Your Phone as a Sensor: Making IoT Accessible for Novice Programmers

Devin Jean
Vanderbilt University
Nashville, USA
devin.c.jean@vanderbilt.edu

Brian Broll
Vanderbilt University
Nashville, USA
brian.broll@vanderbilt.edu

Gordon Stein
Vanderbilt University
Nashville, USA
gordon.stein@vanderbilt.edu

Ákos Lédeczi
Vanderbilt University
Nashville, USA
akos.ledeczi@vanderbilt.edu

***Abstract*—Distributed computing, computer networking, and the Internet of Things are all around us, yet only computer science and engineering majors learn the technologies that enable our modern lives. This paper introduces PhoneIoT, a mobile app that makes it possible to teach some of the basic concepts of distributed computation and networked sensing to novices. PhoneIoT turns mobile phones and tablets into IoT devices and makes it possible to create highly engaging projects through NetsBlox, an open-source block-based programming environment focused on teaching distributed computing at the high school level. PhoneIoT lets NetsBlox programs—running in the browser on the student's computer—access available sensors. Since phones have touchscreens, PhoneIoT also allows building a GUI remotely from NetsBlox, which can be set to trigger custom code written by the student via NetsBlox's message system. The approach enables students to create quite advanced distributed projects, such as turning their phone into a game controller or tracking their exercise on top of an interactive Google Maps background with just a few blocks of code.**

***Index Terms*—IoT, Mobile Devices, Sensors, User Interaction, Block-Based Programming**

I. Introduction

Most of the applications we use on our computers and mobile devices every day are distributed and use the Internet to provide their functionality. Networked sensors and actuators—the Internet of Things (IoT)—are also becoming ubiquitous, with smart homes and health monitoring leading the way. Yet hardly any of the enabling technologies are taught in introductory computer science classes in K-12. There do exist classes and makerspaces where some students are exposed to embedded computers, providing opportunities to program Raspberry Pis or micro:bits with simple sensors and actuators, such as LEDs, using connectivity based on either a USB cable or Bluetooth. While these experiences are fun, they are fairly disconnected from the IoT that otherwise surrounds us. In addition, not many schools offer these types of classes due to cost, logistics, and a lack of teachers.

84% of teenagers in the United States, however, already own a device [1] that comes with a rich set of powerful sensors, including accelerometers, gyroscopes, microphones, cameras, GPS, and many more, and is Internet-enabled out of the box. Smartphones offer an excellent opportunity to expose students to networked sensing and make computing more engaging by enabling them to be creators and not just users of compelling applications. But how can we make these powerful technologies accessible to novice programmers?

Supported by the National Science Foundation (#1835874) and the National Security Agency (H98230-18-D-0010).

We introduce a mobile app, PhoneIoT, to programmatically access smartphones and tablets as IoT devices through NetsBlox [2], a block-based programming environment based on Snap*!* [3]. NetsBlox introduced two powerful networking and distributed computing abstractions to block-based languages. Remote Procedure Calls (RPCs) make a rich set of online services and data sources accessible to student programs, such as Google Maps, gnuplot, earthquake data from USGS, climate change datasets from NOAA, the Open Movie database, and many more [4]. Message passing lets NetsBlox projects running anywhere in the world communicate with one another, making it possible to create multi-player games and other distributed programs. RPCs and messages are also used to control WiFi-enabled devices such as educational robot vehicles [5]. PhoneIoT utilizes these very same abstractions, so any user familiar with NetsBlox will already have the necessary knowledge to create projects using PhoneIoT.

PhoneIoT provides two main features: the ability to read and/or stream live sensor data, and the ability to create an interactive, configurable user interface on a mobile device. Both of these features are accessible through RPCs and messages, allowing for the creation of fun and engaging projects that integrate sensor data and custom user input and provide feedback on the device's display. By using a simple, yet powerful, block-based interface, we abstract away much of the complexity of networking and distributed computing while allowing students to explore and learn the most important concepts in a convenient framework.

II. Previous Work

There are several approaches, including Thunkable, App Inventor, and Kodular (formerly known as AppyBuilder), that allow for the creation of standalone mobile apps that can be constructed online with a block-based programming interface [6]–[8]. Pocket Code (part of the Catrobat project) is similar to these, although its app designer is built into the app itself and is more focused on creating games or simulations [9]. Thunkable is perhaps the most similar to PhoneIoT, as it allows access to Internet resources (e.g., speech recognition and translation services), as well as several device sensors, such as the accelerometer and gyroscope. However, PhoneIoT is fundamentally different from these projects in that it does not aim to be an app creation tool; rather, the custom controls in PhoneIoT are merely a means of interacting with NetsBlox code running in the browser on the student's computer. That is, Thunkable and similar projects are not tools for teaching distributed computing or IoT, as all user interaction and sensor data is kept internal to the device running the app. Additionally, because PhoneIoT is tailored to a distributed computing environment, it offers more possibilities for creating engaging educational projects. For instance, PhoneIoT could be used to turn a phone into a custom game controller, with accelerometer input and soft buttons on the phone's screen making sprites move or shoot on the computer's screen. The phone could also be used to control real robots in the same way, using a single NetsBlox program to control multiple computers: one or more mobile devices, one or more robots, and the laptop, creating an engaging distributed application.

Another project similar to PhoneIoT in terms of intent and network architecture is Sensor Fusion, an education-focused project which collects sensor data from a mobile device and streams it to a computer for analysis [10]. This is similar to the core sensor-based functionality of PhoneIoT but is more heavily focused on a scientific perspective, namely *sensor fusion*, or the combination of data from multiple sensors to achieve greater accuracy or precision than would otherwise be possible. In contrast, PhoneIoT is part of a distributed computing environment, empowering students to utilize incoming live data streams, as well as content from other NetsBlox services, and reconfigure the phone's display in real time based on the desired application (e.g., a game controller, data viewer, or fitness tracker). This is not possible with Sensor Fusion, as its display and interactive components are not configurable. PhoneIoT's programmability is a key factor in creating engaging educational projects for young learners.

III. PhoneIoT

Mobile devices come with a wide variety of hardware sensors, from simple accelerometers and gyroscopes to more specialized sensors like step counters and relative humidity detectors. Although a typical device does not contain all potential sensors, there are several sensors that are reliably present even on older devices, simply due to basic system requirements. These include an accelerometer (used for landscape-portrait rotation) and, for smartphones, a microphone and proximity sensor (to disable the touchscreen when held to a user's ear). While not essential for core device functionality, virtually all modern smartphones and tablets also have a camera, although access to this sensor through PhoneIoT is handled differently due to privacy concerns (see below). Additionally, through services such as Google's Fused Location Provider API, any mobile device connected to the Internet can retrieve live location data, if not by GPS, then by estimating it from network connectivity. PhoneIoT continuously collects and streams data from all available sensors to the NetsBlox server, in addition to handling other specialized requests, such as GUI configuration and interactions. Figure 1 visualizes this system architecture.

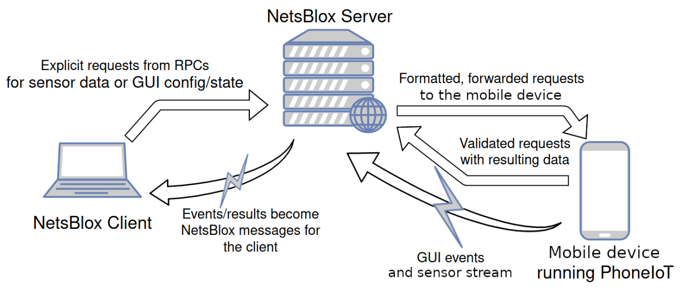

NetsBlox Server
Explicit requests from RPCs for sensor data or GUI config/state
Formatted, forwarded requests to the mobile device
Validated requests with resulting data
NetsBlox Client
Events/results become NetsBlox messages for the client
GUI events and sensor stream
Mobile device running PhoneIoT

Fig. 1. A visualization of backend network interactions between a NetsBlox client, the NetsBlox server, and a device running PhoneIoT.

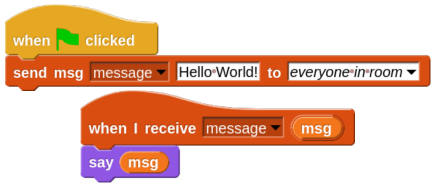
PhoneIoT raises some potential privacy issues due to accessing live sensor data, including microphone, camera, and location information. To address these concerns, PhoneIoT

- Exposes only volume levels and not actual audio samples,
- Allows camera access only through direct user approval (the user must actively take the image), and
- Secures location (and other) data by requiring a randomly generated password that expires after one day.

Additionally, unless the "run in background" setting is explicitly opted into, no communication is permitted unless the app is open and active in the foreground.

A. Network Exchanges

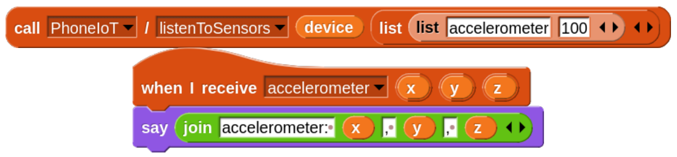
When the PhoneIoT app is started, it connects to the NetsBlox server, announces its presence, provides the server with its unique identifier for further communications, and begins streaming sensor data and accepting network requests via UDP. This protocol was selected because our data exchange model is already packet-based, making UDP a better model than streaming protocols such as TCP. Although UDP has the potential issue of dropping packets, for our purposes, this is actually desirable due to providing real-world lessons on error-handling in fallible network transactions. For instance, an early project for students could be to make robust wrappers for some PhoneIoT functions by repeating the operation until it succeeds.



```
when [flag] clicked
send msg message Hello World! to everyone in room
when I receive message msg
say msg
```

Fig. 2. Example of sending and receiving messages in NetsBlox.

The networking primitives used by the NetsBlox side of PhoneIoT are composed of "messages," the same concept used throughout NetsBlox. In essence, a "message" is a structured block of data that is identified by name and has a set of fields associated with it. Messages can be sent with the "send msg" block and received (typically on a different computer) with a "when I receive" block. As an example, there is a default message type called "message" which has a single field called "msg." Figure 2 shows a simple example of how to send and receive a message of this type.



```
call PhoneIoT / listenToSensors device list list accelerometer 100
when I receive accelerometer x y z
say join accelerometer: x , y , z
```

Fig. 3. Registering for and receiving accelerometer updates at 10 Hz.

PhoneIoT provides two primary ways of accessing sensor data: either through explicit requests, or by registering a sensor update event. Explicit requests are done with normal RPC return values, while sensor update events are received by a custom message type. If an explicit request does not receive a response from the phone (e.g, due to a dropped packet), an error is returned to the user, which would have to be checked. Because of this, the streaming method is simpler for users as sensor data are sent asynchronously from the phone, and a dropped packet simply results in a skipped message. Figure 3 shows example code which registers for and receives sensor updates from the accelerometer every 100ms.

B. *Custom GUI Controls*

An important feature of PhoneIoT is its customizable interactive display. The static GUI for the main screen of the PhoneIoT app is rather barren, showing only a button to access the app main menu and a canvas for rendering custom controls. This canvas is initially empty, but it can be populated with content via various RPCs from the user's program. PhoneIoT supports many standard GUI control types, such as labels, buttons, text fields, image displays, and toggle switches, as well as some controls tailored for designing game controllers, such as virtual joysticks. Each of these controls is fully customizable as to text content, location, size, color, and many other options depending on the specific control. A non-exhaustive example of custom controls is given in Figure 4 (a) and (b).

Most custom controls have some form of event associated with them, which is sent to the server each time it is raised in the form of a message. For instance, buttons send a message when they are pressed, text fields and image displays send a message when a user updates the content (e.g., via the camera), and joystick controls send a message each time the stick is moved. Additionally, there are other RPC requests which access state information about the controls, such as their text or toggle state. As an example, a student could perform some action every second while a button is held down.

This interactive component is important for teaching IoT to younger K-12 audiences because it immediately gives the students a useful tool related to things they already know, such as game controllers or content sharing with text/image displays. Due to how important phones are to today's youth, introducing them to new ways of engaging with and controlling their devices can be especially motivating for continued interest in CS topics. The networking and IoT components are added to this to provide even more functionality and to teach the concepts to an already eager audience as a "side effect."

IV. Example Projects

This section will cover several example projects to demonstrate how phone-based IoT through the NetsBlox platform enables powerful applications with very little code or specialized knowledge.

A. *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 stripped-down form of the Google Maps front-end that can be built in under ten blocks. See Figure 5 for the blocks that set up the display, Figure 6 for the update logic running on NetsBlox, and Figure 4 (b) for the custom GUI shown on the mobile app.

B. *Accelerometer Plotter*

A common topic in introductory IoT is analyzing a live data stream coming from a device. We have already seen that receiving a data stream from PhoneIoT is as simple as one RPC call and listening for a NetsBlox message. Once we have this data, the student can perform whatever analysis is needed and output results to their NetsBlox client display. A simple project that could be conducted on a student's first day of working with PhoneIoT is to receive live accelerometer data and plot its x, y, and z components.

This can be done with the Chart service, a pre-existing NetsBlox service for generating graphs from data points. Figure 7 shows the code required to do this, and Figure 4 (c) shows the NetsBlox display (running on the student's computer) after the phone was picked up from rest, rotated slowly around one axis, then another, and finally dropped onto a pillow.

V. Conclusion

In this brief overview, we have shown that PhoneIoT is a low-cost method for allowing K-12 students to access device sensors for learning the concepts of networked sensing. These concepts include API requests in fallible conditions, methods for handling failures, sensor data processing, event based

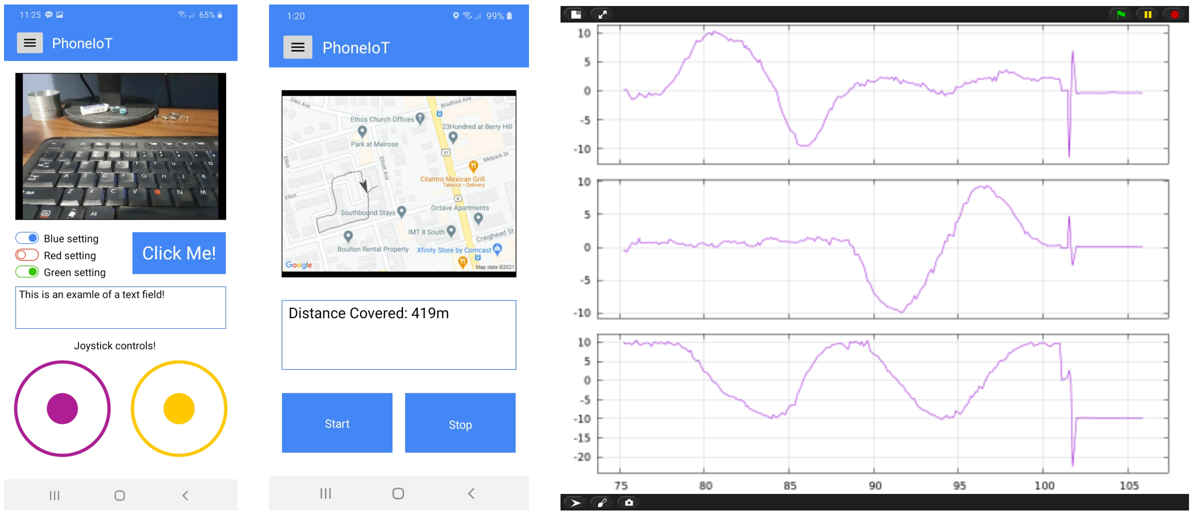
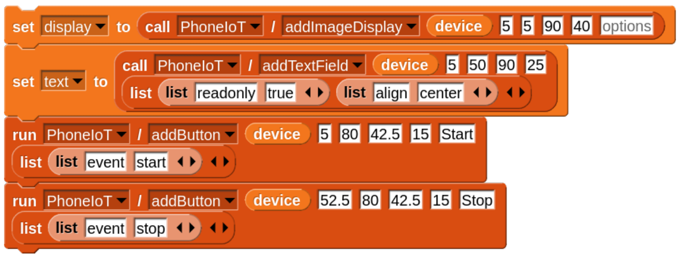
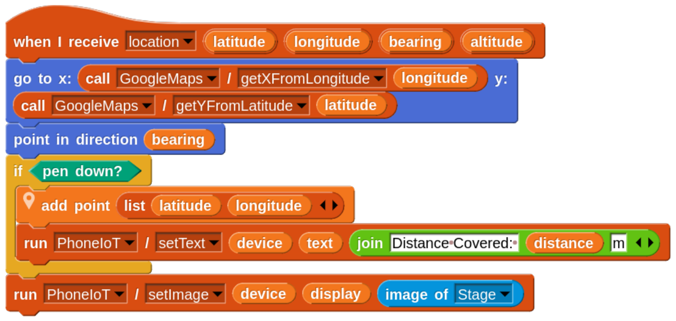


Fig. 4. Two PhoneIoT apps with custom controls (a and b). Streaming acceleration data plotted on the stage of the example project in Section IV-B.



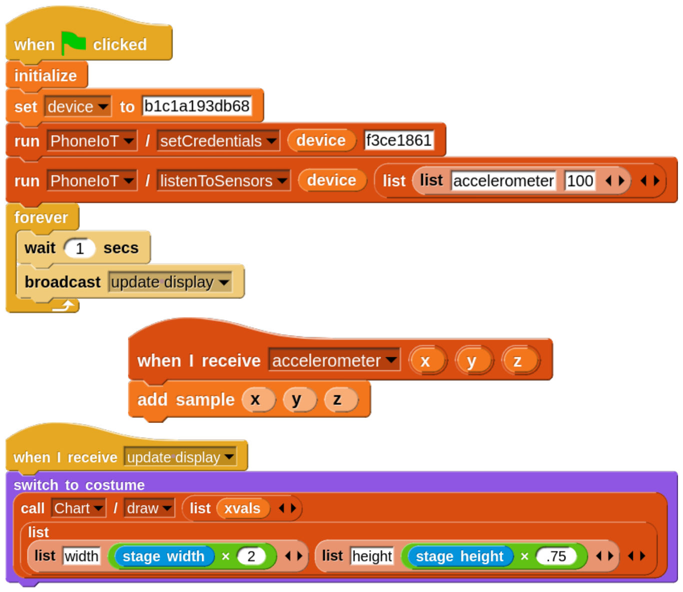
```
set display to call PhoneIoT / addImageDisplay device 5 5 90 40 options
set text to call PhoneIoT / addTextField device 5 50 90 25
            list list readonly true ◂▸ list align center ◂▸ ◂▸
run PhoneIoT / addButton device 5 80 42.5 15 Start
  list list event start ◂▸ ◂▸
run PhoneIoT / addButton device 52.5 80 42.5 15 Stop
  list list event stop ◂▸ ◂▸
```

Fig. 5. Blocks used to set up the PhoneIoT device display for the GPS tracker.



```
when I receive location latitude longitude bearing altitude
go to x: call GoogleMaps / getXFromLongitude longitude y:
  call GoogleMaps / getYFromLatitude latitude
point in direction bearing
if pen down?
  add point list latitude longitude ◂▸
  run PhoneIoT / setText device text join Distance Covered: distance m ◂▸
run PhoneIoT / setImage device display image of Stage
```

Fig. 6. Location message handler. It reads live GPS data from the mobile device, plots the track on a map on the stage, and sends the map/track back as an image to the device. Note that the "add point" custom block maintains the list of locations and computes the distance using a Google Maps RPC.



```
when clicked
initialize
set device to b1c1a193db68
run PhoneIoT / setCredentials device f3ce1861
run PhoneIoT / listenToSensors device list list accelerometer 100 ◂▸ ◂▸
forever
  wait 1 secs
  broadcast update display

when I receive accelerometer x y z
add sample x y z

when I receive update display
switch to costume
  call Chart / draw list xvals ◂▸
    list
      list width stage width × 2 ◂▸ list height stage height × .75 ◂▸ ◂▸
```

Fig. 7. Code running the accelerometer plotter project. The "add sample" custom block maintains three lists (xvals, yvals, zvals). The "update display" script is only shown for the x sprite.

programming via message passing, and potentially many other topics depending on usage. Additionally, the custom display on the phone allows students to come up with novel ways to interact with their code running on NetsBlox. We believe students will find PhoneIoT an enjoyable educational tool that will allow them to envision and create innovative distributed applications. Along the way, they will learn important cutting edge computing concepts rarely taught in K-12 today.

The PhoneIoT app is already available on both Android and iOS. Our future work includes creating curricular units around it, with some of the relevant educational topics having been mentioned in this paper. As a test of learning efficacy and student engagement, we intend to introduce PhoneIoT into our K-12 summer camps and collect data from real users. The current curriculum is project-based and focuses on having students learn the basic concepts and features, then apply them in creative individual or group projects. We will also be covering several other NetsBlox services and advanced CS topics during the camp, giving the students plenty of interesting possibilities for integration with PhoneIoT's sensor data and input controls.

References

[1] "It's a smartphone life: More than half of u.s. children now have one," Oct 2019. [Online]. Available: https://www.npr.org/2019/10/31/774838891/its-a-smartphone-life-more-than-half-of-u-s-children-now-have-one

[2] "NetsBlox website," https://netsblox.org, cited 2021 April 15.

[3] "Snap!: a visual, drag-and-drop programming language," http://snap.berkeley.edu/snapsource/snap.html.

[4] B. Broll, M. Lu, A. Ledeczi, 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.

[5] Á. Lédeczi, M. Metelko, X. Koutsoukos, G. Biswas, M. Maróti, H. Zare, B. Yett, N. Hutchins, B. Broll, P. Völgyesi, M. B. Smith, and T. Darrah, "Teaching Cybersecurity with Networked Robots," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, 2019, pp. 885–891.

[6] D. Siegle, "There's an app for that, and i made it," *Gifted Child Today*, vol. 43, pp. 64–71, Jan 2020.

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

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

[9] W. Slany, "Pocket code: a scratch-like integrated development environment for your phone," in *Proceeding of the companion publication of the 2014 ACM SIGPLAN conference on Systems, Programming, and Applications: Software for Humanity*. ACM, Oct 2014, pp. 35–36.

[10] G. Hendeby, F. Gustafsson, N. Wahlström, and S. Gunnarsson, "Platform for teaching sensor fusion using a smartphone," *International Journal of Engineering Education*, vol. 33, pp. 781–789, Apr 2017.