MIS Final Project: Back-of-Device

Lars Meyer
114719
lars.meyer@uni-weimar.de

Everett. M. Mthunzi
119119
mondliwethu.everett.mthunzi@uni-weimar.de

ABSTRACT

As part of fulfillment for the Mobile Information Systems final Project, the work presented in this report is based on an idea inspired by [1]. Following the papers' argument and motivation behind unoccluded interactions utilizing the backside of mobile devices, our approach fixated on the same aim, focusing on achieving mobile device backside interaction. Identifying use of the rear camera as an input sensor (which presented itself as a perfect ergonomic solution), we successfully implemented back-of-device interactions. Our approach, methods, work-flow, results, shortcomings as well as future recommendations are presented in this report.

INTRODUCTION

The original work in [1] presents the argument that the key to touch-enabling very small devices is to employ touch input on the devices' backside, directing research efforts towards adding pointing input capabilities on the backside of devices. Emphasizing four form factors as part of their argument, the paper highlights how using the devices' front leads to the eventuality of the "fat finger problem", where fingers could occlude contents and prevent precision.

Although [1] develops and presents a 2.4" prototype device as part of four contributory research findings including design guidelines, our implementation followed a more generic approach which entailed employing an android mobile phone and its internal sensors.

Our implementation would make use of the phones rear camera sensor to implement intuitive gesture control from the back of the device utilizing common finger touch gestures (i.e. swiping and tapping).

DESIGN

Implementation sought and successfully achieved two main objectives.

- designing and developing a standalone module for recognizing gestures using the rear camera.
- developing a "proof of concept" application through which the developed gesture recognition module could be subsequently tested and deployed.

Backhand Module

"Backhand" (`https://github.com/elmeyer/backhand`) is the module we developed for Tap and Swipe detection. It can be imported into any Android Studio project, so that any app can be augmented by its capabilities. The module only depends on OpenCV and deals with most camera-related setup and processing tasks itself, transparent to the user and to the rest of the app. To use it in an app, all that needs to be done is to implement the `onTap` and `onSwipe` methods, performing the desired actions when a Tap or a Swipe is registered.

In the background, the Backhand module continuously gathers low-resolution, greyscale images from the camera sensor. It then analyses their luminance, either locally or globally. Local luminance analysis is helpful for determining the direction of a Swipe, and for reducing processing load. Global analysis increases the accuracy of Tap detection in particular, but comes with a significant performance penalty.

The detection is based on simple thresholds for the respective image areas. For example, for a Tap, (ideally) the entire image should be dark for a short amount of time, whereas for a Swipe, the image is subdivided into 6 "stripes" (left, center and right both vertically and horizontally), which allows to determine the direction of the Swipe. If e.g. at first the left vertical stripe is dark, and then subsequently the entire image, we classify this as a Swipe to the right.

To try and reduce the load this detection causes for the mobile device's processor and increase the speed, we opted for computing the luminance of the stripes in parallel. Every stripe is associated with a `Thread` object, which can be run over and over again, each time delivering updated results for the luminance of the current image's selected region. This allows us to not analyse the entire image all the time, but to only focus on the regions which are currently of interest, such as the edges of the image if a swipe isn't currently "in progress". Nevertheless, this way of gesture detection is still significantly computationally intensive and results in less than ideal performance on all but the most powerful devices; an issue which we will further discuss below.

Media Player

A minimalistic media player was developed for proof of concept. The media player interface would provide immediate visual and audio feedback where the immediate feedback would increase back-of-device interactive usability, supporting gesture input to be translated into visual and audible effects.

DISCUSSION

The project workflow, shortcomings and future recommendations are detailed in this section.

Low-light Usability

Currently, the gesture detection will not function in low-light conditions. Using the flashlight for this purpose has been considered, but, in light of the already present performance issues and resulting battery drain, was not pursued any further.

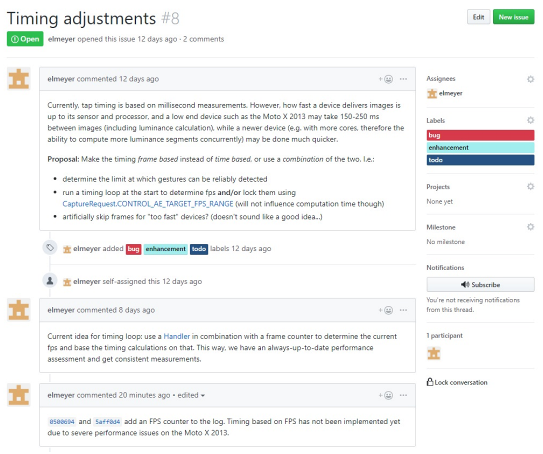
Gesture Detection, Calibration and Performance

As mentioned before, the gesture detection as we perform it requires a significant amount of resources, since for every frame the average luminance of at least some areas of the image has to be calculated. This oftentimes led to single-FPS performance on both development devices. In turn, this caused significant issues for calibrating the gesture detection and finding the correct thresholds. Ideally, 30 FPS or higher are desirable, since this provides a higher resolution of movement changes and improves detection accuracy significantly. However, since this number was not achieved, the detection accuracy in its current state is not ideal.

To improve performance somewhat, at least Tap detection can be switched to a single-pixel mode that only looks at the center pixel in the image and bases its detection on the luminance value of that pixel. This comes with disadvantages for both detection accuracy and ease of use, since the aim now has to be very accurate for a tap to register. No such remedy is for Swipe detection however, and thus it suffers from many wrongly detected Swipes, especially concerning the Swipe direction. However, accuracy can be improved by swiping at a consistent and slow pace, and allowing nothing but the fingertip to obstruct the lens during the swipe.

Workflow

Github played an integral role, ensuring source version control and seamless collaboration. Issues, bugs, enhancements and todos were cited, managed and resolved through Github. A glimpse of the challenges faced and choices made during development can be obtained by reading through the issues and comments at `https://github.com/everettmthunzi/back-of-device/issues`.



Timing adjustments #8

Open elmeyer opened this issue 12 days ago · 2 comments

elmeyer commented 12 days ago

Currently, tap timing is based on millisecond measurements. However, how fast a device delivers images is up to its sensor and processor, and a low end device such as the Moto X 2013 may take 150-250 ms between images (including luminance calculation), while a newer device (e.g. with more cores, therefore the ability to compute more luminance segments concurrently) may be done much quicker.

Proposal: Make the timing *frame based* instead of *time based*, or use a *combination* of the two. I.e.:

- determine the limit at which gestures can be reliably detected
- run a timing loop at the start to determine fps **and/or** lock them using CaptureRequest.CONTROL_AE_TARGET_FPS_RANGE (will not influence computation time though)
- artificially skip frames for "too fast" devices? (doesn't sound like a good idea...)

elmeyer added bug enhancement todo labels 12 days ago

elmeyer self-assigned this 12 days ago

elmeyer commented 8 days ago

Current idea for timing loop: use a Handler in combination with a frame counter to determine the current fps and base the timing calculations on that. This way, we have an always-up-to-date performance assessment and get consistent measurements.

elmeyer commented 20 minutes ago • edited

`0500694` and `5aff0d4` add an FPS counter to the log. Timing based on FPS has not been implemented yet due to severe performance issues on the Moto X 2013.

Assignees: elmeyer

Labels: bug, enhancement, todo

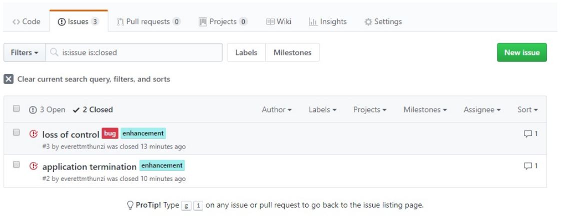
Projects: None yet

Milestone: No milestone

Notifications: You're not receiving notifications from this thread.

1 participant

Figure 1: identifying and resolving critical issues



is:issue is:closed

3 Open ✓ 2 Closed

loss of control bug enhancement
#3 by everettmthunzi was closed 13 minutes ago

application termination enhancement
#2 by everettmthunzi was closed 10 minutes ago

ProTip! Type `g` `i` on any issue or pull request to go back to the issue listing page.

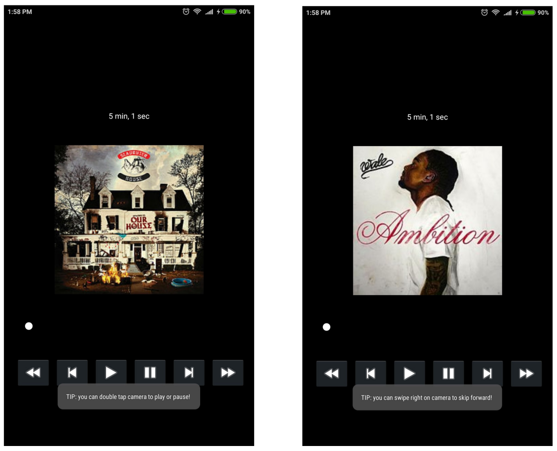
Figure 2: testing, milestones and enhancements

Generalization to other Applications

Since we chose to implement the gesture detection in the Backhand module, this functionality could be added to many Android apps, even existing ones. Which actions to map to the gestures will have to be decided by the developer. It is imaginable that the gesture detection could be used for system actions too, like increasing or lowering the ringtone volume or switching apps byswiping; however, due to the large resource impact of continuously running the image processing, this would be immensely detrimental to battery life and, on lower-end devices, usability as a whole due to lag. Therefore, a case-by-case adaptation of the features that the Backhand module offers to the specific app that uses it is recommended. If a way to improve especially the Swipe detection can be implemented, actions such as scrolling or moving around a map using Swipes may be imaginable.

RESULTS

We have managed to achieve both Tap and Swipe recognition and the triggering of individual actions based on the detected gesture. A demonstration of this can be seen in the video that is part of the final submission. The music player allows playing and pausing as well as skipping through the song based on Taps. Furthermore, the volume can be adjusted through Swipes up, and songs can be skipped through Swipes left and right. However, as discussed above the accuracy and performance of the gesture detection are limited by the device's capabilities and not properly calibrated, leading to false positives or misdetections. Overall, the presented functionality suffices as a proof of concept, keeping in mind the issues detailed above.



TIP: you can double tap camera to play or pause!

TIP: you can swipe right on camera to skip forward!

Figure 3: back of device: music player

CONCLUSION

Development efforts directed towards achieving back-of-device interactions identified the rear camera as an input sensor capable of multi-context applications. A strong takeaway being, outside apparent revision to avert the outlined pitfalls, the camera data input processing approaches can be potentially extended uniquely based on the application context.

Bibliography

[1] P. Baudisch and G. Chu. Back-of-device interaction allows creating very small touch devices. In *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*, page 1923, New York, New York, USA, 2009. ACM Press.