Using Mobile Devices and Visual Programming to Introduce IoT in the Classroom

1st Devin Jean
Institute for Software Integrated Studies
Vanderbilt University
Nashville, Tennessee, USA
devin.c.jean@vanderbilt.edu

2nd Ákos Lédeczi
Institute for Software Integrated Studies
Vanderbilt University
Nashville, Tennessee, USA
akos.ledeczi@vanderbilt.edu

***Abstract*—With the emergence of Cyber Physical Systems (CPS), networked sensing and actuation are all around us. Yet current computer science and engineering education in K-12 do not typically cover them due to various barriers. However, one project which attempts to solve these problems is NetsBlox, which provides a block-based interface to allow students to write distributed programs in a way that abstracts away much of the underlying complexity while still allowing students to learn the most important concepts. We have created an open-source phone app called PhoneIoT that connects to NetsBlox and gives access to many of the sensors of the mobile device. Once connected, students are able to use the block-based NetsBlox environment to communicate with the device and retrieve live sensor data. The user interface on the phone can be also configured through the student's NetsBlox project which, in turn, sends notifications that the students can process by writing event-handlers. The showcase will demonstrate a number of compelling applications of PhoneIoT, such as using the phone's accelerometer to control a game running on the student's computer by tilting the phone or to implement a compass.**

I. Introduction

Not all schools offer makerspaces and other opportunities for students to get their hands on simple embedded computers, sensors and educational robots. Also, the kind of sensors and devices available are limited by cost. Finally, these kinds of activities are not available for remote instruction. However, mobile devices that most students have contain a rich collection of sensors that are connected to the internet out of the box. This presents an opportunity to teach concepts related to IoT, networking and distributed computing in a manner that is not only accessible to novices but also highly engaging and motivating. To make this approach a reality, we have created PhoneIoT, a mobile app for both Android and iOS which allows the built-in sensors of the device to be accessed remotely from NetsBlox [1], an educational block-based programming environment which focuses on networking and distributed computing [2]. Since these devices have touch-screens as well, PhoneIoT makes it possible to configure a graphical user interface on the phone from the same NetsBlox program that processes the sensor data and handles events from the mobile device. Hence, students can build truly distributed applications that run on two or more computers connected via the internet and that interact with the physical world via sensors.

II. Previous Work

There are several projects which are similar to PhoneIoT, such as App Inventor [3], Kodular [4], and Thunkable [5]. Although these tools give access to read and use live sensor data, they are meant for completely different purposes: these projects are used to create apps that run locally on a smartphone, rather than in a distributed system like PhoneIoT, which makes them less useful for teaching IoT. Although some of these projects, like Thunkable, could support students manually adding networking logic through direct API requests, this would add complexity which is not present in PhoneIoT. Additionally, a unique benefit of PhoneIoT is being directly integrated into NetsBlox, which has many other services available through the same block-based interface.

III. Design Choices

PhoneIoT was designed to be as simple as possible while still providing common IoT features like on-demand sensor requests and continuous data streaming. The need for simplicity for novice programmers prompted the use of a block-based system as the primary method of interaction, and NetsBlox was a perfect candidate to support it due to its focus on distributed computing. After starting the app on a mobile device and connecting to the NetsBlox server (a single button press), all that is needed to connect to the device from NetsBlox is the ID and password shown in the app's menu. To make things easier for individual users, the ID for any device is fixed. The password is a 31-bit hexadecimal integer; this can be re-randomized through the app menu, but by default it is set to zero for convenience. For security reasons, especially due to providing access to, e.g., live location data, the password (even the default zero password) expires after 24 hours, effectively cutting active connections.

NetsBlox uses two distributed computing abstractions: remote procedure calls (RPCs) and message passing introducing only three new blocks: "call", "send msg" and "when I receive msg." To minimize the learning curve for PhoneIoT, we use these same techniques for communicating with mobile devices. For instance, there are RPCs like "getAccelerometer" or "getLocation" for requesting on-demand sensor data. There are also RPCs such as "listenToSensors" which send a stream of sensor updates in the form of messages. An example of

requesting and receiving a stream of accelerometer updates is given in Figure 1.

After an introduction to the basic ideas behind PhoneIoT, the showcase will provide live demonstrations of various example projects using PhoneIoT. We list here a couple of examples.

IV. Example Project: Tilting Game

To introduce students to PhoneIoT, we begin by showing how to use the dedicated RPCs for accessing on-demand sensor data, for example, using the "getAccelerometer" RPC. This returns the current value of the sensor. To get continuous updates, we can use streaming access methods with the "listenToSensors" RPC. This method is simpler due to not requiring any error checking, and is a common technique used for networked sensors. Figure 1 gives all the code required to connect to the device and request/receive updates from the accelerometer every 100ms.

To make the project more engaging for K-12 audiences, we make this into a game where there is a target the ball needs to reach while avoiding holes that it can fall into. For simplicity, we use the x and y components of acceleration to control the velocity of a ball which rolls around on the NetsBlox stage (the actual movement logic is omitted for brevity). From a coding perspective, these parts are separate from PhoneIoT, and so should be familiar concepts to all NetsBlox and Snap! users. Figure 2 shows the NetsBlox stage when running the completed project.

This project is meant to be an easy starter project, as it uses only the simplest features of PhoneIoT. However, it already serves to introduce students to distributed computing concepts like message passing, as well as more unique IoT topics like live data streaming. There are several additions that students can make, such as having sensor- or GUI-based methods for resetting the game, having the accelerometer readings actually accelerate the ball instead of setting the velocity, or even handling elastic collisions with the stage walls. The acceleration feature could be done by simply adding the (scaled) acceleration to velocity, which introduces students to basic integral controllers. This project only makes use of sensor data from the device; in the next project, we will show a very simple example of controlling the GUI display.

V. Example Project: Compass App

One of the sensors supported by PhoneIoT is called the "orientation" sensor, which measures the device's angular offset from a fixed geomagnetic reference frame; this is returned as a 3D vector. The first component of the vector is the offset from magnetic north, effectively a compass heading. We will now use this sensor data to implement a compass app which reads orientation data from the device and writes it back to the device display via custom label text (and rotates a sprite on the stage like a compass needle). Figure 3 shows the code needed to create the label on the device and update its content with live compass heading information. Figure 4 shows the PhoneIoT app display when pointing in two different directions; the code in Figure 3 has it show the heading and ordinal direction (i.e. N, NE, E, etc.).

Like the tilting game project, this is meant to be a short project for students to be introduced to PhoneIoT. From this project, a number of more interesting components can be added as students become more comfortable with PhoneIoT and NetsBlox. For instance, one could make use of the GoogleMaps service, a pre-existing service in NetsBlox, to get a map of the area based on location data, and have a rotating compass sprite in the corner. Students could even move a sprite around on the map based on live location data and rotate it around based the the compass heading.

VI. Example Project: GPS Tracker

The NetsBlox platform already supports many online services, one of which is Google Maps. With this service, a program can get and display a map of the current location, specified by latitude and longitude, or get the screen position of a latitude and longitude point on the map and vice versa. By reading live GPS data from a mobile device running PhoneIoT, it is possible to track the location of the device on a map and use NetsBlox's built-in drawing utilities (inherited from Snap!) to plot the course. Thus far, this has all been performed on the NetsBlox client (for user logic and drawing) and the NetsBlox server (for performing API requests), with the device running PhoneIoT only being used as a sensor. However, by using PhoneIoT's custom GUI elements, we can add an image display and send periodic updates to the mobile device. Essentially, this creates a form of stripped-down Google Maps front-end that can be built in under ten blocks. See Fig 5 for the update logic running on NetsBlox and Fig 6 (b) for the custom GUI shown on the mobile app.

VII. Conclusion

As emphasized by the example projects, PhoneIoT provides an extremely simple interface for interacting with live sensor data, to the point that some basic projects only require a few blocks. However, PhoneIoT has many more features, ranging from more specialized sensors like the gyroscope and microphone level sensors, to advanced custom controls like image displays, text boxes, and buttons, which were omitted to focus on low-learning-curve student projects. Due to PhoneIoT's simplicity, it is a good way to introduce students to IoT concepts, and the more advanced features serve as motivations for more involved and engaging projects as students learn the core concepts.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1835874, the National Security Agency (H98230-18-D-0010) and the Computational Thinking and Learning Initiative of Vanderbilt University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] "Netsblox website," https://netsblox.org, 2021, cited 2021 March 8.

[2] Broll, B and Lu, M and Ledeczi, A and et al., "A Visual Programming Environment for Learning Distributed Programming," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, Mar 2017, pp. 81–86. [Online]. Available: 10.1145/3017680.3017741

[3] "MIT App Inventor," https://appinventor.mit.edu/.

[4] "Kodular," https://www.kodular.io/.

[5] D. Siegle, "There's an App for That, and I Made It," *Gifted Child Today*, vol. 43, pp. 64–71, Jan 2020.

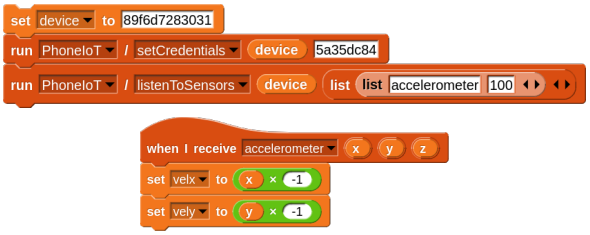


Fig. 1. Code to connect to a device and listen for accelerometer updates. This uses an event-based system where we register to receive updates every 100ms and execute a handler as we receive them.

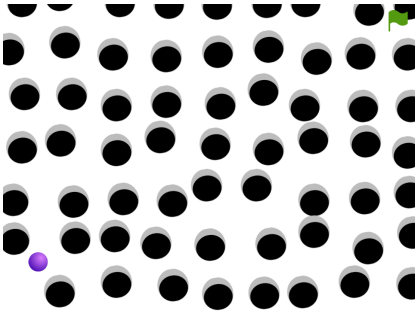


Fig. 2. The NetsBlox stage when running the tilting game example project.

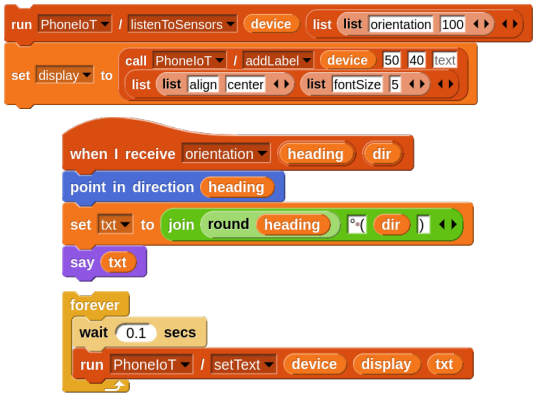


Fig. 3. Code to implement a compass app. The first code segment registers the project to receive updates from the orientation sensor every 100ms. The second receives them and updates the display on the NetsBlox stage. The third updates the display on the device; this is separate because "setText" is synchronous and would otherwise slow down the message handler.

(a) (b)

Fig. 4. The PhoneIoT app display for the compass example project (running on an iPad).

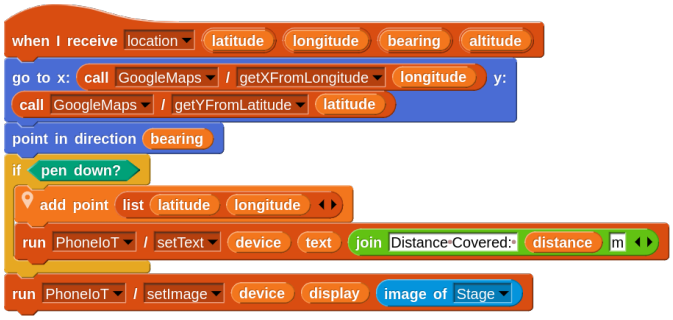
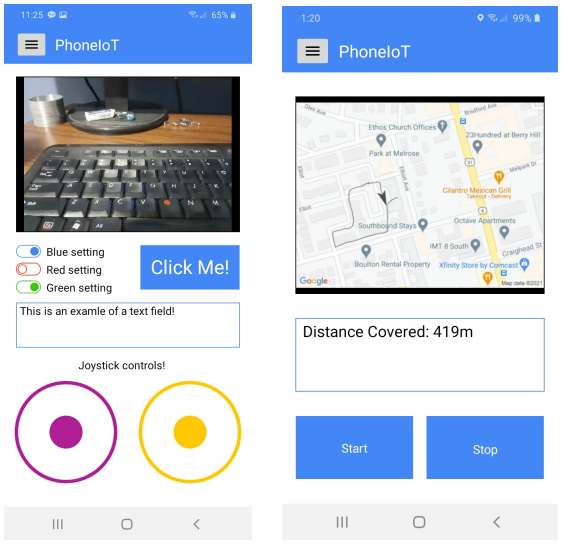


Fig. 5. Code to read live GPS data from the mobile device, plot it on a map, and update the device display. Converting the latitude and longitude into a screen location is provided by a separate GoogleMaps service. PhoneIoT is used to connect these together and update the remote display, as well as to provide the actual location data.



(a) (b)

Fig. 6. The PhoneIoT app display with various custom controls (running on Android phones). Image (a) shows examples of supported controls. Image (b) shows the end result of the GPS Tracker example project.