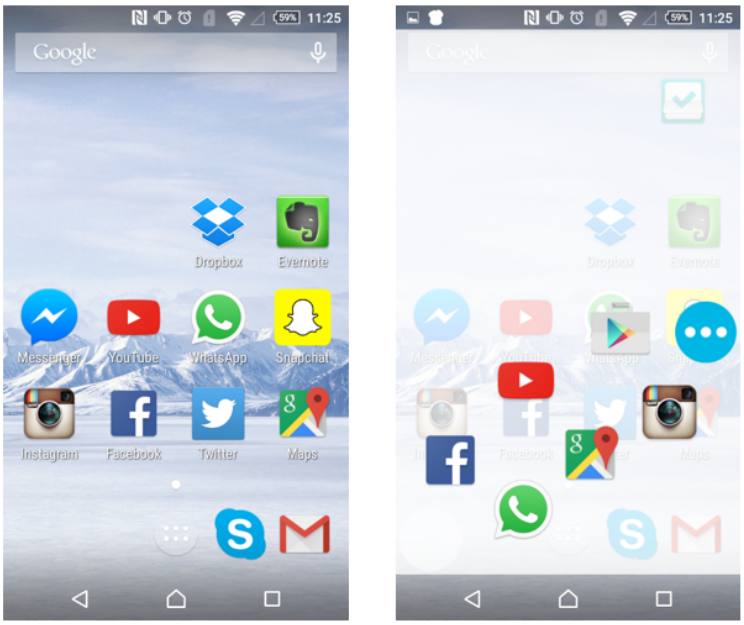


Figure 6.2: Home Screen and SwipeLauncher design

6.1.4 Experiment Procedure

During the experiment, half of the participants are asked to start with SwipeLauncher section first, then home screen; another half participants starts with home screen section, then SwipeLauncher. This design is to avoid the experiment order affect issue, such as long operation induced fatigue, familiarity with experiment device leads to faster operation speed.

Each section includes two conditions: target application is in the launcher; or target application is in application list. These two conditions will occur randomly, and participants could not predict the order in advance.

6.1.4.1 Home Screen

Ask participants to open certain applications through home screen. The workflow of home screen is shown in figure 6.3. Participants can switch quickly to app list if they are sure the target application is not on home screen.

- **Trail task section:** participants complete the task for 60 times in order to get familiar with the system.

- **Real task section:** participants complete the task for 90 times, task complete time will be recorded. 50% target app on home screen and 50% not.

In both situations, there is 50% possibility of finding the target app on home screen and 50% not. These two situations happens randomly and participants will not know the order in advance.

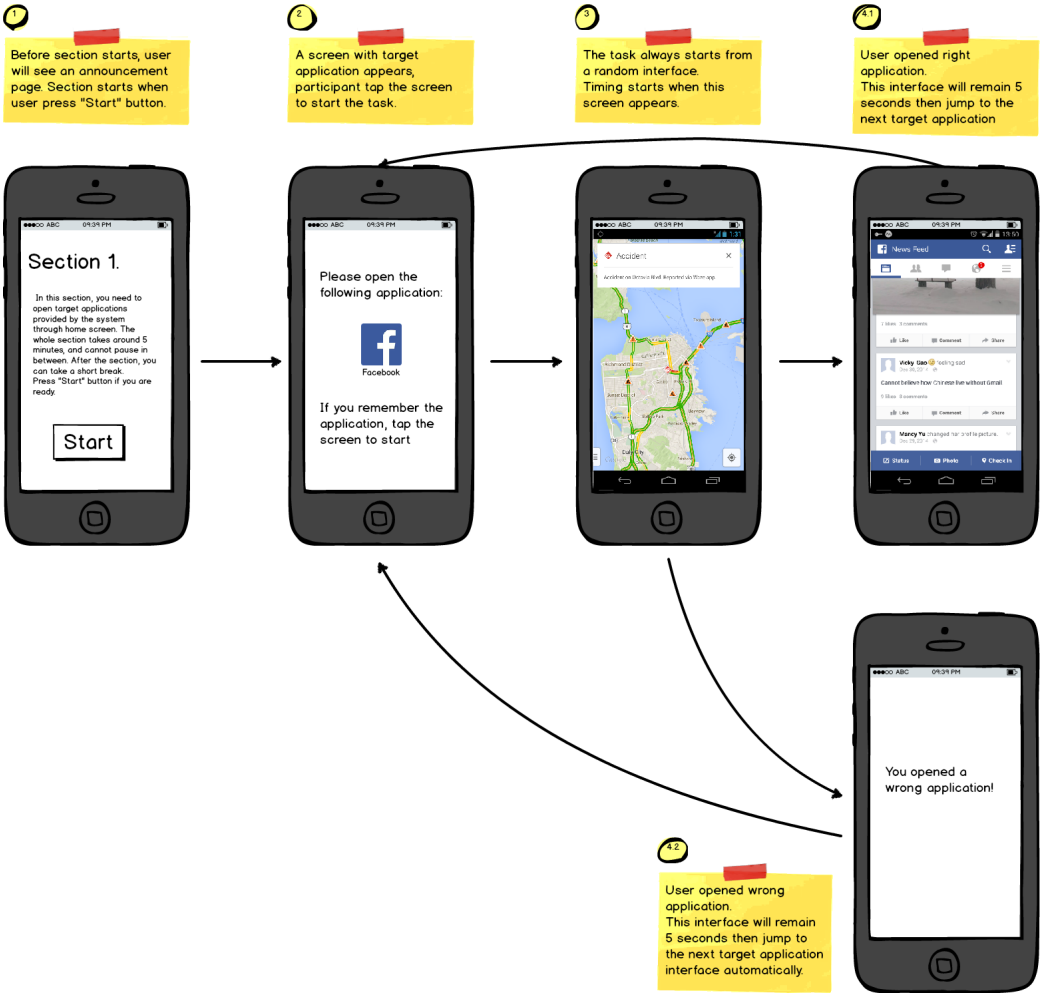


Figure 6.3: Workflow of home screen section

6.1.4.2 SwipeLauncher

Ask participants to open certain applications through SwipeLauncher. The workflow of SwipeLauncher is shown in figure 6.4. Participants should always

open SwipeLauncher to see if the target application is in the launcher. If not, participants open application list (must through SwipeLauncher) and select.

- **Trail task section:** participants complete the task for 60 times in order to get familiar with the system.
- **Real task section:** participants complete the task for 90 times, task complete time will be recorded.

In both situations, there is 66% percentage of finding the app directly in the launcher (33.3% from "slow applications" and 33.3% from "fast applications"), and 33.3% of turn to app list. These two situations happens randomly and participants will not know the order in advance.

After the experiment, participants are asked to fill in an assessment questionnaire (see Appendix 3: Assessment Questionnaire), and a short interview.

6.2 Experiment Result

6.2.1 Analysis

The experiment result includes the data recorded by device during the experiment, such as operation time, error rate, etc.; and participants' feedback through questionnaire and interview after the experiment, such as comfort, system complexity, etc.

The device recorded data include the following aspects:

- **Operation time:** The operation time starts when the "start" button on the instruction interface is pressed; ends when an application is opened, no matter if it is the target application or not. For SwipeLauncher, I recorded the operation time for "fast applications", "slow applications" and application list; for Android home screen, I recorded the operation time for the home screen launcher and application list.

  Afterwards, I calculated the overall operation time based on the operation time in each situation and the prediction accuracy (calculated in Chapter 5. Modeling Based Design Decisions).
- **Error Rate:** participants opened a wrong application.
- **Ignore Rate:** when the target application is in the launcher, but participants ignored it and open it in application list.

The participants' feedback questionnaire includes these aspects:

Figure 6.4: Workflow of SwipeLauncher section

- **Comfort:**The comfort degree of using SwipeLauncher to open applications, compare with Android home screen.
- **Speed:** Participants' feeling about SwipeLauncher's operation speed, compare with Android home screen.
- **Complexity:** Whether SwipeLauncher is easy to learn and use, compare with Android home screen.

6.2.2 Result

6.2.2.1 Operation Time

While opening an application within launcher (SwipeLauncher or home screen, not considering finding target application in application list), the average time of SwipeLauncher is 1.552 s, home screen 1.927 s. SwipeLauncher is 0.375s, 24.5% faster than home screen.

For SwipeLauncher tasks, I also looked into the performance of "slow applications" and "fast applications". Participants take 1.499s while open target application in "slow applications", and 1.600s with "fast applications". "Slow applications" is 0.101s, 6.67% faster than "fast applications".

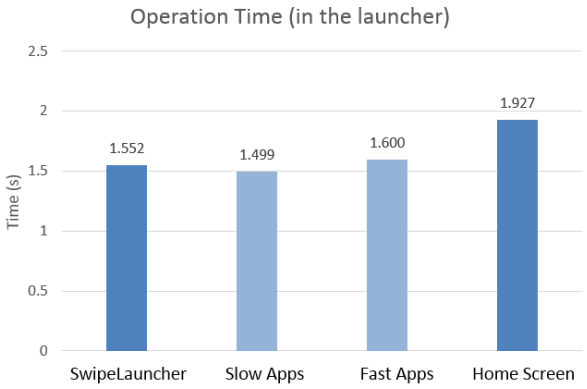


Figure 6.5: Operation time of SwipeLauncher and home screen (in the launcher)

While opening target application in application list, home screen performance is better than SwipeLauncher. On average, home screen takes 3.3s, which is 42% better than SwipeLauncher.

For the overall performance, I used the prediction accuracy of SwipeLauncher and home screen calculated in Chapter 5. Modeling Based Design Decisions. The overall operation time is as following:

SwipeLauncher:

$$T_{SL} = A(T_{P1} + T_{V1}) + (1 - A)(T_{P1} + T_{V1} + T_{P2}) \tag{6.1}$$

A: Accuracy of finding target application in SwipeLauncher: 0.8873

$T_{P1} + T_{V1}$: Time of opening target application in SwipeLauncher: 1.552 s

$T_{P1} + T_{V2} + T_{P2}$: Time of opening target application in application list (through SwipeLauncher): 4.701 s

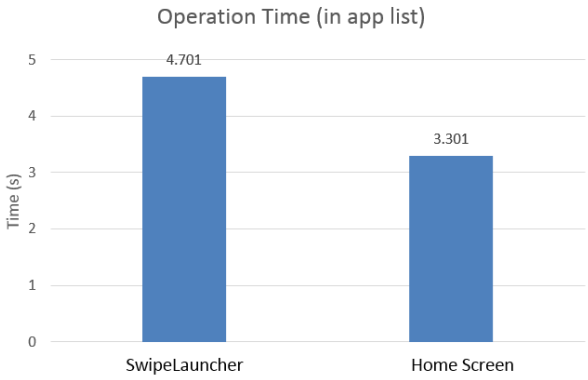


Figure 6.6: Operation time of SwipeLauncher and home screen (in application list)

So $T_{SL} = 1.907s$

Home Screen:

$$T_{HS} = AT_1 + (1 - A)T_2 \tag{6.2}$$

A: Accuracy of finding target application on home screen: 0.54
T_1: Time of finding target application on home screen: 1.927 s
T_2: Time of finding target application on application list: 3.301 s
So $T_{HS} = 2.559s$
As a conclusion, SwipeLauncher is overall 0.652s, 34.2% faster than home screen.

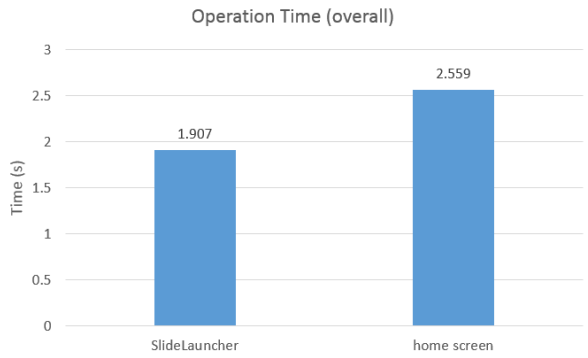


Figure 6.7: Operation time of SwipeLauncher and home screen (overall)

6.2.2.2 Error Rate

Comparing the error rate of SwipeLauncher and home screen, SwipeLauncher have a much lower error rate of 1.569%, while home screen is 3.352%.

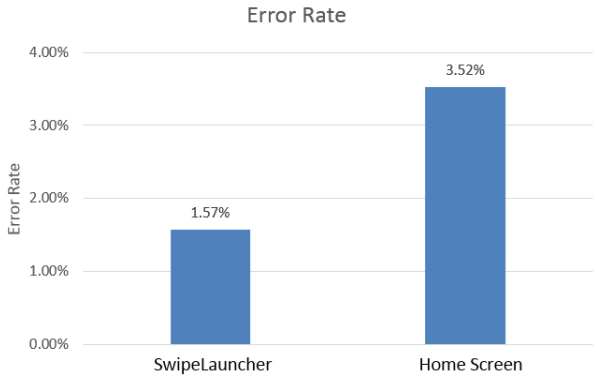


Figure 6.8: Error rate of SwipeLauncher and home screen

6.2.2.3 Ignore Rate

According to the data, SwipeLauncher have a slightly better error rate of 2.309%, while home screen is 2.5%.

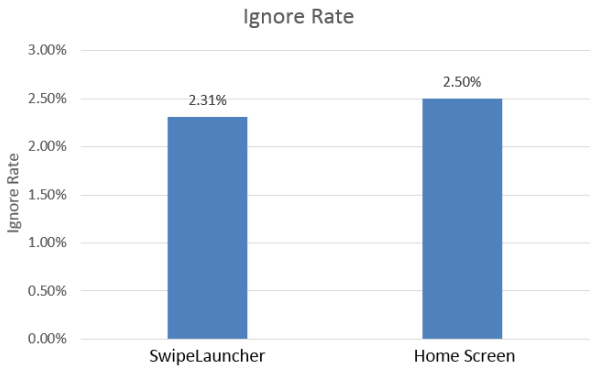


Figure 6.9: Ignore rate of SwipeLauncher and home screen

6.2.2.4 Questionnaire

Each participant is asked to fill in a questionnaire (see appendix 2) after the experiment for evaluation of SwipeLauncher. The questionnaire contains

results of the following dimensions: operation comfort, speed, complexity and overall ranking. Each dimension is evaluated by a score from 1-5, in which 5 is the best and 1 is the worst. The result is shown in figure 6.10.

For operation comfort, most participants think SwipeLauncher is more comfortable than home screen.

For operation speed, most participants think the two solutions are not very different.

For operation complexity, home screen have a better ranking than SwipeLauncher. However, since all the participants are already Android users, which means they are already familiar with Android home screen. So they are tend to feel easier to work with home screen.

For the overall ranking, SwipeLauncher gets a better ranking than home screen.

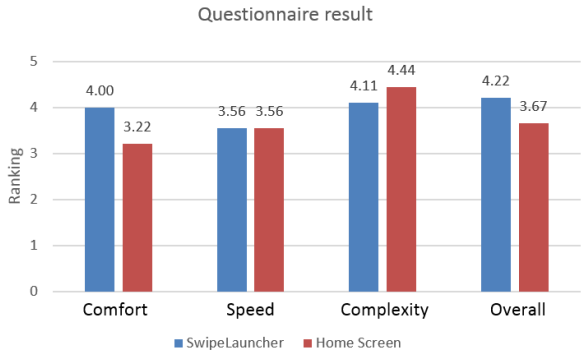


Figure 6.10: Questionnaire result of SwipeLauncher and home screen

6.2.3 Discussion

6.2.3.1 Operation Time

User performance with "slow applications" is slightly better than with "fast applications", which approves making applications more stable will help improving user performance. The reason of this difference is "slow applications" are fixed and "fast applications" keep updating all the time.

What's more, since all "fast applications" are generated randomly during this experiment, users could not predict which applications will appear in "fast applications", so every visual search inside "fast applications" is an unknown list searching. In a real scenario, the "fast applications" are based on user behavior, which means users could have a better understanding about

"fast applications", this may lead to better performance and shorter operation time.

In the application list part, home screen have better performance than SwipeLauncher. I see the influencing factors in two aspects:

- **Visual Search Time:** Since the applications on home screen are fixed, users have a expectation of whether the target application is on home screen or not, thus reduced the visual searching time of deciding "the target application is not on home screen" and continue open the application list.
- **Pointing Time:** The button of opening home screen is very close to the button of open application list on home screen; while in SwipeLauncher, the button of open application list is most far away from the triggering area.

Based on these reasons, although home screen takes one more step for opening application list, it still gets better performance than SwipeLauncher.

6.2.3.2 Error Rate

While designing the experiment, home screen ensures the success rate of finding target application by adding more applications on home screen, while SwipeLauncher solves the problem by keep updating applications in SwipeLauncher.

Since SwipeLauncher gets lower error rate than home screen, which means the strategy of updating applications in the launcher is successful.

6.2.3.3 Ignore Rate

In a real scenario, users should have more training time and understand the system better, so the ignore rate of both SwipeLauncher and home screen should be decreased. However, since home screen applications are fixed, the ignore rate of home screen is supposed to get a bigger decrease.

6.2.3.4 Interview

This is the last step after the experiment and questionnaire, which contributes to a better understanding about SwipeLauncher's user experience.

- **Gesture Control:** Since SwipeLauncher provides a new gesture-evoked method, one of the most important thing is to understand if it is comfortable and fast for users.

Most participants believe SwipeLauncher is more comfortable and easier to open an application, some participants even pointed out that SwipeLauncher takes fewer steps than home screen, and there is no need of exit current application to continue opening another application, while home screen always requires users to go back to home screen first.

Four participants feel it is strenuous to keep pressing the triggering area while looking for target application, they feel it is more difficult to keep pressing rather than press buttons twice.

Two participants think it would be a problem for them under single hand mode, especially with a bigger screen (on their own device). Especially when swiping from triggering area (left bottom corner) to a target application on the right side of the screen, for it has longest swiping distance.

Most participants' performance improved quickly while learned to use SwipeLauncher.

- **Visual Search:** Most participants believe searching in "slow applications" is faster than searching in "fast applications", they gradually remember the "slow applications" and their order in SwipeLauncher during the experiment. This also supports the result of operation timing, in which opening target application within "slow applications" is faster than "fast applications".

  About half participants mentioned unfamiliarity with some target applications and lead to operation error or cannot find target application. But this situation could be prevented in a real scenario, because users are more familiar with the applications they installed on their own devices. So SwipeLauncher and home screen's visual search performance should both get better in a real scenario.

Chapter 7

Discussion and Conclusion

In this thesis project, I looked into the possibility of optimizing multitasking behavior on mobile devices. Due to small screen sizes and few control buttons on mobile devices, handling multitask is difficult. A reasonable solution is to optimize the performance of switching between applications, so I designed SwipeLauncher, a mobile application launcher which supports fast application access.

There are two main unique points about SwipeLauncher: 1) New gesture evoke method. Users do not need to switch back to application list to select an application, but can open another application right on the current interface, which makes SwipeLauncher operation faster and easier. 2) Combination of long-term and short-term preference. By letting the first row of applications the most frequently used ones, the second row algorithm predicted ones, SwipeLauncher ensures the accuracy of finding target application within the launcher while still keeps it stable.

In the aspect of gesture control, I also looked into other gesture selection methods, such as FastTap, Finger-Count Shortcuts, etc. These methods all designed a unique set of gestures which help users open certain menu items quickly, especially when users are already familiar with the system. SwipeLauncher also designed a unique gesture for item selection, which ensures easy item selection. What's more, SwipeLauncher also works well with single hand, because all the operations of SwipeLauncher can be handled by a thumb, and interaction is performed within thumb's "easy access area".

In the aspect of application selection, previous works mainly focused on predicting applications with the highest possibility to be used next, such as Dynamic Home Screen (Shin 2012), FALCON (Yan 2012) and PREPP (Parate 2013). However, these solutions did not consider the instability it brings, which not only leads to longer visual search time for the target, but also makes users face the risk of failure. SwipeLauncher solves these

problems by combining algorithm prediction and frequency statistics, so that the launcher keeps partially stabilized while ensures the prediction accuracy. Even though a part of the items in the launcher keep updating, users can still make decisions in a quite short time.

According to the overall operation (including launcher applications and full application list) result in chapter 6 Experiment, SwipeLauncher takes 1.907 s to open an application on mobile devices, which is 34.2% faster than original home screen launcher of Android. According to Yahoo's study [17], Android users have averagely 100 times of interaction with the phone per day. So, SwipeLauncher could help to save 65.2 seconds per day, 6.61 hours per year, the time and effort saving by SwipeLauncher in a long term can be quite considerable. Considering people are depending on mobile applications more and more nowadays, this improvement will become more valuable in the future.

For the operation within SwipeLauncher (full application list not included), the operation time predicted in chapter 5 Modeling Based Design Decisions is 1.968s, while user performance mentioned in chapter 6 Experiment is 1.552s. In experiment result, home screen is faster than prediction because the "Slow applications" in SwipeLauncher are fixed, which reduced visual search time for all the situations.

There are still some limitations of the experiment, which may influence the user performance in real world: 1) the accuracy of SwipeLauncher is predicted based LiveLab database. In the real world, when users just start using SwipeLauncher, there are not so many user data, which will affect prediction accuracy; 2) the "fast applications" in SwipeLauncher appear randomly during the experiment. But in the real world, SwipeLauncher will predict the next applications in use according to users' behavior, which means users should have some ideas about what will be the "fast applications". What's more, the applications in SwipeLauncher should be ordered by usage possibility. The applications with higher usage possibility are easier to access. This will also help to improve user performance.

I see the possibility of future work in the following areas: 1) Make customized optimization for different users, like customize launcher's triggering area size and location. Because users' preference of phone holding varies a lot, as well as users' finger length compares to mobile phone size. These will all lead to different satisfaction and performance on a same launcher. 2) Optimize the launcher layout. Such as make application icon with higher accessing possibility bigger, or arrange applications by color. So that users can access the application with higher usage possibility easier.

In my thesis work, I provided a new possibility for optimization of mobile multitasking behavior. SwipeLauncher proved that mobile multitasking

can be optimized with the cooperation of gesture control and application selection.

Bibliography

[1] Mit app inventor. http://appinventor.mit.edu/.

[2] Accenture. Video-Over-Internet Consumer Survey 2013: Multitasking and Taking Control Winning the trust of the sophisticated consumer. Tech. rep., 2013.

[3] Bailly, G., Lecolinet, E., and Guiard, Y. Finger-count & radial-stroke shortcuts: 2 techniques for augmenting linear menus on multi-touch surfaces. In *CHI* (2010), E. D. Mynatt, D. Schoner, G. Fitzpatrick, S. E. Hudson, W. K. Edwards, and T. Rodden, Eds., ACM, pp. 591–594.

[4] betaworks. #homescreen. http://homescreen.is/top-apps.

[5] Blackmon, M. H., Polson, P. G., Kitajima, M., and Lewis, C. Cognitive walkthrough for the web. In *Proceedings of the SIGCHI conference on human factors in computing systems* (2002), ACM, pp. 463–470.

[6] Böhmer, M., Hecht, B., Schöning, J., Krüger, A., and Bauer, G. Falling asleep with Angry Birds, Facebook and Kindle – a large scale study on mobile application usage. In *Proceedings of MobileHCI '11: 13th International Conference on Human Computer Interaction with Mobile Devices and Services, Stockholm, Sweden* (2011), pp. 47–56.

[7] Brumby, D. P., Cox, A. L., Chung, J., and Fernandes, B. How does knowing what you are looking for change visual search behavior? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2014), ACM, pp. 3895–3898.

[8] Card, S. K., Moran, T. P., and Newell, A. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM 23*, 7 (1980), 396–410.

[9] comScore. comscore's u.s. mobile app report available for download. https://www.comscore.com/Insights/Press-Releases/2014/8/comScore-s-US-Mobile-App-Report-Available-for-Download.

[10] Dobie, A. Multitasking on the samsung galaxy note 4. http://www.androidcentral.com/multitasking-samsung-galaxy-note-4-0.

[11] Google. Android 4.4 kitkat. http://www.android.com/intl/en_us/versions/kit-kat-4-4/.

[12] Gutwin, C., Cockburn, A., Scarr, J., Malacria, S., and Olson, S. C. Faster command selection on tablets with fasttap. In *CHI* (2014), M. Jones, P. A. Palanque, A. S. 0001, and T. Grossman, Eds., ACM, pp. 2617–2626.

[13] Kieras, D. E., and Hornof, A. J. Towards accurate and practical predictive models of active-vision-based visual search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2014), ACM, pp. 3875–3884.

[14] Nielsen, J., and Molich, R. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1990), CHI '90, ACM, pp. 249–256.

[15] Parate, A., Böhmer, M., Chu, D., Ganesan, D., and Marlin, B. M. Practical prediction and prefetch for faster access to applications on mobile phones. In *UbiComp* (2013), F. Mattern, S. Santini, J. F. Canny, M. Langheinrich, and J. Rekimoto, Eds., ACM, pp. 275–284.

[16] Pew Research Center. Mobile technology fact sheet, 2014. http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/.

[17] Sawers, P. Android users have an average of 95 apps installed on their phones, according to yahoo aviate data. http://thenextweb.com/apps/2014/08/26/android-users-average-95-apps-installed-phones-according-yahoo-aviate-data/.

[18] Shepard, C., Rahmati, A., Tossell, C., Zhong, L., and Kortum, P. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review 38*, 3 (2011), 15–20.

[19] Shin, C., Hong, J.-H., and Dey, A. K. Understanding and prediction of mobile application usage for smart phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 173–182.

[20] Wikipedia. Finite-state machine. `http://en.wikipedia.org/wiki/Finite-state_machine`.

[21] Wolf, T. V., Rode, J. A., Sussman, J., and Kellogg, W. A. Dispelling design as the black art of chi. In *Proceedings of the SIGCHI conference on Human Factors in computing systems* (2006), ACM, pp. 521–530.

[22] Wroblewski, L. Responsive navigation: Optimizing for touch across devices, 2012. `http://www.lukew.com/ff/entry.asp?1649`.

[23] Yan, T., Chu, D., Ganesan, D., Kansal, A., and Liu, J. Fast app launching for mobile devices using predictive user context. In *MobiSys* (2012), N. Davies, S. Seshan, and L. Zhong, Eds., ACM, pp. 113–126.

Appendix A

Informed Consent Sheets

This informed consent sheet is used before the experiment starts. It introduced the content of experiment and participants's task. Participants must read and sign the document before experiment, for get a rough understanding of the experiment process and authorizing the usage of operation data during the experiment.

Research Topic

Optimizing Multitasking User Behavior on Mobile

Researcher In Charge

The experiment will be fully responsible by Chen Wang, master student of ICT Innovation programme, Aalto University. The research is supported by User Interface group, School of Electrical Engineering, Aalto University.

Background And Objectives Of Study

The aim of this study is to evaluate a mobile application launcher. The experiment will last 0.5 - 1 hours, the whole process will be video recorded. During the study, you will be instructed to perform movements with your right hand. Your performance will be recorded by a mobile application on the experiment device. You are welcome take breaks at any time during the experiment.

The whole experiment includes two series: SwipeLauncher and Home Screen. Each part begins with a trail section, which helps you understand

how the task works, last about 5-10 min; then the real experiment section, last about 10-20min. The operation data will be used for further research analysis, for comparison of the two application launching methods.

Nature Of Study

The study represents the research field of Human Computer Interaction.

The studies conducted are of non-medical nature. This study does not aim to increase knowledge of health or of the reasons, symptoms, diagnostics, treatment or prevention of diseases or of the nature of diseases in general. No drugs are administered to the study subjects during the study.

Data Protection

The recorded data will be used for scientific purposes by the User Interface group of Aalto University, School of Electrical Engineering. In particular it will be used for scientific publications and presentations. It will not be given to any third party. All recorded data will be completely anonymized.

If you have any further questions regarding this experiment, please contact the following researcher:

Chen Wang. Aalto University, School of Computer Science. Email: chen.3.wang@aalto.fi

Consent Clause

I have read and understood the study information sheet given to me and I have sufficient information on the process of the study. I understand that my participation in the study is completely voluntary and that I have the right to discontinue my participation at any stage without any consequences. It has been explained to me that a designated researcher will, at my request, provide me with additional details of the general principles of the study and its progress or of the results concerning myself.

I have understood that the material and research data is gathered for scientific purposes only and it will not be given even in part to the study subject him/herself.

The research results related to me are only available to the researchers of the research group and they will not be presented to a third party without my written consent. The researcher in charge of the study may, however, give permission to his/her other cooperation partners to analyse my research

results for scientific purposes or ask for a professional consultation on possible unexpected incidental findings without separate consent provided that the anonymity of the results has been ensured. Any type of commercial exploitation of the results is prohibited.

I am not aware of any medical condition preventing me to attend the tests. I approve that in case there appears an unexpected incidental finding I will be informed about this.

By my signature, I confirm my participation in this study and agree to volunteer as a study subject.

Name **Place and Date**

_____ _____

Signature

_____

Appendix B

Participant Basic Information Questionnaire

The Participant basic information questionnaire is used for collecting participants' general information before experiment sections start. The aim is to confirm that the participants meet the needs of experiment and prove the selection of participants is not bias.

1. How old are you?

_____

2. What's your gender?
   A.Male B.Female

3. What?s your smart phone model? And what?s the system version (if you know)?

_____

4. How long have you been using your current smart phone?
   A. Under 6 month B. 6 month - 1 year
   C. 1-2 years D. Over 2 years

5. How long have you been using Android smart phone(s)?
   A. Under 1 year B. 1-2 years
   C. 2-3 years D. Over 3 years

6. How much time do you use smart phone every day?
   A. Under 1 hour B. 1-2 hour

C. 2-3 hour D. Over 3 hour

7. Please leave your email address (your email address will only be used within this research project)

_____

Appendix C

Assessment Questionnaire

Participants are asked to fill this assessment questionnaire after the experiment. This questionnaire is used for evaluation of users' subjective user experience of SwipeLauncher and Home Screen design.

Please rank the operation comfort degree of SwipeLauncher, 1-very uncomfortable 5-very comfortable				
1	2	3	4	5
Please rank the operation comfort degree of home screen, 1-very uncomfortable; 5-very comfortable				
1	2	3	4	5
Please rank the operation speed of SwipeLauncher, 1-very slow; 5-very fast				
1	2	3	4	5
Please rank the operation speed of home screen, 1-very slow; 5-very fast				
1	2	3	4	5
Please rank the operation complexity of SwipeLauncher, 1-very difficult; 5-very easy				
1	2	3	4	5
Please rank the operation complexity of home screen, 1-very difficult; 5-very easy				
1	2	3	4	5
Please give an overall ranking for SwipeLauncher, 1-very bad; 5-very good				
1	2	3	4	5
Please give an overall ranking for home screen, 1-very bad; 5-very good				
1	2	3	4	5