LensGesture: Augmenting Mobile Interactions with Back-of-Device Finger Gestures

Xiang Xiao[♦], Teng Han[§], Jingtao Wang[♦]

[♦]Department of Computer Science
University of Pittsburgh
210 S Bouquet Street
Pittsburgh, PA 15260, USA
{xiangxiao, jingtaow}@cs.pitt.edu

[§]Intelligent Systems Program
University of Pittsburgh
210 S Bouquet Street
Pittsburgh, PA 15260, USA
teh24@pitt.edu

ABSTRACT

We present LensGesture, a pure software approach for augmenting mobile interactions with back-of-device finger gestures. LensGesture detects full and partial occlusion as well as the dynamic swiping of fingers on the camera lens by analyzing image sequences captured by the built-in camera in real time. We report the feasibility and implementation of LensGesture as well as newly supported interactions. Through offline benchmarking and a 16-subject user study, we found that 1) LensGesture is easy to learn, intuitive to use, and can serve as an effective supplemental input channel for today's smartphones; 2) LensGesture can be detected reliably in real time; 3) LensGesture based target acquisition conforms to Fitts' Law and the information transmission rate is 0.53 bits/sec; and 4) LensGesture applications can improve the usability and the performance of existing mobile interfaces.

Categories and Subject Descriptors

H5.2 [**Information interfaces and presentation**]: User Interfaces. - *Graphical user interfaces; Input devices and strategies, Theory and methods.*

Keywords

Mobile Interfaces; Gestures; Motion Sensing; Camera Phones; LensGesture; Text Input.

1. INTRODUCTION

The wide adoption of multi-touch enabled large displays and touch optimized interfaces has completely changed how users interact with smartphones nowadays. Tasks that were considered challenging for mobile devices one decade ago, such as web browsing and map navigation, have experienced rapid growth during the past a few years [3]. Despites these success stories, accessing all the diverse functions available to mobile users on the go, especially in the context of one-handed interactions, are still challenging.

For example, when a user interacts with her phone with one hand, the user's thumb, which is neither accurate nor dexterous, becomes the only channel of input for mobile devices, leading to the notorious "*fat finger problem*" [2, 22], the "*occlusion problem*" [2, 18], and the "*reachability problem*" [20]. In contrast, the more responsive, precise index finger remains idle on the back of mobile devices throughout the interactions. Because of this, many compelling techniques for mobile devices, such as multi-touch, became challenging to perform in such a "*situational impairment*" [14] setting.

Many new techniques have been proposed to address these challenges, from adding new hardware [2, 15, 18, 19] and new input modality, to changing the default behavior of applications for certain tasks [22]. Due to challenges in backward software compatibility, availability of new sensors, and social acceptability [11], most of the solutions are not immediately accessible to users of existing mobile devices.

Figure 1. LensGesture in use for menu navigation.

In this paper, we present LensGesture (Figure 1), a new interaction technique that augments mobile interactions via finger gestures on the back camera of mobile devices. LensGesture detects full or partial lens covering actions as well as dynamic lens swiping actions by analyzing image sequences captured by the built-in camera.

We describe both implementation details and the benchmarking performance of the LensGesture algorithm. We show the potential and feasibility of leveraging on-lens finger gestures to enable a richer set of mobile interactions. Key contributions of this paper also include the design, exploration and performance evaluation of the LensGesture interaction technique, a quantitative performance study of LensGesture, and an empirical validation of LensGesture enhanced applications.

2. RELATED WORK

Related work fall into two categories: motion gesture interfaces, and back of device interaction.

ICMI'13, December 9–13, 2013, Sydney, NSW, Australia.

http://dx.doi.org/10.1145/2522848.2522850

2.1 Gestural Interfaces

Gesture is a popular and effective approach for mobile interfaces. Gestures on mobile devices can be performed by moving fingers or a stylus across a touch screen (i.e. touch-surface stroke gestures [23]), by moving the devices directly [10, 12, 16] (i.e. motion gestures), or a combination of both [8, 12, 16, 17]. Properly designed gestures can make mobile applications intuitive and enjoyable to use [11], improving performance for important tasks such as text entry [17, 19, 21], and making tasks such as selecting small on screen targets [22] or using application on the go easy to complete [8]. However, touch-surface stroke gestures could be tricky to perform [20] with a user's thumb in one-handed usage scenarios; at the same time, motion gestures require more space to complete and may also have social acceptability concerns [11].

LensGesture is similar to TinyMotion [16] in that both techniques rely on analyzing the image sequences captured by the built-in camera to detect motion. However, there are two major differences between these two methods. First, TinyMotion detects and interprets *background shifting* caused by the physical movement of mobile devices: a user needs to move or tilt the phone in order to interact with TinyMotion enabled applications. In comparison, LensGesture detects the *intrusion of finger* to the background while the mobile phone is being held still; Second, TinyMotion only supports "dynamic" motion gestures, which requires explicit device motion to activate a gesture while LensGesture also allows a user to perform "static" gestures such as covering the camera lens fully or partially.

The usage of in-the-air finger gestures in front of a mobile camera was investigated in [1, 7, 16] previously. Wang et al [16] discovered that 2D finger/hand movements in front of a camera can be detected by motion estimation algorithms in mobile interactions. An et al [1] tracked 2D, in-the-air finger gestures via skin color segmentation. In the PalmSpace project, Kratz et al [7] detected the 3D location and posture of a user's palm via an external depth camera. Our approach differs from these *in-the-air* gesture techniques in two ways. First, the LensGesture is directly performed *on the lens* of the camera. This paradigm greatly simplifies the detection algorithm and improves both the speed and accuracy of gesture detection. In addition, the bezel of camera lens provides natural tactile feedback during gesturing. Second, in addition to interactions enabled by motion sensing, LensGesture also systematically explores the design spaces of full/partial lens covering based static gestures.

The *direction scanner* envisioned by Ni and Baudisch [9] for ultra-small devices is similar to *Dynamic LensGesture* in terms of marking based input language. Instead of working for "disappearing mobile devices" in the future, LensGesture is designed as a complementary input channel to augment today's palm-size smartphones. The unique affordance of camera bezels also allows LensGestures to support unique input vocabulary such as partial covering gestures.

Hinckley and Song [6] systematically explored how two basic interactions, i.e. touch and motion, can be combined together via a set of "touch-enhanced motion" and "motion-enhanced touch" scenarios. Their sensor synaesthesia techniques [6] use either implicit device motion or explicit hand movements captured by built-in sensors such as accelerometers or gyroscopes. In contrast, LensGesture relies on back-of-device index finger and the camera to complement front-screen interactions when the device is holding still.

2.2 Back of Device Interactions

The LensGesture provides a pure-software, complementary input channel on the back of the device. Back of device interactions have been studied by researchers in recent years for both ergonomics concerns and practical benefits [2, 5, 13, 15, 18, 20].

Wobbrock et al. [20] discovered that index fingers on the back of mobile devices can outperform thumb finger on the front in both speed and accuracy. Wobbrock and colleagues [20] used a pocket-sized touchpad to simulate conditions in their study due to the limited availability of mobile devices with back-mounted touch surfaces in 2008. While devices equipped with a back side touchpad have started to appear in recent years, e.g. SONY PlayStation Vita and Motorola Spice XT300 smartphone, the mainstream mobile devices do not benefit directly from such inventions.

Back of device interaction techniques are especially intriguing on small devices. Operating on the backside of the device allows users to navigate menus with single or multiple fingers and interact with the device without occluding the screen. nanoTouch [2] and HybridTouch [15] rely on back-mounted touchpad to support inch-sized small devices, and LucidTouch [18] uses back-mounted camera and to track users' fingers on a tablet sized device and shows a semi-transparent overlay to establish a "pseudo-transparent" metaphor during interactions. Minput [5] has two optical tracking sensors on the back of a small device to support intuitive and accurate interaction, such as zooming, on the device. RearType [13] places physical keyboard keys on the back of the device, enabling users to type text using the rear keys while griping the device with both hands.

3. THE DESIGN OF LENSGESTURE

LensGesture is motivated by four key observations when using mobile devices. First, a user's index finger, which is usually the most nimble finger, stays idle during most interactions. Second, the built-in camera of mobile devices remains largely unused outside of the photographic applications. Third, the built-in camera lens is reachable by the user's index finger on the back of the device regardless of whether the user is operating the phone with one hand (thumb based interactions) or both hands (operating the phone with index finger on the dominant hand). Fourth, the edge and bezel of cameras are usually made of different materials and on different surface levels, which can provide natural tactile feedback for direct touching and swiping operations on the lens.

3.1 The LensGesture Taxonomy

We propose two groups of interaction techniques, *Static LensGesture* and *Dynamic LensGesture*, for finger initiated direct touch interactions with mobile cameras (Figure 2).

Static LensGesture (Figure 2, top row) is performed by covering the camera lens either fully or partially. Supported gestures include covering the camera lens in full (i.e. full covering gesture) and covering the camera lens partially (e.g. partially covering the left, right, and bottom region of the lens[1]). *Static LensGesture*

[1] According to informal tests, we found the top-covering gesture both hard to perform and hard to distinguish (when compared with left-covering gestures, Figure 3, third row, first and last images). So we intentionally removed the top-covering gesture as a supported *Static LensGesture*. Please also note that the definition of "top", "left", "right" and "bottom"

converts the built-in camera into a multi-state push button set. Interestingly, the edge/bezel of the camera optical assembly can provide natural tactile feedback to the user's index finger when performing static gestures. Froehlich et al [4] proposed a family of barrier pointing techniques that utilize the physical properties of screen edges on mobile devices to improve pen based target acquisition. LensGesture is unique in that it leverages the affordance of a camera's bezel to create a new touch input channel on the back of mobile devices.

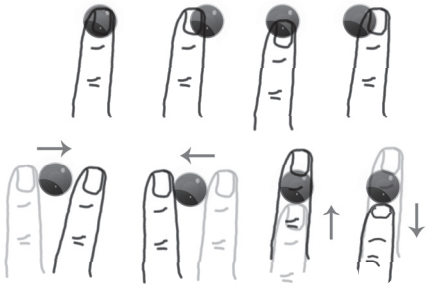

Figure 2. Top Row: Static LensGestures; Bottom Row: Dynamic LensGestures.

A user can also perform a *Dynamic LensGesture* (Figure 2, bottom row) by swiping her finger horizontally (left and right) or vertically (up and down) across the camera lens[2]. *Dynamic LensGestures* convert the back camera into a four-way, analog pointing device based on relative movement sensing. As we later show, allowing the direct swiping of fingers on camera lens significantly simplify the detection algorithm and improve the corresponding detection performance.

3.2 The LensGesture Algorithm

We designed a set of three algorithms to detect full coverage, partial coverage and dynamic swiping of fingers on the lens. Depending on usage scenarios, these three algorithms can be cascaded together to support all or part of the LensGesture set.

In all LensGesture detection algorithms, the camera is set in preview mode, capturing 144x176 pixel color images at a rate of 30 frames per second. We disable the automatic focus function and the automatic white balance function to avoid interference with our algorithms.

Static LensGesture - Full covering: The full covering gesture (Figure 3, second row) can be detected quickly and reliably via a linear classification model on the global mean and standard deviation of all the pixels in an incoming image frame in the 8-bit gray scale space.

The intuition behind the underlining detection algorithm is that when a user covers the camera's lens completely, the average illumination of images drops, while the illumination among pixels in the image will become homogeneous (i.e. smaller standard deviations).

depends on the holding position (e.g. portrait mode or landscape mode) of the phone.

[2] It is possible to create another Dynamic LensGesture by moving the finger close to or away from the camera lens. However, such gestures are relatively hard to perform when a user is holding the phone with the same hand. We leave this type of Dynamic LensGesture on z-axis to future work.

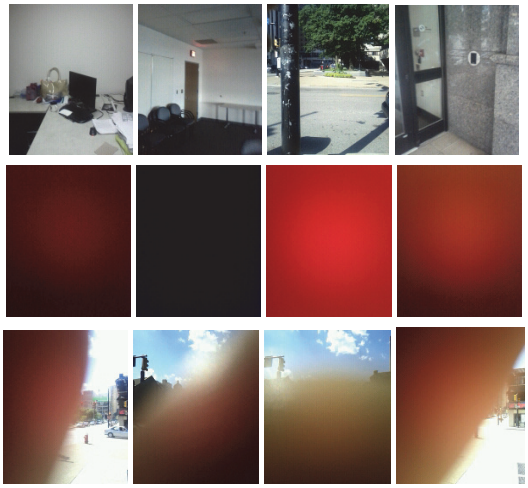

Figure 3. Samples images of Static LensGesture. First row: no gesture. Second row: full covering gestures. Third row: partial-covering gestures. Left to right: left-covering, right-covering, bottom-covering, and top-covering (not supported).

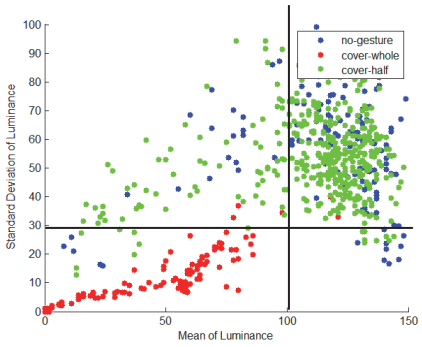

Figure 4. global mean vs. standard deviation all the pixels in images with (full-covering : red dots, partial covering: green dots) and without (blue dots) Static LensGestures. Each dot represents one sample image.

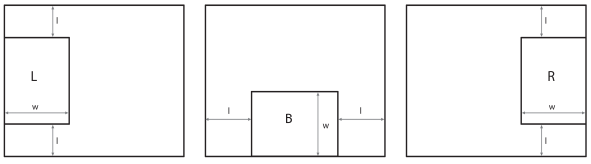

Figure 5. From left to right, extracting local features from Region L (*covering-left* classifier), Region B (*covering-bottom* classifier), and Region R (*covering-right* classifier).

Figure 4 shows a scatter plot of global mean vs. global standard deviation of 791 test images (131 contained no LensGesture; 127 contained *full-covering gestures*; 533 contained *partial covering gestures*). We collected test images from 9 subjects and in four different environments: 1) indoor bright lighting, 2) indoor poor lighting, 3) outdoor direct sunshine, and 4) outdoor in the shadow. All the subjects in the data collection stage were undergraduate and graduate students in a local university, recruited through school mailing lists. The number of samples in each environment condition is evenly distributed. When we choose mean <= 100, stdev <=30 as the linear decision boundaries for detecting full-covering gestures (highlighted in Figure 4), we can achieve an

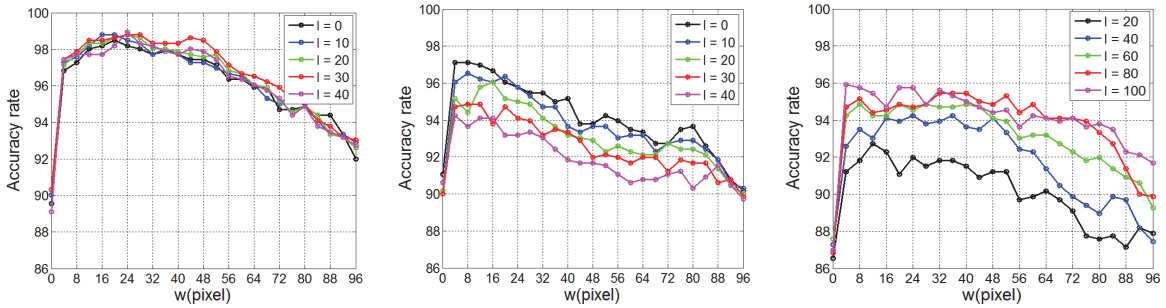

Figure 6. Classification accuracies of partial-covering classifiers. (Left to right: covering-left, covering-bottom, covering-right)

accuracy of 97.9%, at the speed of 2.7 ms per estimate. While more advanced detection algorithms could definitely improve the accuracy, we believe an accuracy of 97.9% is sufficient in interactive applicants where users can adapt their behaviors via real-time feedback.

Static LensGesture - Partial covering: To detect partial covering gestures in real time, we designed three serial cascaded binary kNN (k=5) classifiers to detect covering-left, covering-bottom, and covering-right gestures. After deciding that the current frame does not contain a full covering gesture, the image will be fed to the covering-left, the covering-bottom, and the covering-right classifier one after the other. If a partial covering gesture is detected, the algorithm will stop immediately, if not, the result will be forwarded to the next binary classifier. If no partial-covering gesture was detected, the image will be labeled as "no gesture". We adopted this cascading approach and the kNN classifier primarily for speed concerns.

Features we used in the kNN classifiers include both global features (mean, standard deviation, maximal and minimal illuminations in the image histogram) and local features (same features in a local bounding box, defined in Figure 5). There are two parameters (w, l) that control the size and location of the local bounding boxes. The (w, l) values (unit = pixels) should be converted to a relative ratio when used in different preview resolutions.

We use the data set described in the previous section, and ten-fold classification to determine the optimal values (w and l) for each classifier (Figure 6). As shown in Figure 6, we found that for the *covering-left* classifier, $w = 24$, $l = 40$ will give us the highest binary classification accuracy at 98.9%. For the *cover-bottom* classifier, $w = 4$, $l = 0$, gives the highest accuracy at 97.1%, for the *covering-right* classifier, $w = 4$, $l = 100$, gives the highest accuracy at 95.9%. The overall accuracy of the cascaded classification is 93.2%. The speed for detecting partial covering ranges from 16 – 42 ms.

Dynamic LensGesture: As reported by Wang, Zhai, and Canny in [16], TinyMotion users discovered that it is possible to put one's other hand in front of the mobile camera and control motion sensing games by moving that hand rather than moving the mobile phone. As shown in Figure 7, the fundamental causes of image change are quite different in TinyMotion and LensGesture. In TinyMotion (Figure 7, bottom row), the algorithm was detecting the *background shifting* caused by lateral movement of mobile devices. When performing Dynamic LensGestures (Figure 7, top row), the background keeps almost still while the finger tip moves across the lens. Another important observation is that in Dynamic LensGesture, a user's finger will completely cover the lens in one or two frames, making brute force motion estimation results noisy.

Figure 7. The difference between image sequences captured by LensGesture (up) and TinyMotion (down) in the same scene.

The *Dynamic LensGesture* algorithm is based on the TinyMotion algorithm with minor changes and additional post processing heuristics. Figure 8 shows the relative movements from the TinyMotion algorithm, as well as the actual images captured, when a left-to-right *Dynamic LensGesture* was performed. In Figure 8, we see that although the TinyMotion algorithm successfully captured the strong movements in x-axis (frames 3, 4, 5, 7, 8, 10, 11), estimations became less reliable (frame 6) when a major portion of the lens was covered. To address this issue, we use a variable weight moving window to process the raw output from the TinyMotion algorithm. We give the output of the current frame a low weight when a full covering action is detected.

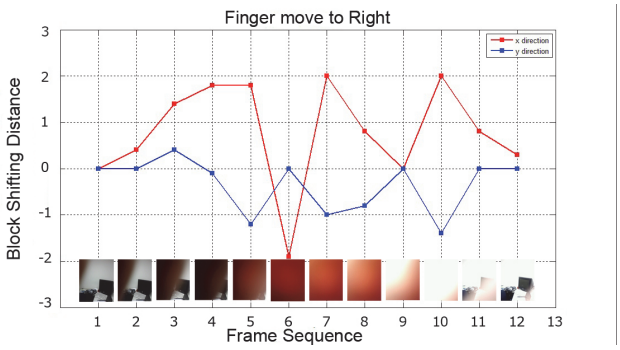

Figure 8. Plot of the distant changes in both x and y directions for 20 gesture samples.

We collected 957 sets of *Dynamic LensGesture* sample from 12 subjects. There were more than 30000 images in this data set. For each Dynamic LensGesture, depending on the finger movement speed, 10-20 consecutive images were usually captured. We achieve an accuracy of 91.3% for detecting *Dynamic LensGestures* on this dataset, at a speed of 3.9 ms per estimate. We looked deeper into the misclassified sample sequences and found that most errors were caused by the confusion between the

swiping down and the swiping left gestures. Most of the misclassified sequences looked confusing even to human eyes because the actual swiping actions were diagonal rather than vertical or horizontal. We attribute this issue to the relative positioning between the finger and the lens, as well as the lack of visual feedback during data collection.

To explore the efficacy of LensGesture as a new input channel, we wrote six applications (LensLock, LensCapture, LensMenu, LensQWERTY, LensAlbum, and LensMap). All these prototypes can be operated by Static or Dynamic LensGestures (Figure 9). All but one application (LensQWERTY) can be operated with one hand.

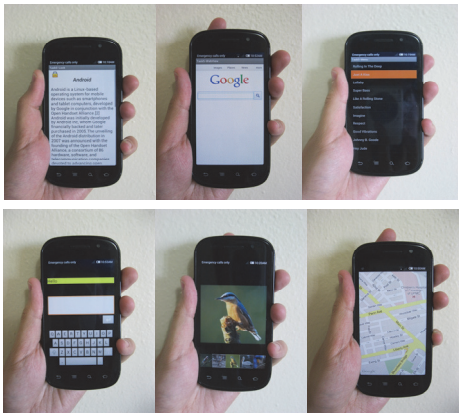

Figure 9. Sample LensGesture applications. From left to right, top to bottom - LensLock, LensCapture, LensMenu, LensQWERTY, LensAlbum, and LensMap.

LensLock leverages the Static LensGesture and converts the camera into a "clutch" for automatic view orientation changes. When a user covers the lens, LensLock locks the screen at the current landscape/portrait format until the user's finger releases from the lens. LensLock can achieve the same "pivot-to-lock" technique proposed by Hinckley [6] without using the thumb finger to touch the front screen, which may lead to unexpected state changes.

LensQWERTY uses Static LensGesture to control the SHIFT state of a traditional on screen QWERTY keyboard. The user can use the hand holding the phone to toggle the SHIFT state when the other index finger is being used for typing.

LensAlbum and LensMap are two applications that leverage *Dynamic LensGestures* for one-handed photo album/map navigation. These two application shows that LensGesture can alleviate "*fat finger problem*" and the "*occlusion problem*" by avoiding direct thumb interaction on the touch screen. The LensMenu also illustrates a feasible solution to the "*reachability problem*" via a supplemental back-of-device input channel enabled by LensGestures.

3.3 Feasibility

Three major concerns arise for interacting with cameras on mobile devices in such a "non-traditional" approach. First, is it possible and comfortable to reach the camera on the back with a user's index finger under normal grip? Second, does covering and swiping directly on the surface of the lens scratch or damage the lens? Third, will the LensGesture algorithm drain the battery of a smartphone quickly?

We carried out an informal survey to answer the first question. Reviewing the smartphones in the market, we found that most phones have 4 to 5 inch touch screens, such as Nokia Lumia 900 (4.3"), Samsung Galaxy Nexus (4.65"), LG Lucid (4"), Motorola Droid 4 (4"), Samsung Focus S (4.3"), HTC Vivid (4.5"). Some have smaller screens, like iPhone 4S (3.5") and some have bigger ones, like Samsung Galaxy Note (5.3"). Basically, the phones with various sizes are easy to be hold with one hand (Figure 10) and the lens can be easily reached with one's index finger. The only exception we are aware of is an Android based MP3 music player named Archos 32. Its camera is located in the bottom left region of the device.

We also consulted design experts in leading mobile phone manufacturers to see if covering and swiping directly on the surface of the lens scratch or damage the lens. According to them, mainstream optical assemblies in the mobile phones have been carefully designed to avoid damages from accidental drops, scratches and collisions. The external lens component in an optical unit is usually made of crystal glass, cyclic olefin copolymer, or sapphire. While they are not scratch free, these materials are strong enough to resist frictions caused by finger touch. Interestingly, the surrounding bezel of the camera is usually made of a different material, slightly higher than the external lens surface. Such material and height difference provide good tactile feedback for both locating the lens and performing different LensGestures (especially partial occlusion gestures and dynamic gestures).

Figure 10. Performing LensGesture on different phones.

We ran a total of four mini experiments to quantify the impact of LensGesture to battery life. We used a Google Nexus S smartphone (running Android 4.0.3) in the follow-up battery tests. First, we measured the battery life while LensGesture was continuously running in the background with the camera turned on and the backlight of the screen turned off. Our test phone ran 4 hours 13 minutes after a full charge. Second, when the backlight of the screen was turned on to minimal backlight, the same phone lasted 2 hours 35 minutes. Third, when we turn the flashlight of the camera to always on, and screen backlight to minimal, our smartphone lasted 2 hours 13 minutes. In the last controlled condition, we tested a regular android app (i.e. the Alarm Clock) with minimal backlight; the battery lasted 4 hours 11 minutes.

We have two major findings from the battery experiments. 1) A major power drain of the modern smartphone is the screen backlight. This finding agrees with the existing battery test for camera based motion sensing on features phones [16]. 2) Paradoxically, the flashlight feature of today's smartphones only takes minimal amount of power so the inclusion of flashlight to

improve low-light performance may be worth exploring in future research.

3.4 Implementation

We implemented LensGesture on a Google Nexus S smartphone. We wrote the LensGesture algorithms and all the LensGesture applications in Java. The LensGesture algorithm can be implemented in C/C++ and compiled to native code via Android NDK if higher performance is needed.

4. USER STUDY

Although the results of our LensGesture algorithm on pre-collected data sets were very encouraging, a formal study was necessary to understand the capabilities and limitations of LensGesture as a new input channel.

4.1 Experimental Design

The study consisted of six parts:

Overview. We first gave participants a brief introduction to the LensGesture project. We explained each task to them, and answered their questions.

Reproducing LensGestures. This session was designed to test whether users could learn and comfortably use the LensGestures we designed, and how accurate/responsive the gesture detection algorithm was in a real world setting. A symbol representing either a *Static LensGesture* or a *Dynamic LensGesture* was shown on the screen (Figure 11, (1) (2)). Participants were required to perform the corresponding LensGesture with their index fingers as fast and as accurately as possible. The application would still move to the next stimulus if a user could not perform the expected gesture within the timeout threshold (5 seconds). A user completed 20 trials for each supported gesture. The order of the gestures was randomized.

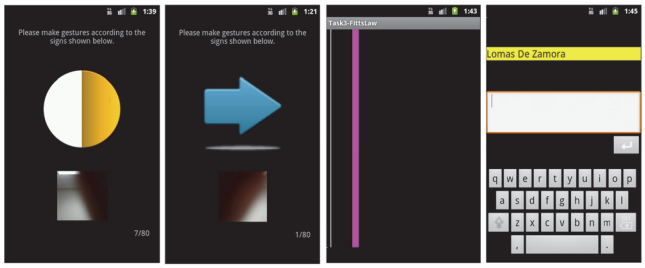

Figure 11. Screen shots of applications in the user study.

Target Acquisition/Pointing. The goal of this session was to quantify the human performance of using LensGesture to perform target acquisition tasks. For each trial, participants needed to use *Dynamic LensGestures* to drive an on-screen cursor from its initial position to the target (Figure 11, (3)). After the cursor hit the target, participants were required to tap the screen to complete the trial. Regardless of whether participants hit the target or not, the target acquisition screen disappeared and an information screen indicating the number of remaining trials in the current block would show up. We encouraged participants to hit the target as fast as possible and as accurately as possible. Each participant completed 160 randomized trials.

Text Input. In this task, we compared the performance of standard Android virtual keyboard with the LensQWERTY keyboard (Figure 11, (4)).

Each participant entered 13 short phrases in each condition. The 13 test sentences were: "Hello", "USA", "World", "Today", "John Smith", "Green Rd", "North BLVD", "Lomas De Zamora", "The Great Wall", "John H. Bush", "Sun MicroSystem", "Mon Tue Wed Thu", and "An Instant In The Wind". These test sentences were intended to maximize the usage of LensGesture based shifting feature and simulate commonly used words in a mobile environment (person names, place names etc).

Other Applications. In this session, Participants were presented with five LensGesture applications we created (LensLock, LensCapture, LensMenu, LensAlbum, and LensMap, Figure 9). After a brief demonstration session, we encouraged the participants to play with these applications as long as they wanted.

Collect Qualitative Feedback. After a participant completed all tasks, we asked him or her to complete a questionnaire. We also asked the participant to comment on each task, and describe one's general feeling towards LensGesture.

4.2 Participants and Apparatus

16 subjects (4 females) between 22 and 30 years of age participated in our study. 15 of the participants owned a smartphone. The user study was conducted in a lab with abundant light. All of the participants completed all tasks.

Our experiments were completed on a Google Nexus S smartphone with a 480 x 800 pixels display, a 1GHz ARM Cortex-A8 processor, running Android 4.0.3. It has a built-in 5.0 mega-pixel back camera located at the upper right region.

5. EVALUATION RESULTS

5.1 Reproducing LensGestures

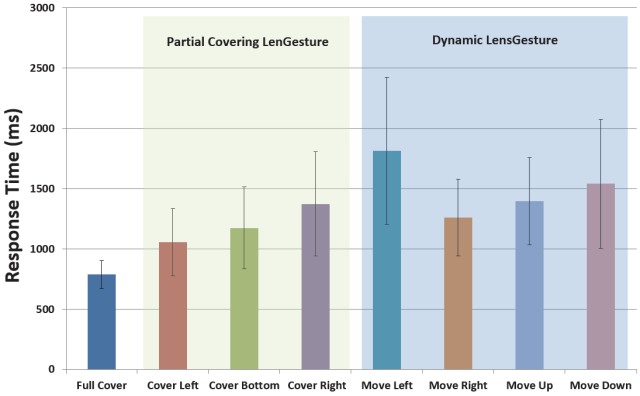

Figure 12. Average response time of Static and Dynamic LensGestures with one standard deviation error bars.

As shown in Figure 12, the time needed to perform a static gesture varied on gesture type. Repeated measure variance analysis showed significant difference due to gesture type: $F(7, 120) = 9.7$, $p < .0001$. Fisher's post hoc tests showed that the response time of full-occlusion gesture (787 ms) was significantly shorter than any of the partial occlusion gestures (left = 1054 ms, $p < 0.01$; right = 1374 ms, $p < 0.0001$; bottom= 1175 ms, $p < 0.0001$) and dynamic gestures. The left partial occlusion gesture is

significantly faster than right partial occlusion, $p < 0.01$, the speed differences between other partial occlusion gestures are not significant. For Dynamic Gestures, the move-right gesture (1258.6 ms) was significantly faster than move-left (1815.2 ms, $p < 0.01$) and move-down (1540.6 ms, $p < 0.05$) gestures, but there was no significant time difference between move-right and move-up (1395.7 ms, $p = 0.15$). The move-up gesture was also significantly faster than move-left ($p < 0.01$). The differences in detection time of Dynamic LensGestures might be caused by the location of the camera. The camera was located on the upper right region of the experiment device, making it easier to make the move-right and move-up gestures.

5.2 Target Acquisition/Pointing

2560 target acquisition trials were recorded. 2298 pointing trials were successful, resulting in an error rate of 10.2%. This error rate is about twice as that of popular pointing devices in Fitts' law studies.

After adjusting target width W for the percentage errors, linear regression between movement time (MT) and Fitts' index of difficulty (ID) is shown in Figure 13:

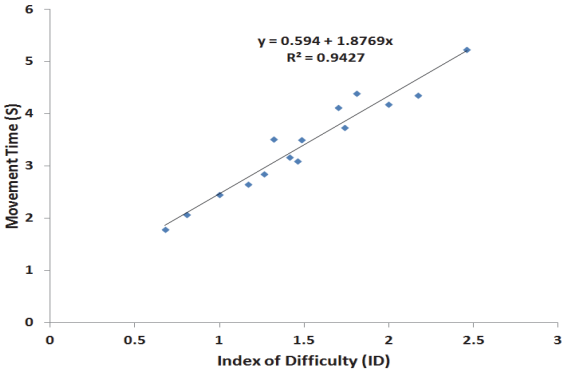

Figure 13. Scatter-plot of the Movement Time (MT) vs. the Fitts' Law Index of Difficulty (ID) for the overall target acquisition task controlled by Dynamic LensGestures.

$$MT = 0.594 + 1.8769 \log_2(A/W_e+1) \quad (sec)$$

In the equation above, A is the target distance and W_e is the effective target size. While the empirical relationship between movement time (MT) and index of difficulty ($ID = \log (A/W_e + 1)$) followed Fitts' law quite well (with $R^2 = 0.9427$, see Figure 14), the information transmission rate $1/b = 1/1.8769 = 0.53$ bits/sec) indicated a relatively low performance for pointing. In comparison, Wang, Zhai and Canny [16] reported a 0.9 bits/sec information transmission rate for device motion based target acquisition on camera phones. We attribute the performance difference to the usage patterns of *Dynamic LensGestures* - due to the relatively small touch area of the built-in camera, *repeated* finger swiping actions are needed to drive the on-screen cursor for a long distance. We believe that the performance of LensGesture could be improved with better algorithms and faster camera frame rates in the future. More importantly, since LensGesture can be performed *in parallel* with interaction on the front touch screen, we believe that there are opportunities to use LensGesture as a supplemental input channel and even use LensGesture as a primary input channel when the primary channel is not available.

5.3 Text Input

In total, 6273 characters were entered (including editing characters) in this experiment. There were a total of 42 upper case characters in the test sentences that required shifting operations when using the traditional keyboard.

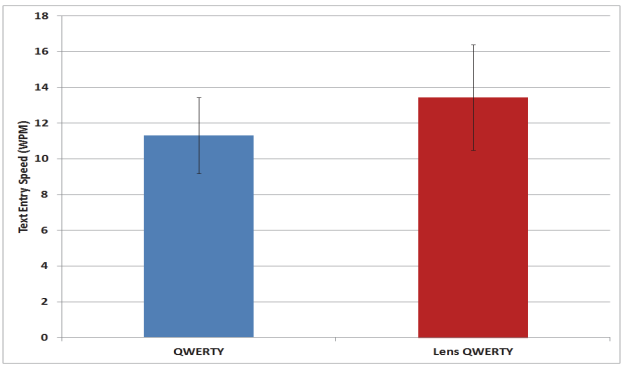

Figure 14. Text entry speed from the experiment with one standard deviation error bars.

As shown in Figure 14, the overall speed of LensGesture enabled virtual keyboard, i.e. LensQWERTY (13.4 wpm), was higher than that of the standard virtual keyboard (11.7 wpm). The speed difference between these two keyboards was significant $F(1, 15) = 4.17$, $p < 0.005$.

The uncorrected error rate was less than 0.5% for each condition. The average error rates for the standard keyboard and LensQWERTY were 2.1% and 1.9% respectively. The error rate difference between the standard keyboard and LensQWERTY was not significant ($p = 0.51$).

5.4 Other Applications

All participants can learn to use the LensGestures applications we provided with minimal practice (< 2 min). Almost all participants commented that the portrait/landscape lock feature in LensLock was very intuitive and much more convenient than alterative solutions available on their own smartphones. Participants also indicated that changing the "shift" state of a virtual keyboard via LensGesture was both easy to learn and time saving.

6. DISCUSSIONS AND FUTURE WORK

The participants reported positive experiences with using LensGesture. All participants consistently rated LensGesture as "useful" on the closing questionnaire using a five-point Likert scale. When asked about how easy it was to learn and use LensGesture, 13 participants selected "easy", 3 participants rated the experience "neutral". 9 participants commented explicitly that they would use LensGesture on their own smartphones. 4 of them expressed a very strong desire to use LensGesture applications every day.

Our study also revealed usability problems in the current implementation. Some participants noticed that accidental device movements were recognized as Dynamic LensGestures from time to time. We suspect that such kind of accidental device moments could be one major cause of the relatively high error rate in our target acquisition task. These false positives can be reduced by enforcing the full lens covering heuristic illustrated in Figure 8 in the future.

LensGesture has three advantages when compared with most existing techniques:

- *Technology availability*. LensGesture is a pure software approach. It is immediately available on today's main stream smartphones.
- *Minimal Screen Estate*. LensGestures can be enabled without using any on-screen resources.
- *Social Acceptability* [11]. When compared with other motion gesture related techniques such as TinyMotion [16], and DoubleFlip [12], interacting with LensGesture applications is barely noticeable to others.

LensGesture also has its own disadvantages. First, due to its internal working mechanism, LensGesture cannot co-exist with picture taking and video capturing applications. Second, since LensGesture detects the illumination changes caused by finger covering activities, it might not work well in extremely dark environments. However, this restriction may be relieved by leveraging the camera flashlight. Third, given the relatively low information transmission rate (0.53 bits/sec), it could be slightly tedious to complete pointing tasks via LensGesture for an extended amount of time.

Our current research has only scratched the surface of LensGesture-based interactions. For example, an adaptive Control-Display (C/D) gain algorithm could be implemented to improve the performance of Dynamic LensGesture driven target acquisition tasks, where repetitive finger movements are necessary. Custom cases or attachments with grooves for guiding finger movements could be made to enable EdgeWrite style gesture input [21] via LensGesture.

The LensGesture channel is orthogonal to most existing input channels and techniques on mobile phones. Acting as a supplemental input channel, LensGesture can co-exist with software or hardware based front or back-of-the-device interaction techniques. We believe that there are many new opportunities in the design space of multi-channel, multi-stream interaction techniques enabled by LensGesture.

7. CONCLUSIONS

In this paper, we present LensGesture, a pure software approach for augmenting mobile interactions with back-of-device finger gestures. LensGesture detects full and partial occlusion as well as the dynamic swiping of fingers on the camera lens by analyzing image sequences captured by the built-in camera in real time. We report the feasibility and implementation of LensGesture as well as newly supported interactions. Both offline benchmarking results and a 16-subject user study show that LensGestures are easy to learn, intuitive to use, and can complement existing interaction paradigms used in today's smartphones.

8. ACKNOWLEDGEMENTS

The authors would like to thank Andrew Head, Wencan Luo, Antti Oulasvirta, Jesse Thomason, Daniel Walker, and Emilio Zegarra for their suggestions and help. We also thank the anonymous reviewers for their constructive feedback.

9. REFERENCES

[1] An, J., Hong, K., Finger gesture-based mobile user interface using a rear-facing camera, In *Proc. ICCE 2011*, pp 303-304.

[2] Baudisch, P. and Chu, G. Back-of-Device Interaction Allows Creating Very Small Touch Devices. In *Proc. CHI 2009*.

[3] Callcredit Information Group, Mobile Web Traffic Triples in 12 Months, http://finance.yahoo.com/news/mobile-traffic-triples-12-months-050000265.html 9/8/2013.

[4] Froehlich, J., Wobbrock, J., Kane, S., Barrier Pointing: Using Physical Edges to Assist Target Acquisition on Mobile Device Touch Screens, In Proc. ASSETS 2007.

[5] Harrison, C., and Hudson, S., Minput: Enabling Interaction on Small Mobile Devices with High-Precision, Low-Cost, Multipoint Optical Tracking, *In Proc. CHI 2010*.

[6] Hinckley, K., Song, H., Sensor Synaesthesia: Touch in Motion, and Motion in Touch, In *Proc. CHI 2011*.

[7] Kratz, S., Rohs, M., et al, PalmSpace: continuous around-device gestures vs. multitouch for 3D rotation tasks on mobile devices, In *Proc. AVI 2012*.

[8] Lu, H., Li, Y., Gesture Avatar: A Technique for Operating Mobile User Interfaces Using Gestures, In *Proc. CHI 2011*.

[9] Ni, T., and Baudisch, P., Disappearing Mobile Devices, In *Proc. UIST 2009*.

[10] Rekimoto, J., Tilting Operations for Small Screen Interfaces. In *Proc. UIST 1996*, pp. 167–168.

[11] Rico, J. and Brewster, S.A. Usable Gestures for Mobile Interfaces: Evaluating Social Acceptability. In *Proc. CHI 2010*.

[12] Ruiz, J., Li, Y., DoubleFlip: a Motion Gesture Delimiter for Mobile Interaction, In *Proc. CHI 2011*.

[13] Scott, J., Izadi, S., et al, RearType: Text Entry Using Keys on the Back of a Device, In *Proc. of MobileHCI 2010.*

[14] Sears. A., Lin M., Jacko, J. and Xiao, Y., When computers fade… Pervasive computing and situationally-induced impairments and disabilities, In *Proc of HCI International 2003*, Elsevier Science.

[15] Sugimoto, M., and Hiroki, K., HybridTouch: an Intuitive Manipulation Technique for PDAs Using Their Front and Rear Surfaces, In *Proc. MobileHCI 2006*, 137-140.

[16] Wang, J, Zhai, S., Canny, J., Camera Phone Based Motion Sensing : Interaction Techniques, Applications and Performance Study. In *Proc. UIST 2006*.

[17] Wang, J., Zhai, S., Canny, J., SHRIMP - Solving Collision and Out of Vocabulary Problems in Mobile Predictive Input with Motion Gesture, In *Proc. CHI 2010*.

[18] Wigdor, D., Forlines, C., Baudisch, P., et al. LucidTouch : a See-Through Mobile Device. In *Proc. UIST 2007*.

[19] Wobbrock, J.,Chau, D., Myers, B., An Alternative to Push, Press, and Tap-Tap-Tap: Gesturing on An Isometric Joystick for Mobile Phone Text Entry, In *Proc. CHI 2007*.

[20] Wobbrock, J., Myers, B., and Aung, H. The Performance of Hand Postures in Front- and Back-of-Device Interaction for Mobile Computing. *International Journal of Human-Computer Studies* 66 (12), 857-875.

[21] Wobbrock, J., Myers, B., Kembel, J., EdgeWrite: A Stylus-Based Text Entry Method Designed for High Accuracy and Stability of Motion, In *Proc of UIST 2003*.

[22] Yatani, K., Partridge, K., Bern, M., and Newman, M.W. Escape: a Target Selection Technique Using Visually-Cued Gestures. In *Proc. CHI 2008*, ACM Press (2008), 285-294.

[23] Zhai, S., Kristensson, P.O., et al, Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review, *Foundations and Trends in Human–Computer Interaction* Vol. 5, No. 2, 97–205, 2012.