EE149 Project Final Report: [BLOCK]

Emily Chen, Stephanie Ho, Johnny Shen
EECS149/249A - Fall 2014

Motivation

Time at home and with loved ones should not be spent clutching onto one's phone, waiting for the newest notification to arrive and having to wake the phone screen to find out what they are. We want a product that provides a simple and aesthetically pleasing way of being notified of incoming alerts on our phones so we can choose to respond to only those notifications that we deem important.

Project Vision

[BLOCK] is a simple, unobtrusive at home notification hub so that users can easily tell what new notifications are currently on their mobile device instead of constantly referring to their phone to see them. This project can model the arrival of different notifications and various sensor inputs, from both the smartphone and on [BLOCK] itself, and its reaction to these inputs in a finite state machine. The goal will be to accurately display notifications from your personal linked devices in real-time, interact and control with [BLOCK] through gestures, and adjust the [BLOCK]'s overall color scheme depending on the type of notification to create a visually appealing notification hub for your home.

Figure 1. The finished look of [BLOCK]

The product consists of a laser-cut plywood base which houses the Arduino Mega, BluetoothLE breakout, and half-size breadboard. Inside the white acrylic base, we have a 16x32 RGB LED screen against the front panel and NeoPixels lined along the rest of the inner body to create an ambient glow effect. Along the top, there are ultrasonic sensors to detect swipe gestures.

System Block Diagram

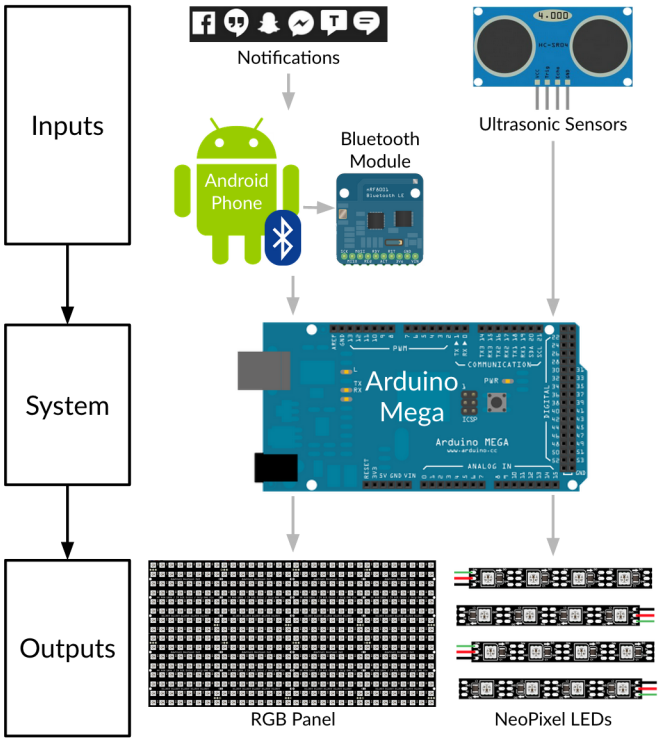

Figure 2. Block diagram showing the inputs and outputs of our system

The inputs of [BLOCK] include incoming notifications to your Android phone, ultrasonic sensor data, and Bluetooth Low Energy (BLE) data packets sent from your Android phone. Further software details are discussed below in the "Software Components" section.

The outputs of [BLOCK] include actuating both the RGB LED Panel Display and the NeoPixel LEDs. Based on the notifications received by your smart phone, the Arduino Mega will change the NeoPixels and RGB display accordingly. For example, if one receives a Facebook notification, the NeoPixels will demonstrate a blue light show to match the colors of Facebook, and the RGB display will read "Facebook", as shown below in Figure 3.

Figure 3. Receiving Facebook and SnapChat notifications

Hardware Components

Table 1 below lists out the hardware components integrated in [BLOCK]. We have categorized the list into components required for [BLOCK]'s inputs, system, outputs, power, and other.

	Components
Inputs	- Ultrasonic sensors HR-SR04 - Bluetooth Low Energy module nRF8001
System	- Arduino Mega 2560
Outputs	- Neopixel LED strip - 16x32 RGB LED matrix panel
Power	- 5V/10A power supply
Other	- Half-size breadboard - 2.1mm female DC power plug - Toggle switch

Table 1. List of Hardware Components

Figure 4 and 5 below show the insides of [BLOCK], and how all the components were housed in the wooden base and acrylic box.

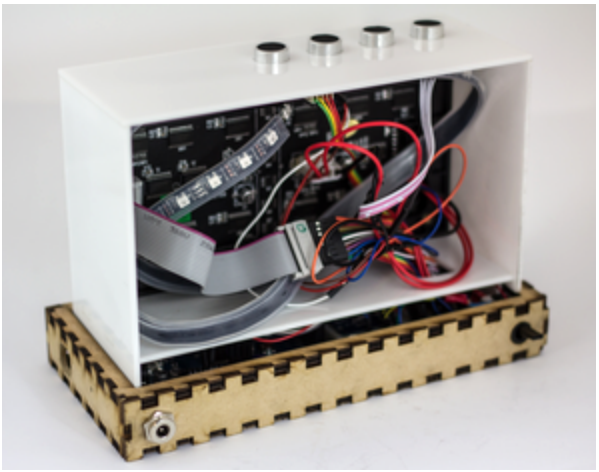

Figure 4. Back view of [BLOCK]; female DC power plug, toggle switch, NeoPixels, and wiring are visible

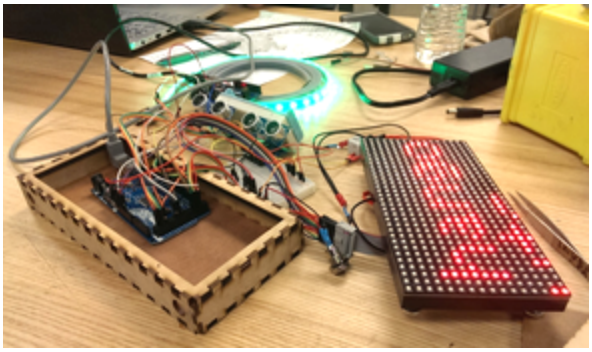

Figure 5. Work in progress - wired hardware components

Software Components

Software components include the development of our Android mobile application and Arduino program that accepts inputs, and actuates our outputs. Figure 6 below exhibits [BLOCK]'s companion application that facilitates the Bluetooth connection and alert transmission.

The Android Application

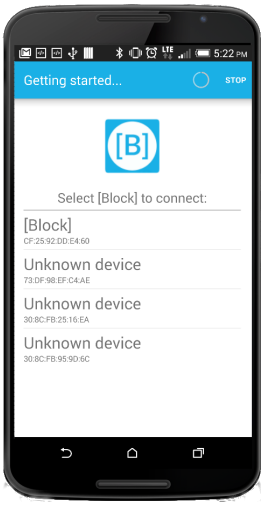

Figure 6. The [BLOCK] Android Application.

System requirements: Because [BLOCK] requires running BluetoothLE and a NotificationListenerService in the background, devices would need to have at least Android 4.3 (API 18+) to be compatible.

Description: Upon launching the Android Application, the user will be prompted to turn on Bluetooth if it is not already on. The app will then scan for available Bluetooth LE devices. Once the user chooses "[Block]", a connection is established, and the user can navigate away from the app if s/he chooses. When a notification is posted to the Android device, it will be pushed to [BLOCK]'s stack of notifications. When the user swipes, selects, or otherwise removes a notification from the Android device's status bar, it will correspondingly be removed from [BLOCK].

The Arduino Program

Description: When the Arduino board is turned on, it sets up and configures all its tools, most notably the BLE chip. Then it enters the main program loop, where it handles gesture detection, updates the displays, and awaits Bluetooth data interrupts. The Finite State Machines in Figures 8-10 give an overview of what the program achieves, though some redundant transitions are left out to make them more readability particularly in Figure 10, which is the largest part of the program.Since the RGB panel and NeoPixel strips independelty look to the global variable `notifs` to grab the details it needs to display that notification, the role of this central part is to update the index counter of the current app to display in the array. Then these light actuators would grab the display information from `notifs[current].name` or `color`.

Performance and Model Analysis

	Performance
Notification receive time	~1000ms
Bluetooth connection speed	~200ms
Ultrasonic sensor response time per reading	~20 - 30 ms
Minimum time to recognize gesture	4 readings per gesture = ~80ms
Ultrasonic sensor range	~200 mm

Table 2. Performance Statistics

Reliable Real-time Behavior

From the user's experience, Bluetooth connection and data transfer between the Android device and [BLOCK] are extremeley fast, with little to no discernable latency between the Android device receiving the notification update to it appearing on [BLOCK]. Possible delays that may occur for notifications would arise from Android device's data connection, depending on how quickly the network pushes the notification to the phone. As shown in Table 2, the performance speeds for all of the time-related parameters are in the milliseconds, demonstrating fast and reliable real-time behavior of the system.

Ultrasonic Sensors

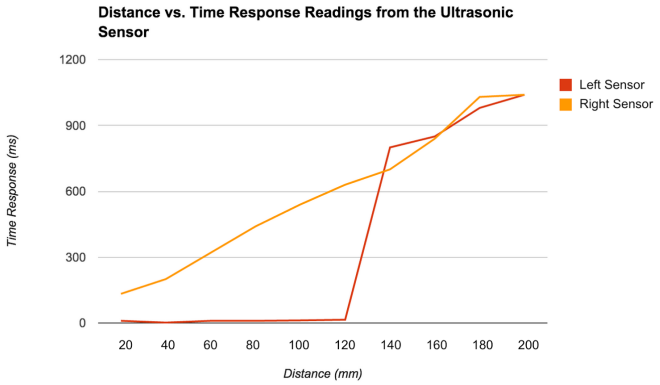

Figure 7. Average ultrasonic sensor response times observed by holding a hand between the sensors.

Upon initial inspection of the ultrasonic sensor reading values, one may be confused as to why the left sensor seems broken or unresponsive at close range (120mm). Given that the ultrasonic sensors work by emitting a high signal and then timing the delay in receiving its response, there are several possible interpretations to explain these abnormally low but nonzero readings.

The most probable explanation is that since the right sensor is sampled before the left, the immediate responses read from the left sensors could be residue signal from the right sensor's pulse out that reflected off our experimental surface and returned to the sensor. This would explain the left sensor's recovery to perform normally at a more reasonable range, as it leaves enough time for the residue noise to dissipate.

To remedy this problem, rather than spread the sensors further apart which would force the user to swipe across a wider range in an unnatural way, we adapted our gesture detection algorithm to ignore the depths of readings and simply consider the flow of motion acorss the left-right axis by translating the response times as booleans indicating whether the sensor has detected an object within a threshold. Final conclusions on the direction of that object would be based on its entrance and exit; that is, only the first and last readings that have detected an object are considered while everything in between is interpreted as intermediate motion. Given the high sampling rate of the sensors, if an object is entering from a certain side, the field of view or range of the sensors is able to pick it up without fear of reflection noise.

For example, for a left to right swipe, our algorithm would look for a pattern like (1, 0) !(0, 0)* (0, 1), where each pair is the (left, right) response time and the middle pair indicating any number of readings where both sensors are not zero simultaneously are accepted. This negates the effects of noise and results in accurate readings 95% of the time with fast (< 1s) swipe speeds.

Finite State Machines

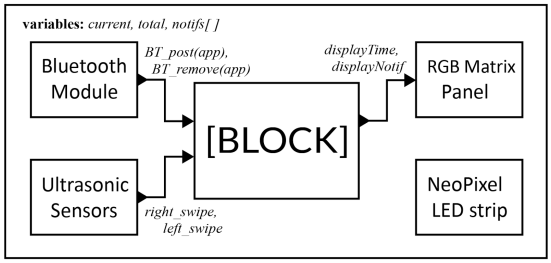

Figure 8. Relations between the various components.

Here is a high level overview of the logic flow of the components that make up [BLOCK]. The global variable of the array containing accumulated notifications is the core constituent of the program, as it is what tracks which notifications are active. The current and total variables help navigate between the active notifications, and are what the actuators (RGB panel and LED strip) use to display alert information.

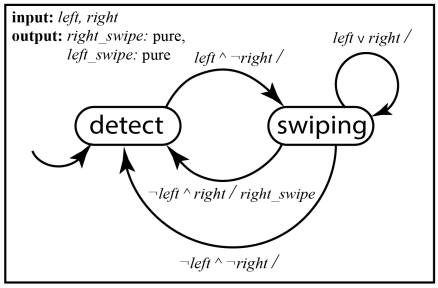

Figure 9. FSM of right swipe detection transitions.

Figure 9 models how [BLOCK] will respond to swipe gestures, specifically right gestures (for ease of understanding). When a right swipe is initialized or "detected" from the left sensor, it will enter a "swiping" mode. If it detects an exit motion on its companion sensor, it will either register a "right_swipe". If no exit motion is detected or if motion is detected on the originating sensor, it will reset itself.

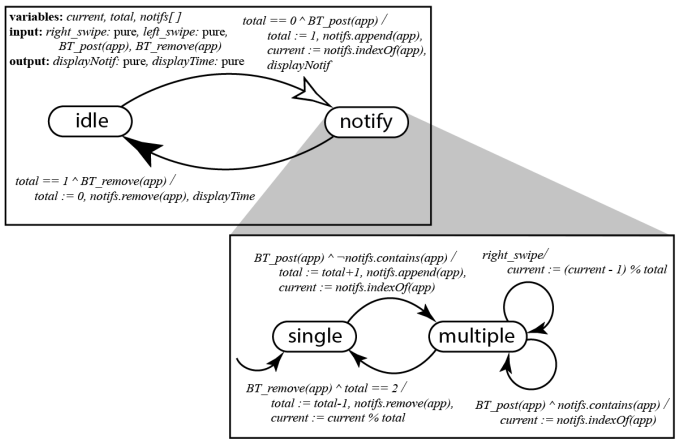

Figure 10. FSM of main [BLOCK] logic updating global variables for the actuators (some transitions omitted)

The figure above models [BLOCK] behavior at a higher level. When there are no notifications in its stack, the product will enter an idle state. Upon receiving a notification or swipe gesture, [BLOCK] enters a "notify" state. Inside, there are two substates, single or multiple. In single mode, it will either wait for a notification "post" or "remove". On a "remove", it will revert back to idle state. On "post", it will push the new notification onto the stack and also wait for swipes in addition to additional post/removes.

Future Improvements

- *Ticker text* : In addition to notifying users of the apps from which there are notifications, we can also display the content of the particular notification.
- *PCB*: Currently, the Arduino Mega, breadboard, and wiring are all hidden away inside the wooden platform that BLOCK rests on. Using a PCB can allow us to do away with the base and create a sleeker-looking product.
- *Informative actuators*: We can add additional actuators such as a microphone or speaker to handle voice calls or notification readouts through [BLOCK], further reducing the need for one to physically pick up the phone while connected.
- *User modes*: Can allow the user to choose between simple notifications mode (as is) or verbose mode where notification ticker text is also displayed.

Product Introduction:
Project video demonstrating [BLOCK] in action:
http://youtu.be/QS4oOv85tpQ

References

1. http://www.kpbird.com/2013/07/android-notificationlistenerservice.html
2. https://developer.android.com/guide/topics/connectivity/bluetooth-le.html
3. https://github.com/adafruit/Adafruit_nRF8001
4. https://github.com/adafruit/Adafruit_NeoPixel
5. https://github.com/adafruit/Adafruit-GFX-Library