Slight:
Context-Aware Flashlight

Ja Jan Hsu
Leilei Liu
Leon Lai
Siddharth Coontoor
Turki Alenezi

INTRODUCTION

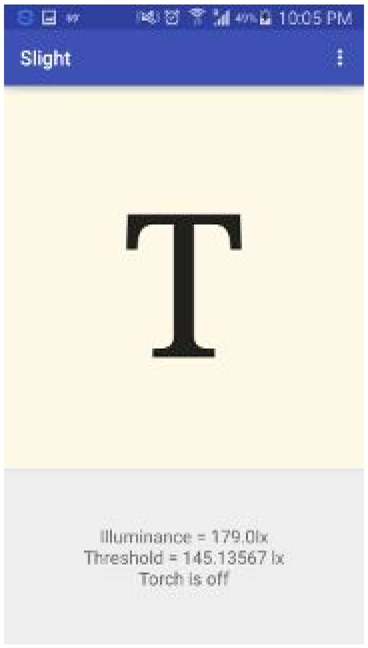
The goal of this project is to design, implement, and test an Android application that takes action based on a combination of explicit inputs from the user and implicit inputs from sensors, with focus on signal processing and inference. *Slight*, the application, turns on the user's phone camera torch when it infers that the user feels the environment is too dark and turns it off when it infers that the user feels otherwise.

The rest of this document first describes the functionality, user interface design and architecture of the application. Afterwards, the design choices made from the inception of the project to the completed deliverable are presented in roughly chronological order.

THE APPLICATION

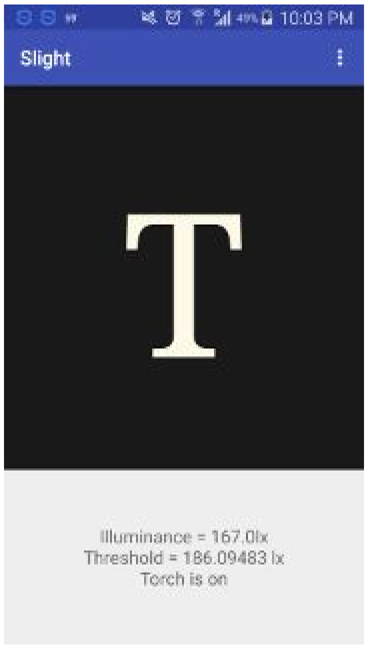
The application is named *Slight*, a combination of "sight" and "light". This app enhances visibility of the surrounding when it is dark or when the user comes from a brighter environment by turning on the camera light. After the app thinks the user is used to the place's luminosity, the light will be turned off.

There are two input categories, implicit input from the light sensor and explicit input from the user. The light sensor input provides the context. As a function of the current illuminance, the app calculates a value for inferring the user's need for a flashlight. The semantics of this value is the illuminance at which it is subjectively too dark. The user may also directly request for the flashlight by issuing a button press. The semantics of this input is that the app has inferred the user's intention incorrectly.



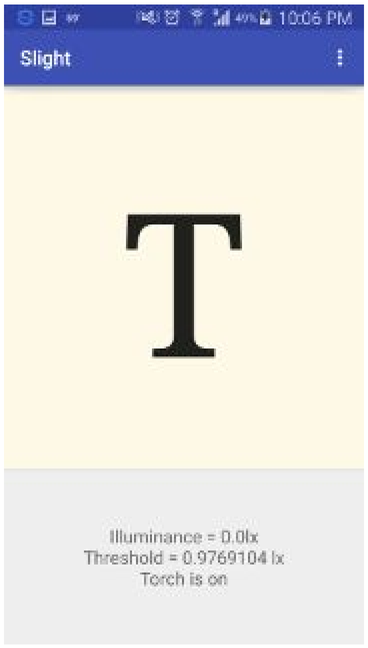
Illuminance = 179.0lx
Threshold = 145.13567 lx
Torch is off

Figure 1: Slight in a bright setting



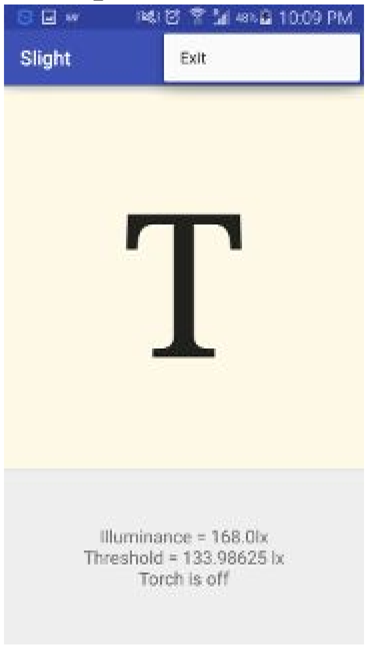
Illuminance = 167.0lx
Threshold = 186.09483 lx
Torch is on

Figure 2: Manually turn on the flash by hitting the Torch button



Illuminance = 0.0lx
Threshold = 0.9769104 lx
Torch is on

Figure 3: Slight in a sudden dark environment



Illuminance = 168.0lx
Threshold = 133.98625 lx
Torch is off

Figure 4: Exit menu item to terminate the app

DESIGN CHOICES

Considered App Ideas

We first considered the idea of an Android application that helps people involved in fire incidents. When the app detects a sudden temperature increase through the ambient temperature sensor, it thinks the user is in a fire, tells the user "help is on the way", and automatically provides 911 with the type of emergency and the incident location. Unfortunately, we stopped working on this application when we found out that our phones do not have the ambient temperature sensor. Besides, we did not have a necessary piece of hardware, the female to female hub, to connect an external temperature sensor to our phones.

Sometimes one wants to check whether one of his friends is in a certain place or not. Therefore we considered making an app that pings the phones of nearby friends to determine their presence. This idea was unfeasible due to the fact that the range of Near Field Communication was limited to only a few centimeters.

The third other idea considered was an app that is useful in activities like camping or hiking where the user is more likely to get lost. When launched, the app should call for help using the camera flash when it is dark or send a distress signal when it is bright. We did not go any further with this idea. However, it gave birth to the current idea.

App Design Journal

App Setup

Android Studio was used as the IDE because it was the official Android IDE. To get our first working (hello-world) app, we looked at an existing code project called android-sensor-logger [1] to study the Android activity lifecycle.

Most smartphones have a light sensor to detect the ambient illuminance. From the android-sensor-logger project, we learned to obtain values from the light sensor by using a SensorManager to register a sensor event listener that provides a new sensor value and executes a method whenever the light sensor reads a different value. The light sensor reports illuminance values in lux (lx), the SI unit for illuminance. At this point, the app successfully obtained ambient illuminance values and logged it in the IDE's console.

Most smartphones also have a bright LED light. This light is usually the camera torch and accessible through the camera's API. Android provides two APIs to access the camera, android.hardware.Camera and android.hardware.camera2. While the former has been deprecated in favor of the latter [2], some say the former still enjoys a larger market share [3], and it is the simpler interface for activating the torch, thus that API was chosen for this application.

To access the camera, the application must also declare intention to do this and be granted the permission to do so. We added uses-feature element with android:required attribute and uses-permission element to AndroidManifest.xml. We then programmed the app to turn on the torch when the illuminance changes to zero and turn it off otherwise. At this point, the app successfully functioned in this manner.

Next, a graphical user interface was created. It contained a large button for explicit user input and a small text view for providing feedback to the user. The button inverted its colors when pressed down. In addition, an option menu item to exit the program was added. At this point, the app successfully additionally displayed button press events and allowed the user to exit gracefully.

Logic for Sensor Input

The app maintained a threshold value below which the torch turned on. It had the meaning of the illuminance below which the user would feel it is too dark to see. At this point, this value was a constant.

Because humans normally get used to the environment, the same ambient illuminance would be either too dark or not too dark depending on how bright the person's previous environment was. This meant that a dynamic threshold would make the app more usable than a static threshold would. We thus made the threshold time-variant to simulate this property. The threshold was now calculated as the average of the last five obtained ambient illuminances.

However, at this point, when it was completely dark and the illuminance is 0, because it was unlikely that illuminance changes at all, the threshold was unlikely to be updated, decaying into a static value. We tried to directly obtain the light sensor's value periodically instead of relying on the asynchronous sensor change event listener. However, some said that it was not possible to do so [4]. We developed a workaround by having the sensor change event listener set a field and creating a timer to get that field's value and update the threshold periodically. The threshold was now a running average of the ambient illuminance.

At this point, we faced another problem. As the threshold was average, the torch shined roughly half of the time. Recall that threshold should mean the "it is too dark to see" value. If a user is used to the current illuminance, their threshold for "too dark to see" will intuitively be somewhat lower. We assumed the latter was a constant value, called the "still not too dark to see" zone, lower than the former. We thus let the threshold value be calculated as the running average ambient illuminance minus a constant.

To improve the user model, we considered that exponential decay into the current ambient illuminance would be a better approximation for brightness acclimation than a running average. Exponential decay can be iteratively calculated by decreasing the difference between the current value and the asymptote value by a fraction of the difference in each iteration. We thus let the threshold be updated periodically as:

```
target_threshold = current_illuminance - still_not_too_dark_zone

threshold = threshold + delta_coefficient * (target_threshold - threshold)
```

Another problem surfaced at this point. When the ambient illuminance was already low, say, 3 lux, the threshold dipped to about -99 lux. Therefore, when it got dark and illuminance level became 0 lux, because the threshold was too low already, the torch did not switch on. One solution was to change the delta coefficient and "still not too dark to see" constants to improve responsiveness. A better solution was to let the "still not too dark to see" zone be a polynomial of the ambient illuminance rather than a constant, which thus meant the target threshold value was a polynomial of the ambient illuminance as well. We tried the latter solution and it improved the application's usability greatly. The equation for calculating the target threshold was now:

```
target_threshold = C_1 * current_illuminance + C_0
```

Logic for User Input

The button the user could press had the function of indicating that the app had incorrectly inferred that the user no longer felt it was too dark to see. In the beginning, it simply caused the torch on-off logic to be inverted. Later, it changed to set the threshold to a certain arbitrarily high value so that the threshold may re-decay when pressed. However, this made little sense just as a constant threshold made little sense. The logic was later changed so that when the button was pressed, the threshold became the ambient illuminance plus a constant value.

One problem faced was that when the environment suddenly became dark or when the button was pressed, the torch did not turn on immediately. This was rooted in the fact that the torch was updated periodically but not when the sensor value changed or when the button was pressed. The solution was to have updateTorch, the method that turns on the torch if the ambient illuminance was below the threshold, be called by the sensor change event listener and by the button touch event listener in addition to the timer. This solved the problem.

References

1. https://bitbucket.org/nonninz/android-sensor-logger
2. https://developer.android.com/reference/android/hardware/Camera.html
3. http://stackoverflow.com/questions/27086078/why-is-camera-deprecated-api
4. https://stackoverflow.com/questions/6433889/getting-light-sensor-value