LegoSENSE: An Open and Modular Sensing Platform for Rapidly-Deployable IoT Applications

Minghui Zhao, Stephen Xia, Jingping Nie, Kaiyuan Hou, Avik Dhupar, Xiaofan Jiang
Columbia University
USA, {mz2866, stephen.xia, jn2551, kh3119, ad3910}@columbia.edu, jiang@ee.columbia.edu

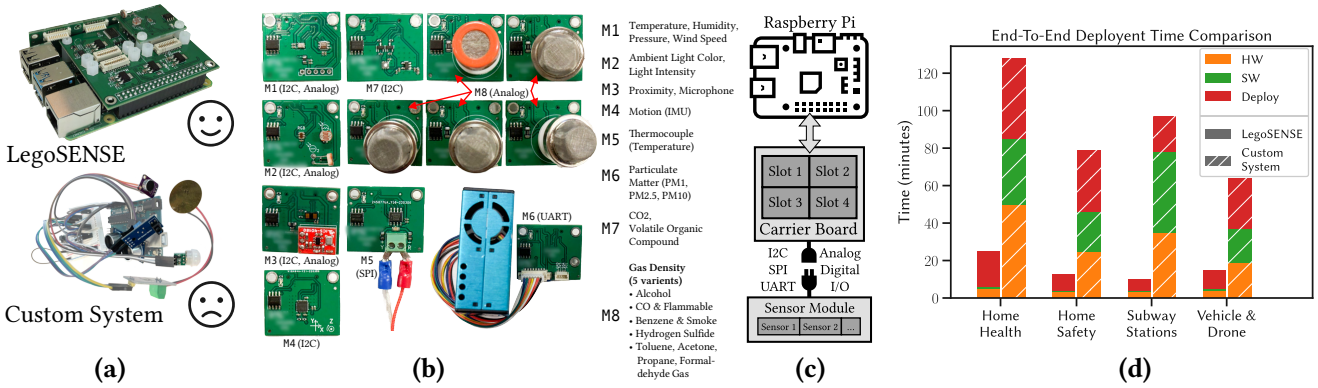


Figure 1: (a) Comparison between LegoSENSE and existing custom sensing solutions. (b) Eight LegoSENSE modules with a total number of 16 different types of sensors using four different communication protocols. (c) LegoSENSE's hardware architecture. (d) End-to-end time comparison of LegoSENSE vs. custom systems for four applications. The graph shows the time spent in four user studies on setting up the sensing system. The blue bar represents the average time consumption of people without embedded system background using LegoSENSE, and the orange, green and red bars represent the time consumption of people experienced with embedded system building and deploying same system without the use of LegoSENSE. It shows that on average even professionals would take 5.4× longer to deploy a system than beginners using LegoSENSE.

ABSTRACT

Domain-specific sensor deployments are critical to enabling various IoT applications. Existing solutions for quickly deploying sensing systems require significant amount of work and time, even for experienced engineers. We propose LegoSENSE, a low-cost open-source and modular platform, built on top of the widely popular Raspberry Pi single-board computer, that makes it simple for anyone to rapidly set up and deploy a customized sensing solution for application specific IoT deployments. In addition, the 'plug and play' and 'mix and match' functionality of LegoSENSE makes the sensor modules reusable, and allows them to be mixed and matched to serve a variety of needs. We show, through a series of user studies, that LegoSENSE enables users without engineering background to deploy a wide range of applications up to 9× faster than experienced engineers without the use of LegoSENSE. We open-source the hardware and software designs to foster an ever-evolving community, enabling IoT applications for enthusiasts, students, scientists, and researchers across various application domains with or *without* prior experiences with embedded platforms or coding.

IoTDI '23, May 09–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0037-8/23/05…$15.00
https://doi.org/10.1145/3576842.3582369

CCS CONCEPTS

• **Computer systems organization** → **Embedded hardware**; **Embedded software**; • **Hardware** → *PCB design and layout*; *Wireless devices*; **Sensor devices and platforms**; **Sensor applications and deployments**.

KEYWORDS

sensor, platform, plug and play system, IoT, rapid deployment

ACM Reference Format:
Minghui Zhao, Stephen Xia, Jingping Nie, Kaiyuan Hou, Avik Dhupar, Xiaofan Jiang. 2023. LegoSENSE: An Open and Modular Sensing Platform for Rapidly-Deployable IoT Applications. In *International Conference on Internet-of-Things Design and Implementation (IoTDI '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3576842.3582369

1 INTRODUCTION

Embedded sensing systems have gained popularity in recent years as they provide some of the most important aspects of the Internet of Things (IoT): detecting an object and/or event in the environment, allowing for the capture of relevant data in real-time and data analytics. With information perceived from surroundings by deploying embedded sensor systems, a wide range of domains can

	Industrial Systems	Open-Source DIY Systems	**LegoSENSE**
Cost	High	**Low**	**Low**
Set-up time	**Quick**	Slow	**Quick**
Customizability	Low	**High**	**High**
Ease of use	**Easy**	Difficult	**Easy**
Inexperienced-user friendliness	**Friendly**	Not Friendly	**Friendly**
Repairability & Reusability	Low	**High**	**High**

Table 1: Comparison between LegoSENSE and existing approaches: LegoSENSE combines the pros of both existing category of approaches.

be enhanced, including transportation, supply chain, healthcare, smart environments, and personal and social interactions [7]. Users in many of these domains, such as building managers, construction workers, scientists, or doctors may need to deploy sensing systems to enable a wide range of applications (e.g.,configurable robots and drones [26, 52, 56], intelligent buildings [19, 22, 24, 51], urban safety [10, 50, 53], digital health [8, 13, 20, 25, 33, 34, 49], etc.). However, many of these users do not have the technical, programming, or embedded systems background required to efficiently build and deploy these systems. Despite the growth in embedded sensing and IoT, there is a lack of flexible and easy-to-deploy sensing systems that can satisfy a broad range of IoT applications.

There are currently two broad approaches to deploying sensing systems. The first category involves custom-made *industrial sensing solutions* bundled with software to access data [42]. Such systems are simple to deploy, but usually cost several thousands of dollars for each bundle for sensors, and lack the flexibility to tailor sensor combinations and data for specific use cases. This means that different applications and user groups need to purchase and learn separate products and systems. For example, a multi-channel readout gas detection devices may cost upwards of $5K [39], and a fire and flame detector costs $2K. [40]. The second category involves connecting and programming inexpensive sensor break-out boards on microcontrollers (MCUs) or microprocessors [12, 21]. These custom *do-it-yourself (DIY) systems* are typically low-cost and offer users the most flexibility. However, sourcing the right components and putting the system together not only require engineering skills, but can also be time consuming and very unwieldy. This is because these systems typically support only one type of common communication protocol in each slot (e.g., I2C), and users still need to program and write their own sensor drivers to read and process data, therefore further increasing the difficulty for people unfamiliar with such platforms. As shown in Figure 1(a), these custom systems often combine many parts and modules haphazardly, which affects even small scale deployments since these systems are prone to connection failures and damages.

In this paper, we present the design, implementation, and experimentation of LegoSENSE, a modular and flexible platform for rapidly deploying sensing systems in a diverse range of IoT applications and domains, such as digital health, smart homes, and urban safety. LegoSENSE provides an *easy-to-use interface for any end user*, such as building managers, middle school students, or doctors, regardless of their programming or embedded systems background, thus allowing them to most efficiently deploy sensing systems for their own needs. Additionally, building on top of one of the most well-known single-board computers, Raspberry Pi, LegoSENSE enables simple-to-design add-ons for developers to introduce new sensors into the LegoSENSE ecosystem, all while being *low-cost*. As shown in Table 1, LegoSENSE fills an unmet need for allowing anyone, particularly those with limited embedded systems background, to quickly and inexpensively deploy sensing systems in a diverse range of IoT applications.

To summarize, we make the following main contributions:

(1) We design and build LegoSENSE, a Raspberry Pi-based integrated hardware and software platform that enables anyone regardless of his/her technical background to easily and quickly deploy sensing systems for a diverse range of IoT applications. We iterated and improved the hardware and software design, aiming for ease-of-use through a series of prototypes and surveys with a large user group with diverse background.

(2) We design and implement a modular hardware architecture with (sensor) modules inserted into a multiplexed carrier board. We design (i) a uniform interface for modules that incorporates all common communication protocols (I2C, SPI, UART, and Analog), allowing modules to be easy to plug-and-play and mix-and-matched; and (ii) a software system to automatically scan for hot-plugged modules and identify, load and run respective sensor drivers, and stream sensor outputs to an easy-to-use web dashboard, API, and/or local filesystem without any configuration from users. As such, users can use modules with "one-click", and let LegoSENSE handle all the complexity.

(3) We build and test the *carrier* board and eight different *module* boards, with a total number of 16 different types of sensors (Figure 1(a, b)). We *open-source* both the software and hardware designs of LegoSENSE to the community at https://github.com/Columbia-ICSL/LegoSENSE. This allows companies, developers and enthusiasts to easily and inexpensively design and incorporate new sensors into the LegoSENSE ecosystem.

(4) We conduct a series of case studies to showcase examples of the diverse range of IoT applications that can be enabled and supported by LegoSENSE. We also conduct extensive user studies comparing LegoSENSE against custom built systems, and show that users of LegoSENSE, on average, take 5.4× less time to set up the system end-to-end. Our user studies encompass a group of diverse end users from various background, including adults with and without technical background, as well as middle school students.

2 RELATED WORKS

IoT is envisioned to be capable of incorporating a vast number of disparate and heterogeneous end systems transparently and smoothly, while offering open access to specified subsets of data to develop a plethora of digital services. Thus, creating a generic architecture for the IoT is an arduous endeavor, owing to the enormous range of devices, link layer technologies, and services that may be included in such a system [55]. Businesses constantly invest in research and development in order to innovate and create new products and services and improve their offerings. Numerous research projects have built modular sensor platforms readily available to the community. Each, however, addresses distinct issues, resulting in subtle but significant distinctions, which we go into detail next.

There are many modular platforms that are based on the Berkeley TinyOS [23]. These platforms [9, 11, 18, 30, 31, 35] are often

small and consume little power, allowing long network lifetimes and inconspicuous node dispersal. A low-power microprocessor and a wireless radio transceiver are used in these sensor nodes. In one such modular platform, MASS [11], each module board has its own dedicated processing unit instead of having a single main processing unit controlling all resources. Although this design style is adaptable in terms of both software and hardware, it adds a significant amount of hardware and software overhead to each sensor node. Mica and its derivatives are other popular wireless sensor networks (WSN) devices, with a form factor of plug-in sensor boards and Atmega processor [9]. However, Mica devices target the users *with a decent amount of knowledge* in embedded platform and coding, which doesn't provide layperson-friendly services, including web-based dashboard and sensor auto-configuration. In addition, Atmega processor is not as popular as Raspberry Pi platforms.

Apart from TinyOS platforms, certain modular sensor systems are purely microcontroller-based without real operating systems. [54] is a plug-and-play sensor system for monitoring urban air pollution. However, it is limited in the types of sensors that can be communicated over the Serial Peripheral Interface (SPI). This has a significant impact on the platform that lowers its scalability. All of these designs either made incremental improvements without significantly increasing scalability and modularity, or they addressed specific issues without providing a holistic solution.

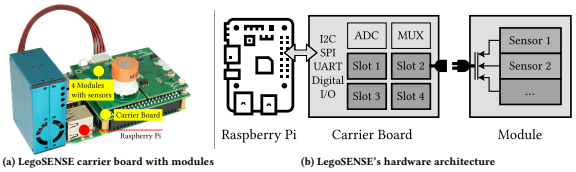
In addition, there are numerous modular sensor systems on the market. [41, 42] provide numerous separate plugs and encapsulate a range of sensors for connecting commercially available devices to a node. However, they often cost several thousand dollars, making them unaffordable to the majority of people. Developing a new custom module for this platform is also difficult.

Open-source electronics vendors such as SparkFun [12] and Adafruit [21] also developed their sensor ecosystem to enable rapid hardware prototype by using a unified connector interface. While their users could connect the sensors with development boards through a cable without the need of building any circuits, their system is based on 3.3V Inter-Integrated Circuit (I2C) communication bus, which limits the selection of sensors. Moreover, from the user's perspective, their connector system only eliminates the need to build the hardware circuit. They do not address the challenges in sensor driver/software development and integration.

In this paper, we present the design and implementation of an open and modular sensing platform for rapidly-deployable IoT applications, called LegoSENSE, which combines the pros of existing systems shown in Table 1. *To the best of our knowledge, LegoSENSE is the first system that provides a complete and easily extensible hardware and software suite that supports all major sensor communication protocols (I2C, SPI, UART, analog), while keeping end-user simplicity and ease of use as the top design priority.*

3 SYSTEM DESIGN

The primary goal of LegoSENSE is to save users time and effort in obtaining the data they want. Thus, LegoSENSE is designed around the concept of user-friendliness, while still meeting the design goals outlined in Table 1. LegoSENSE aims to abstract away implementation details, providing users with the bare minimum necessary to configure the system in any manner they desire. In this section, we will present the overall architecture of LegoSENSE, the various design choices made, and the reasons behind them.



(a) LegoSENSE carrier board with modules (b) LegoSENSE's hardware architecture

Figure 2: LegoSENSE's carrier board and module architecture.

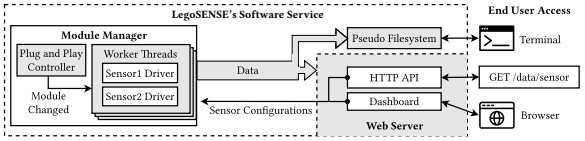


Figure 3: LegoSENSE's software architecture.

3.1 Hardware Design

LegoSENSE functions on top of one of the most popular open-source Linux-based single board computer – Raspberry Pi [14]. As a central component of LegoSENSE, we designed a carrier board, as a hat (shield) inserted into Raspberry Pi's 40-pin header, and an expandable list of modules with various sensors on board, as *module* boards to be inserted into carrier board and interfaced by Raspberry Pi. LegoSENSE's carrier board and an example module are shown in Figure 2.

3.1.1 ***Carrier Board.*** Systems such as Raspberry Pi and Arduino [5, 14] are frequently used in both prototyping and custom-designed systems, because of their low-cost and abundant community support. On such systems, manufacturers commonly package sensors and actuators onto hats (shields) – expansion boards that directly stack on top of the main board's expansion header/pins, abstracting away efforts of circuit building, while ensuring a correct, solid connection. LegoSENSE follows this practice by designing its carrier board into a Raspberry Pi hat, allowing users to assemble the system by simply plugging the carrier board into Raspberry Pi.

To enable users to mix and match different modules, we create four interchangeable slots with a uniform interface. As different sensors may use a variety of communication protocols and power levels, we need to further consider including common protocols and different power supplies to the interface. Thus, LegoSENSE carefully divides and groups the Raspberry Pi's general purpose input & output (GPIO) pins into four groups, each of which connects to one of the four identical slots for modules. With additional circuits on the carrier board, each group (slot) is connected with all the common sensor interfaces (I^2C, SPI, UART, analog inputs, digital I/O) and with two common voltage levels (5V and 3.3V). The implementations are detailed in Section 4.1.1.

3.1.2 ***Module.*** In LegoSENSE's architecture, a *module* is the component that connects sensor(s) to the carrier board. Following a standard dimension, each module contains one or more sensors, their peripheral circuits, and a male connector that can be hot-plugged into its female counterpart on a carrier board slot. As such, LegoSENSE eliminates the need for the user to fiddle with cables or

other peripheral components. LegoSENSE envisions a community and ecosystem that adopts this carrier board-to-module interface standard. However, an issue that arises is that the user has to log into the Raspberry Pi and register the type of module and the slot it was inserted into. This defeats our "beginner-friendly" requirements. To tackle this issue, LegoSENSE further simplifies the user's setup process by including a non-volatile memory chip on the modules, which can be used to support plug-and-play. The memory would be factory-programmed with the carrier board's type, along with any calibration information for the sensor(s). In this way, the carrier board can read from the memory and automatically register and set up the sensor as soon as it is plugged in.

3.2 Software Design

Our top design goal is that the system needs to be easy enough for users in various background to use, and at the same time flexible enough for various applications. To fulfill such a requirement, the software system for LegoSENSE is built around three major components: a web server, various sensor drivers, and a module manager that manages sensor drivers and connects with system interfaces. The module manager contains a plug and play controller that detects hardware changes and manages a list of worker threads running the sensor drivers. This architecture is shown in Figure 3.

3.2.1 ***Web Server.*** LegoSENSE employs a web server on the Raspberry Pi, and makes all data flow in its pipeline through HTTP. First, the web server serves a web-based dashboard via which users may examine system status and sensor data in real time, and adjust sensor configurations. The dashboard makes the system beginner-friendly by allowing the user to control the system by just clicking buttons on any web browser, even on smartphones.

However, such basic graphical user interfaces make customization difficult. When building a cluster of LegoSENSE systems, opening web pages to configure each is obviously very cumbersome. Thus, we also implement RESTful Application Programming Interface (API) within the web server. Not only does the dashboard utilize such API to obtain sensor data, but an advanced user is also able to programmatically access the system via API calls. We further develop a command line application that makes use of such APIs for users with scripting needs.

The third advantage of the web server (HTTP) architecture is its high compatibility and scalability. For advanced users to control the system programmatically, they can utilize LegoSENSE's API in any programming language they prefer. We also note that since the system can be controlled via API calls, in an application scenario where the user chooses to deploy a cluster of LegoSENSE, they can use the IP addresses to easily control the system in the way they want. In addition, in an outdoor deployment with limited Wi-Fi or Ethernet access, we have also made the system compatible with LTE network cards via USB, to offer Internet connectivity via cellular data, and the HTTP-based architecture would keep the rest of the pipeline the same. As shown in Figure 3, LegoSENSE's software system interfaces with users via a web server hosted on the Raspberry Pi that serves both web pages and API calls. The web server is wrapped into a service daemon process that manages sensor drivers and data acquisition.

3.2.2 ***Sensor Drivers.*** The core idea for LegoSENSE is to read the data from the various sensors and deliver to the user. However, there are different types of sensors on the market, and they need to be interfaced in different ways. For example, some sensors output data over different digital communication buses such as SPI and I^2C, whereas some other sensors output data as analog voltages. LegoSENSE needs to be carefully designed so that the system is flexible enough to automatically load a variety of drivers, adapting different implementations.

We note that many common sensors have an abundance of community support with their drivers and code examples. We would like developers to spend as little time as possible developing sensor drivers by taking advantage of existing code. In other words, we want developers to be able to easily transform example programs into LegoSENSE's system with minimal modifications. To address the above challenges, LegoSENSE begins with a generic template that imposes a set of uniform software interfaces to work with the rest of the software system. Using the template, LegoSENSE developers define and fill out the bare minimum of code. Specifically, developers need to define the name of the module and sensors on the module, initialization code for sensor drivers, the code to obtain a sensor reading, when to expect the next reading, and the code to read and parse user configurations. More details about the template will be described in Section 4.2.1.

3.2.3 ***Plug-and-play Controller.*** One core motivation of LegoSENSE is to simplify the workflow for end users as much as possible. Specifically, we want users to plug in the sensor and obtain the data without having to configure anything or execute any command. Hardware-wise, as mentioned in Section 3.1.2, LegoSENSE includes a non-volatile memory chip on each module in order to store their identifier. In LegoSENSE's software system, a background service, the plug-and-play controller, periodically pulls data out of the non-volatile memory chips to identify the type of the module, and obtains the driver for the module from a repository automatically.

3.2.4 ***Worker Threads for Drivers.*** Now that we have the sensor drivers written in a format that has uniform interface, someone needs to automatically run the drivers, obtain the data, and pass the data to the end user. After LegoSENSE detects the presence of a module, its plug-and-play controller recognizes the type of module, and starts a thread running its driver (worker thread). As seen in Figure 3, each worker will execute its respective module's driver, and contains an interface to take in updated sensor configuration, and output sensor data to the user.

3.2.5 ***End User Access.*** In order for end users to access the data, LegoSENSE provides three different methods targeting both advanced and beginner users. As seen in Figure 3, the software service follows the Unix design philosophy, in that "everything is a file". It maintains an in-memory pseudo filesystem to temporarily store sensor data obtained from worker threads. This allows advanced end users who are familiar with UNIX commands to integrate LegoSENSE into their workflow, namely, writing scripts to obtain sensor data, or simply previewing data in the terminal.

Next, as mentioned in Section 3.2.1, LegoSENSE takes advantage of its web server and adds two more approaches for users to interface with the system: (1) RESTful API through which advanced

	I^2C	SPI	UART (TX, RX)	GPIO	ADC
Module1	I^2C1	SPI0, CS0	UART0 (14, 15)	24	ADC1
Module2	I^2C1	SPI0, CS1	UART2 (0, 1)	25	ADC2
Module3	I^2C6	SPI1, CS0	UART3 (4, 5)	26	ADC1
Module4	I^2C6	SPI1, CS1	UART5 (12, 13)	27	ADC2

Table 2: Carrier board - module interface configuration.

users can interact with the system using their preferred programming language; (2) web dashboard allowing users to configure the system, acquire sensor data, and visualize it in real time.

4 SYSTEM IMPLEMENTATION

In this section, we describe the detailed hardware and software implementation of LegoSENSE.

4.1 LegoSENSE Hardware

As described in Section 3.1, the architecture of LegoSENSE includes a carrier board as a Raspberry Pi hat and modules with a standardized interface that works with the carrier board.

4.1.1 ***Carrier Board as a Raspberry Pi Hat.*** LegoSENSE's Raspberry Pi hat carrier board allocates the communication buses and pins from Raspberry Pi's 40-pin connector, divides them into four functionally-equivalent groups, and connects each to one module slot. The allocation is shown in Table 2.

Besides allocating and utilizing the data buses provided by Raspberry Pi's 40-pin connector, LegoSENSE also adds two ADS1115[47] 16-bit 4-channel analog-to-digital converters (ADCs) onto the carrier board. The two ADCs are connected to Raspberry Pi through I^2C bus, and the 8 available analog channels are connected to the 4 slots, with 2 channels per slot, to support modules with analog sensors in both single-ended and differential mode.

One potential hardware issue that arises when interfacing between microcontrollers and sensors is the mismatched input/output (I/O) voltage levels. For example, many analog sensors operate at 5V, whereas most microcontrollers, including the Raspberry Pi, only tolerate 3.3V I/O. To accommodate 5V analog sensors, we power the two ADCs from a 5V supply so that it can safely read outputs from 5V analog sensors, and add level shifter circuits to the ADC's I^2C interface to match the voltage to Raspberry Pi's 3.3V.

To support module plug-and-play, LegoSENSE adds non-volatile memory onto each module to store module's information. We can monitor the module slot allocation by continuously attempting to find and read from the memory at each slot. If the memory is not present, it indicates that the module is not installed. If we were able to read the module's information from the memory, the software performs automated actions to set up that module. However, a challenge that we came across is that, assuming all four module slots are taken, we have four memory chips to read from. There are not enough I/O pins to be allocated to four separate memory chips. To resolve this, we connect all four memory chips onto the same I^2C bus, through an I^2C multiplexer (MUX), TC9548A [48], to access the memory chips in a round robin manner.

As seen in Figure 4a, the carrier board contains 4 module slots, each with dimension 32mm x 28mm. We chose this size for each

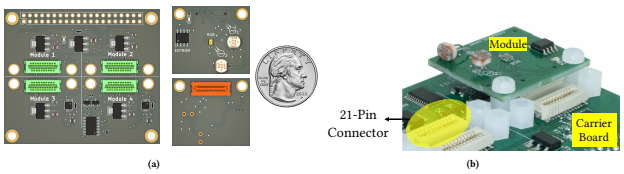


Figure 4: (a): LegoSENSE's carrier board (left), module (middle) vs a US quarter (right). (b): LegoSENSE's module attached to carrier board.

module in order to have four of them on the carrier board to satisfy the need for various sensors in the common deployment tasks (Table 3), as well as to be big enough to fit in the majority of sensor chips on the market. To support high current-consuming sensors such as analog gas sensors which rely on resistive heating elements, each slot contains a dedicated 1A low-dropout voltage regulator (LDO), AMS1117-3.3V [1], to regulate the 5V from Raspberry Pi power supply to 3.3V. As shown in Figure 4b, each slot uses a pair of 21-pin polar mezzanine board-to-board connectors to mate the carrier board with module. The connectors used are 91931-31121LF [3] (receptacle used on carrier board) and 91911-31321LF [2] (plug used on module).

4.1.2 ***Modules.*** LegoSENSE's module contains one or more sensors that has corresponding software in the system. Each module can be snapped onto one slot on the carrier board with a connector as shown in Figure 4b. The modules have a standard size of 30mm x 26mm with two holes on both sides of the connector for fastening the module to the carrier board with stand offs and M3 screws. Note, as mentioned in Section 4.1.1, the dimension of a carrier board slot is 32mm x 28mm, which is 2mm longer on each dimension as compared to the size of module. The extra 2mm tolerance ensures that all modules can fit easily onto the carrier board.

To implement plug-and-play, each module also includes a M24C02 [44] 2Kb I2C bus electrically erasable programmable read-only memory (EEPROM) to store an identifier for LegoSENSE's software system to recognize the presence and type of module. We have developed 8 different modules (see Figure 1(b)) to evaluate LegoSENSE with four common communication protocols for sensors (I2C, SPI, UART, and analog), as shown in Figure 1(b).

- M1 **Ambient Environment:** Temperature & Humidity (AHT20) [6], Pressure (SPL06) [17], Wind Speed (Modern Device) [32]
- M2 **Ambient Light:** RGB (TCS34725FN) [4], Light Intensity (photoresistor)
- M3 **Human Activity:** Distance (VL53L1X) [43], Microphone (ICS-40180) [45]
- M4 **Motion:** Accelerometer & Gyroscope (MPU6050) [46]
- M5 **Thermocouple:** -200°C to +1350°C thermocouple (MAX31855KASA) [29]
- M6 **Particulate Matter (PM):** PM1 & PM2.5 & PM10 Sensor (PMS5003) [36]
- M7 **Indoor Air Quality:** Volatile Organic Compounds & CO2 (SGP30) [38]
- M8 **Gas Sensor:** Alcohol (MQ-3), Carbon Monoxide & flammable gas (MQ-9), Benzene & Alcohol & smoke (MQ-135), Hydrogen Sulfide (MQ-136), Benzene & Toluene & Alcohol & Acetone & Propane & Formaldehyde gas (MQ-138)

4.1.3 ***Community and Module Developers.*** The carrier board and module design and assembly files, along with their complete software stack are open sourced [1] . We also release design templates for modules adhering to LegoSENSE's module standard for developers to promptly get started with their module designs. 5 developers with different backgrounds participated in this study and their evaluation and feedback are described in Section 6.

4.2 LegoSENSE Software

For ease of modification, LegoSENSE's software system is based on Python. As discussed in Section 3.2, LegoSENSE's software service is a daemon process that runs automatically in the background at boot time. The service daemon starts with a module manager that manages the execution of sensor drivers and passes the data to two interfaces for users to access - a web server, and a pseudo in-memory file system. In this section, we dive into the details of each component in the software stack.

4.2.1 ***Module Drivers.*** As discussed in Section 3.2.2, to fulfill the requirement of minimal coding for developers, module drivers follow a predefined template so that developers only need to fill in the minimum code necessary. We implemented a base driver class to which each module driver must inherit from. It contains the following sections that module developers need to override:
(1) Declaration of what sensor(s) exist on the module, and for each sensor, what data columns are expected. For example, our ambient environment module contains *"Temperature Humidity Sensor" with ["temperature", "humidity"], "Pressure Sensor" with ["pressure"], and "Wind Sensor" with ["wind speed"].*
(2) Loading and parsing the user configuration file. Most sensors don't simply output readings – they need to be configured in different modes, to measure different ranges, or to vary the sampling frequency etc. While we want to abstract all implementation details away from end users, we also want them to have access to such configurations. Thus, we allow module developers to optionally define a separate *.conf* file for end users to modify sensor configurations. The *.conf* format configuration file is parsed and displayed on the web dashboard for end user. In the sensor driver, the same *.conf* file is loaded into program to allow developers to use those configuration values in their sensor drivers.
(3) Initialization of sensor driver(s). After the configuration *.conf* file is parsed, driver developers need to fill this section to initialize their sensors with the configuration values. Since the module can be plugged in different slots that are connected to different protocol buses, a data structure is passed in by the worker thread which runs the driver. The data structure defines what protocol bus the current module is connected to. module driver developers can take the bus from the data structure and initialize the sensor(s) on the module through the given protocol bus.
(4) Code to read sensor readings. Once sensors are initialized, driver developers need to fill the code to obtain sensor readings and return the readings to worker threads.

4.2.2 ***Module Manager.*** At system initialization, the module manager first starts the plug-and-play controller that detects during run-time what modules are plugged in, and creates worker threads that execute the respective module driver. The structure of the module manager is illustrated in Figure 5, with the implementation details listed below.

[1] https://github.com/Columbia-ICSL/LegoSENSE

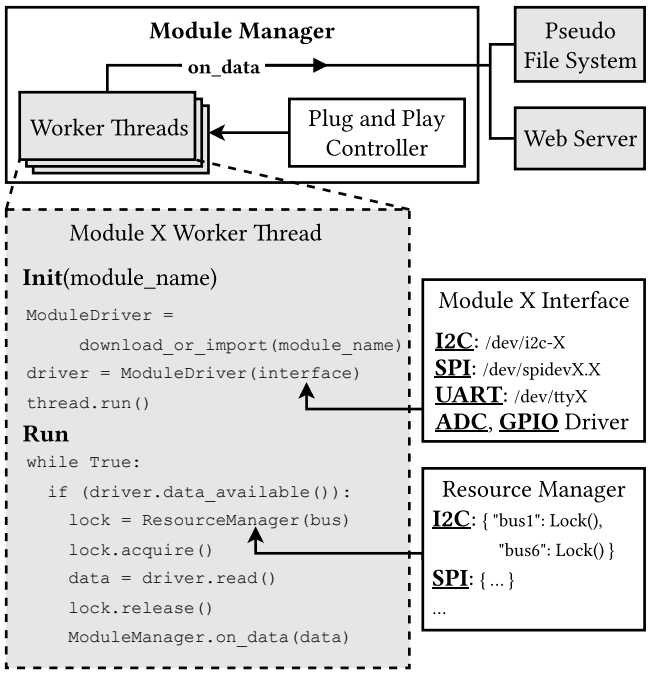


Figure 5: Module manager implementation.

4.2.3 ***Plug-and-play Controller.*** To automatically detect modules once they are plugged in, we add EEPROMs to each module and have them preprogrammed with the name of the modules, along with a MUX on the carrier board to switch between the EEPROMs on each module. On the software side, we implement a background service to check the presence of EEPROM on each slot by first controlling the MUX to switch to the respective channel, and write the EEPROM's fixed I^2C address to the I^2C bus and check if it receives an acknowledgement (ACK) from the EEPROM. If so, it reads the content in EEPROM to identify the module. The service performs this action in a round-robin manner and checks all four module slots once per second. The service runs on a thread continuously in the background as part of the module manager. If the service detects any module change, either new module(s) plugged in or an existing module being removed, it reports to the module manager, which is responsible for starting and/or stopping respective worker threads running the module's driver. As such, when the user modifies the hardware during run-time, the system detects and loads the respective driver for the module automatically.

4.2.4 ***Worker Threads for Drivers.*** As described in Section 3.2.4, we implement a generic worker thread that is started for each module in order to execute the module's driver code. The life-cycle of a worker thread starts from the plug-and-play (PnP) controller recognizing the module, either when the system starts, or after a hardware change by the user. Once the module manager receives

the information of the new module from the PnP controller, it starts a worker thread passing in the name of the module.

The worker thread initialized by the module manager is passed in with the name of the module and its slot number. The name helps identify the correct module driver to load, and the slot number helps in identifying the hardware interface to be allotted to the driver. The pseudo-code for the initialization can be found in Figure 5.

First, the worker thread checks if the driver for the module exists in the system locally. If the driver does not exist, it downloads the driver from an online git repository automatically. Once the driver is ready, it obtains the hardware interface corresponding to its slot number. As seen in Figure 5, the interface contains the path to I^2C, SPI, and UART protocols, as well as the driver for ADC and corresponding GPIO pins. The worker thread then initializes the module driver, passing in the interface. Finally, an infinite loop is started, where the worker thread waits for the next sensor reading to be available from the module driver. It then reads the data and passes it back to the module manager.

In rare cases, certain modules may experience hardware faults (such as connection problems, sensor overheating, etc), or the driver software may malfunction and crash or freeze. The design decision to use threads for each module gives us the freedom to implement a software watchdog to protect a malfunctioning module from adversely affecting the rest of the system, and ease of resetting a module to recover it.

Now a natural question is, as the sensor data will propagate through the software system – being sent to the API server, to the dashboard for real-time visualization, and to log files, how do we standardize the format of a "generic" sensor reading in the system? Since a generic module could contain multiple sensors that may be sampled at different rates, and not all sensors produce an output at the same time, a strictly structured data structure for the sensor data would be difficult to work with. We define the data structure to be a dynamic hash table, with the keys being the names of the available data columns, and the values being their data. When the hash table is being returned from the driver, the system adds an addition entry to the hash table for the timestamp. Such a flexible data structure not only solves the challenge of arbitrary data arriving at arbitrary time, but also makes itself easy to be converted to JSON and CSV format, that are used in API / web dashboard and in log files.

4.2.5 ***Potential Multiple-access Issue.*** With the worker threads and drivers, we can now start to run the sensor drivers and acquire data. However, an issue quickly arises during our testing: the analog reading of a sensor sometimes shows up as the actual reading from a different analog sensor connected to the same ADC. It turns out that this is because although threads will not truly run in parallel, they can be context switched at any time, leaving potential race conditions and multiple access issues to the communication buses.

To avoid drivers interfere with each other, we need to implement a mechanism to make our sensor drivers "thread-safe". The common practice is to add software "locks" to grant mutual exclusive access to the resource, in our case, the communication bus that the drivers share. To do so, we add a resource manager class that is a singleton (in other words, restricted to be instantiated only once). The resource manager contains locks for each bus that is shared between multiple modules. With the locks in the resource manager, a natural thought one would come up with is to pass the them to each module driver so they can use it. This is a bad idea for multiple reasons. First, this adds complexity to the driver developers as they have to worry about adding those locks into appropriate places and not to miss any. Further, badly developed drivers may easily interfere other drivers by not using a lock when accessing the shared bus, or holding the lock for too long, which results in a dead-lock in the system. Thus, *by default*, we leave the work of appropriate locking to the worker thread. Before the worker thread calls the driver to read data, it acquires the respective lock (if any) for the bus that will be used. In this way, all that the driver developers need to add is to declare what communication bus(es) their driver will be using, and then the worker thread will handle the rest accordingly.

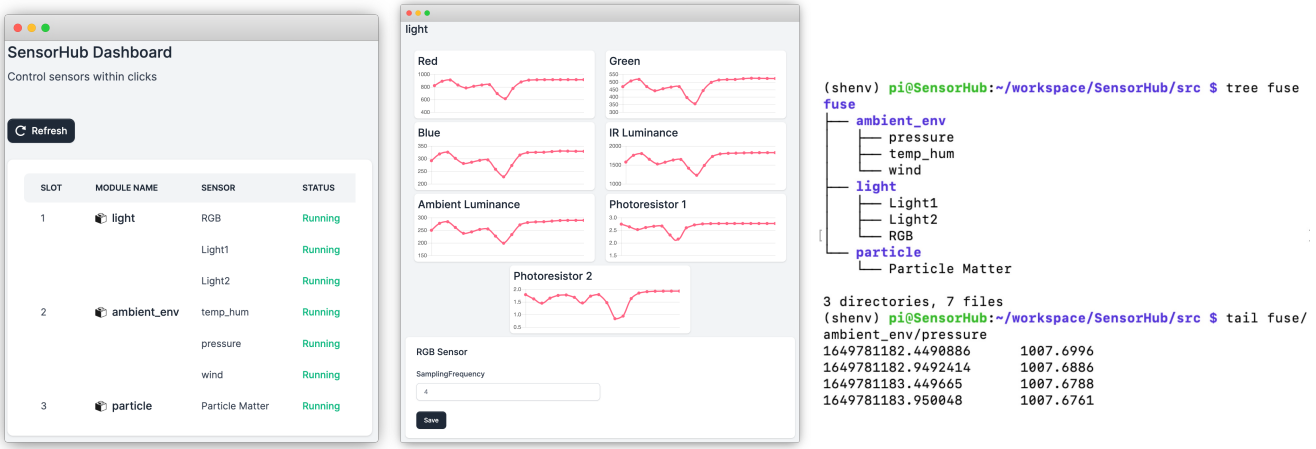
This is the default behavior. We do note that some drivers need to perform sleep for an arbitrary amount of time before data is available – in this case sleeping with a lock acquired is a bad practice, and slows down the entire system. Thus, we still leave the option for the module drivers to handle the locking themselves.

4.2.6 ***End User Access via Pseudo In-memory File System.*** As discussed in Section 3.2.5, the first method that end users can access LegoSENSE data is through a pseudo in-memory file system. The in-memory file system is maintained using Linux kernel's "Filesystem in Userspace (FUSE)" framework [15], with "fusepy" library [16].

First, we implement a background service that utilizes FUSE to maintain the pseudo file system (FS). The service runs in parallel with the module manager (Section 4.2.2). As seen in Figure 6(c), we structure the pseudo FS in a tree format – it first mounts to a directory, and creates up to four directories representing the four modules. The directories will be named after each module. Within each directory, there would be multiple files named after each sensor on the module. These names are defined by the module drivers as discussed in Section 4.2.1. Each file would contain a stream of the data from the sensor it belongs to.

To implement this, we need to implement several callbacks defined by FUSE, with the two most important ones as: list directory, and read files. As our module manager keeps track of the list of modules currently running and their corresponding worker threads, in the list directory callbacks, if the user is at the root level, we would call module manager to retrieve the list of modules. If the user is within the module's folder, we would retrieve the list of sensors within that module from its driver. In the read file callback, we would identify the file and have module manager find its corresponding worker thread, and retrieve the data from its in-memory buffer that stores the latest data. The data would then be returned in a csv format, so that it is easy for users to read, plot, or compare.

4.2.7 ***End User Access via Web Server.*** LegoSENSE's web server is built using Flask[37], a lightweight Python web framework, where we implement a series of RESTful APIs:
(1) API to obtain system status, which returns run-time information and statistics from the plug-and-play controller and worker threads that run sensor drivers;
(2) API to obtain sensor data, which retrieves and returns the data from the corresponding worker thread that runs the driver for the specific sensor;



(a) Dashboard homepage **(b) Realtime data visualization & configuration** **(c) File system containing data via terminal**

Figure 6: End user access via (a, b) web dashboard and (c) file system through terminal.

(3) API to edit sensor configuration, which updates the sensor configuration file stored locally and tells the worker thread to reload and restart sensor driver;
(4) API to start or stop sensor, which tells worker thread to perform the corresponding operation.

As seen in Figure 6(a,b), a dashboard is implemented in the web server to allow users to monitor system status, visualize sensor data in real-time, and edit sensor configurations. To carry out these functionalities, the dashboard makes use of the aforementioned APIs. In addition, to display live sensor data to users, the dashboard's JavaScript code subscribes to server-sent events (SSE) containing live data, from the web server. The data flow in the backend is as follows: first, we set up a *Redis* in-memory pub-sub message broker. Then, we set up each worker thread that is running sensor driver to publish the data to *Redis* as soon as they become available, and have the web server subscribe for those sensor data. When the user accesses the dashboard page that streams live data for a particular sensor, the JavaScript code opens a persistent HTTP connection to an endpoint in the web server that serves SSE. When data becomes available, the web server receives the data through its subscription to *Redis* and sends the data to user's browser through SSE in the corresponding channel. As the browser receives the data, it then uses *plot.js* to plot the data in real time for the user.

4.2.8 ***User Bootstrapping.*** A common issue with wireless systems is user bootstrapping. In other words, how does a user deploy the wireless system on the first execution? To resolve this, we first scan for any available wireless access points that LegoSENSE can connect to. These could be pre-configured in a text file in the Raspberry Pi's SD card. However, in the scenarios when there is no available network, LegoSENSE hosts a Wi-Fi Access Point, with a known Service Set Identifier(SSID) and password. The user can now connect to this access point using any of his or her devices. Once connected, the user can access the web dashboard, where he can start using the system right away, or configure the Wi-Fi setting directly on the dashboard. The user enters the SSID and password of the Wi-Fi access point. LegoSENSE then disables its own access point and tries latching to the provided SSID. If the connection succeeds, the user can access the device through the network. In case the connection fails using the given credentials, it hosts the WiFi access point again, and the user could repeat the steps to reconfigure the Wi-Fi credentials.

Another challenge on the user side is obtaining the IP address of the LegoSENSE device. If we had to ask user every time to either hook up LegoSENSE to a monitor and keyboard, or log onto the router to retrieve the IP address, it would defeat our objective of easy-to-use. To solve this, we first note that each Raspberry Pi has a unique media access control address (MAC address) associated with its network interface. This MAC address can serve as a unique identifier for the device. With this, we have the LegoSENSE software service automatically post its IP address, along with its own unique ID, to an external server that maintains a lookup table between the unique ID and its IP address. Then, we set up an external web server that takes in the unique ID and returns an HTTP redirect to the IP address corresponding to the unique ID in the lookup table. We call this our "web redirect server". Now, for each LegoSENSE device, since its unique ID is fixed, we embed a link to the web redirect server, including the unique ID, into a QR code printed on the device. As such, when the user scans the QR code, the web redirect server redirects the user's browser to the IP address of the LegoSENSE device, on which the dashboard will then be present. By doing so, we hide all the networking details from the user. The user can always scan the QR code or bookmark the link in the QR code to access the dashboard directly.

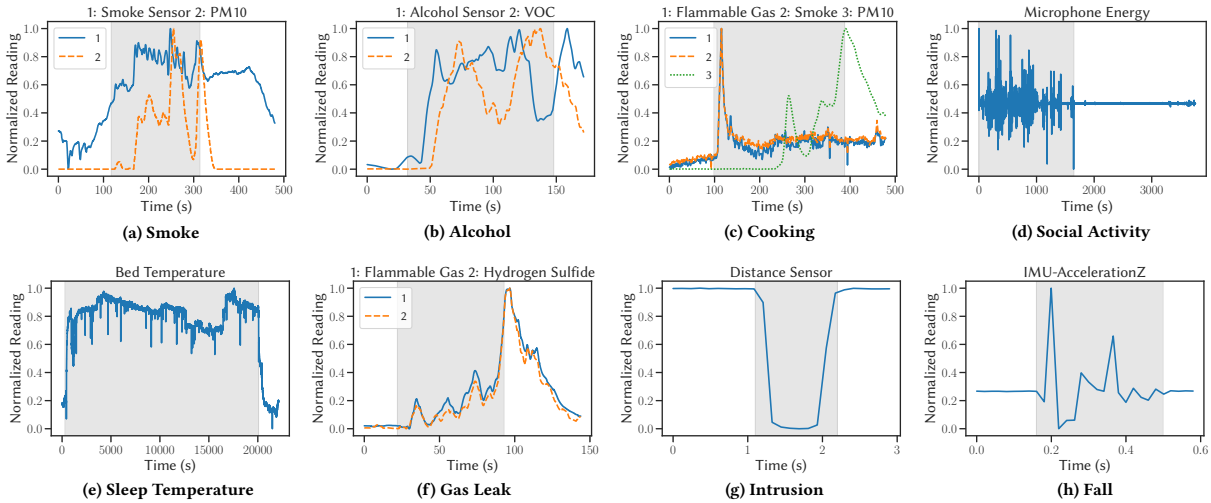
5 EVALUATION

In this section, we evaluate LegoSENSE by performing user studies with four distinct application scenarios to demonstrate how the design of LegoSENSE enables people from diverse domains to rapidly build sensing systems for a variety of tasks.

We recruited two groups of users to build and deploy sensor systems for the four applications scenarios (**A1** to **A4**). We obtained

Application		LegoSENSE Setup			Off-the-shelf Components + Microcontroller Solution		Average Time Saved with LegoSENSE (Percentage)
		Modules Used	Component Cost	Avg. Time + Std (Non-Expert)	Configuration (cost)	Avg. Time + Std (Expert)	
Home - Health	Smoke Habit Inference	M6 **Particulate Matter**, M7 **Indoor Air Quality**	$45.1	0.41 + 0.12 Hour	PMS5003 ($26) SGP30 Breakout ($17.5)	2.13 + 1.56 Hour	1.72 Hour (81%)
	Drink Habit Inference	M7 **Indoor Air Quality**, M8 **Gas** (Alcohol)	$13.4		SGP30 Breakout ($17.5) MQ-3 + Adapter ($2.3) ADS1115 ADC ($5.3) BSS138 Logic Level Converter ($0.8)		
	Cooking and Eating Habit Inference	M6 **Particle Matter**, M8 **Gas** (flammable), M8 **Gas** (smoke)	$40.9		PMS5003 ($26) MQ-9 + Adapter ($2.3) MQ-135 + Adapter ($3) ADS1115 ADC ($5.3) BSS138 Logic Level Converter ($0.8)		
	Social Activities Inference	M3 **Human Activity** (microphone)	$14.6		USB Microphone ($15)		
	Sleep Temperature Monitoring	M5 **Thermalcouple**	$10.5		MAX31855K Breakout ($5.8) Thermocouple Type-k ($7)		
Home - Safety	Gas Leak Detection	M8 **Gas** (flammable), M8 **Gas** (hydrogen sulfide)	$13.8	0.21 + 0.07 Hour	MQ-9 + Adapter ($2.3) MQ-136 + Adapter ($35) ADS1115 ADC ($5.3) BSS138 Logic Level Converter ($0.8)	1.32 + 1.22 Hour	1.11 Hour (84%)
	Intrusion Detection	M3 **Human Activity** (distance)	$14.6		VL53L1x Breakout ($15)		
	Fall Detection	M4 **Motion** (IMU)	$6.7		MPU6050 Breakout ($3.3)		
Subway	Ambient Environment	M1 **Ambient Environment** (temperature, humidity, pressure, wind speed) M2 **Ambient Light**	$29	0.16 + 0.06 Hour	BME680 Breakout ($18) Wind Sensor ($22) TCS34725 Breakout ($4.3) Photoresistor ($0.1) ADS1115 ADC ($5.3) BSS138 Logic Level Converter ($0.8)	1.61 + 1.34 Hour	1.45 Hour (90%)
	Air Quality	M6 **Particulate Matter**, M7 **Indoor Air Quality**	$45.1		PMS5003 ($26) SGP30 Breakout ($17.5)		
Vehicle	Ambient Environment	M1 **Ambient Environment** (temperature, humidity, pressure)	$6.8	0.25 +0.11 Hour	BME680 Breakout ($18)	1.08 Hour + 0.45 Hour	0.83 Hour (77%)
	Air Quality	M6 **Particulate Matter**, M7 **Indoor Air Quality**	$45.1		PMS5003 ($26) SGP30 Breakout ($17.5)		

Table 3: Evaluation setup and comparison with a custom solution built with off-the-shelf components and microcontrollers. Most of the components cost tens of dollars, which is comparable with putting together custom solutions using widely-available low-cost discrete components.



(a) Smoke **(b) Alcohol** **(c) Cooking** **(d) Social Activity**

(e) Sleep Temperature **(f) Gas Leak** **(g) Intrusion** **(h) Fall**

Figure 7: Sensor data collected by LegoSENSE in our home deployment that shows home safety-related events and potential activities that may indicate mental health problems. Highlighted area denotes the time where the corresponding event occurs.

approval from the Institutional Review Board (IRB) prior to the experiments. We recruited 10 people between the ages of 20 and 30 to use LegoSENSE and obtained their informed consent. Sensor data obtained from user activities were anonymized and deleted after the study.

The first group of users consists of five individuals who have no background in sensor deployment or coding, including therapists, civil engineers, and data analysts. The second group contains five *experienced* users who have coding, electrical engineering, or sensor

deployment background. We denote the first group as the "non-expert group" and the second group as the "expert group". We had the non-expert group deploy LegoSENSE for the four application scenarios, **A1** to **A4**, listed in this section, and as a comparison, had the *experienced* users complete the same setup with custom systems based off of Raspberry-Pi.

For each deployment task, we measure the end-to-end time it takes both groups from learning the application requirements to constructing the system and deploying it to a point where it is ready to collect data. We also display plots and figures detailing the events that LegoSENSE can detect in each scenario of the application. We summarize our application scenarios, deployment setup, and user study results in Table 3.

5.1 Application Scenarios

5.1.1 Home Monitoring System. In the first two applications, we used LegoSENSE to deploy a variety of sensors to monitor home safety (**A1**) and mental health (**A2**) in ten participant's home for two weeks. For each home, we used LegoSENSE to deploy two particulate matter (PM) sensors in the living room and kitchen to detect cooking and smoking; MQ3, MQ9, MQ135, MQ136, and VOC / CO2 sensors to detect gas leaks; a distance sensor near the entrance of the home to detect intruders; a temperature sensor in the bedroom to monitor sleep quality; a microphone in the living room to detect social activities; and a vibration sensor in the living room to detect falls.

Participants recorded all activities and the time the activities took place in their home. Figure 7 shows the sensor readings for several events. For example, we see an increase in the amount of flammable gas and smoke measured at 100 s, when the participant began to cook, and an increase in particulate matter reading starting 250 s. We trained and evaluated a classifier that differentiates between four daily activities as shown in Figure 8c. We see that we are able to identify most events accurately with low error using data collected from LegoSENSE.

5.1.2 Subway Study. A third important application is monitoring the PM level in subway stations or other areas with heavy traffic (**A3**). Inhaling dangerously high amounts of particles can trigger heart attacks, exacerbate asthma, or even cause early death [28]. Sensing particulate matter across a wide area can inform the general public about areas that may be dangerous for health. For this application, we chose New York City, a busy urban city, and deployed LegoSENSE in 20 subway stations to sense the PM2.5 levels at each station for a 4-hour period.

Figure 8a shows the plot of PM2.5 readings across subway stations in Manhattan, New York City. It can be seen that subway stations in lower Manhattan have lower levels of PM2.5, potentially due to better ventilation or larger spaces in these stations. Our initial hypothesis was that more people passing through each station should result in higher PM levels. We see that the PM level (darkness of circle) at each station is largely uncorrelated with how busy each station is (size of circle). Although this deployment produced a negative result, it narrows down the list of major factors that could cause poor air quality in some stations. An extension to this deployment could be to estimate the fine-grained traffic flow and population density of people within stations themselves to inform subway riders which parts of the station may have better air quality (less particles) and are more suitable for waiting for the next train.

	Non-Experts: LegoSENSE	Non-Experts: RPi	Experts: LegoSENSE	Experts: RPi
Clarity	8.6	4.1	9.4	7.3
Usability	9.1	4.7	8.3	7.7
Customizability	8.9	6.5	6.2	7.4
Reliability	8.3	5.6	8.1	7.7
Aesthetics	8.5	4.1	8.8	6.0

Table 4: Survey responses from "non-expert" and "expert" groups (averaged) across four different applications using both LegoSENSE and creating a custom platform using a Raspberry Pi (RPi). Scores range from 1 (worst) to 10 (best).

5.1.3 Sensing and Intelligence in Vehicles. Another class of applications is using vehicles as powerful mobile sensing platforms (**A4**). As a vehicle passes by districts and neighborhoods, it can sense factors such as traffic flow, air quality, or noise level, which homeowners or commercial business owners may be interested in when purchasing real estate. We integrate LegoSENSE into a vehicle and use a PM2.5 module to sense particulate matter.

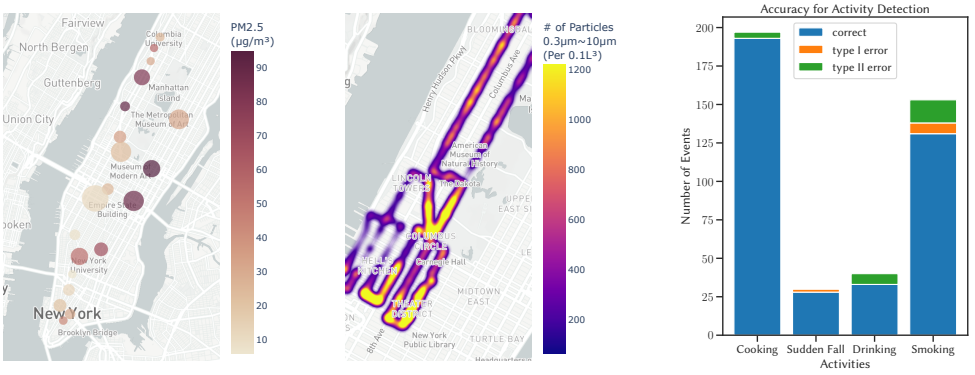
We chose to drive around Manhattan due to its diverse environments and record the levels of particulate matter (PM), shown in Figure 8b. Here, we see that areas around Columbus Circle and midtown (Times Square) have high levels of PM (light yellow), while all other areas have moderate levels (dark purple). Those areas are among the busiest areas in the upper and midtown Manhattan area, and as such, we can expect to see higher particulate matter readings due to the large traffic of vehicles that are in these areas.

5.2 LegoSENSE User Study

To evaluate the convenience that LegoSENSE brings users, we measure the time it took both groups to build the hardware, program the software, and deploy the system to collect data while implementing their application scenarios (**A1** to **A4**).

Figure 1(d) shows the average time it took for both groups of users to set up each application using LegoSENSE and custom platforms. We see that because LegoSENSE does not require users to perform coding and because of its modularity, the *non-expert* user group completed the task relatively quickly, although they lack experience and expertise. The *expert group* took much longer to deploy a solution using a custom-built platform, mainly due to the wiring and coding involved. The average *end-to-end* deployment time needed by the *non-expert* user group is shorter than the time taken by the *experienced* user group to build *only the hardware*. On average, we see that the *novice* user group took up to 5.4× less time using LegoSENSE than the *experienced* users. This comparison shows that LegoSENSE speeds up and greatly simplifies sensor deployments for a variety of applications.

We also surveyed each participant across five dimensions: clarity of instructions, ease-of-use, customizability, reliability, and aesthetics. The results are tabulated in Table 4. We see that both groups scored LegoSENSE higher across most dimensions. We also see that users scored the *usability* and *clarity* of LegoSENSE significantly higher. This shows that LegoSENSE greatly improves the clarity and ease of deploying sensing systems over custom solutions for anyone, regardless of background. *Customizability* was scored poorly for LegoSENSE compared to discrete components in the



(a) (b) (c)

Figure 8: (a) PM2.5 levels sensed by LegoSENSE across 20 subway stations across Manhattan, New York City. Larger circles denote larger number of subway riders; darker color denotes higher levels of PM2.5. (b) Number of particle between 0.3μm and 10μm per 0.1 liter of air measured by LegoSENSE integrated into a vehicle. Bright colors indicate high levels of particle density in the area. (c) Classification accuracy for four daily activities from data obtained by LegoSENSE placed in multiple real homes.

expert group. This is because of the limited number of sensors that seasoned developers had access to at the moment compared to the suite of available sensors compatible with Raspberry Pi. However, tapping into the vast number of sensors compatible with Raspberry Pi requires more knowledge and is more time consuming, as shown in Table 3. To help increase the library of sensors compatible with LegoSENSE and mitigate the *customizability* issue, we open-source our designs, allowing the community to contribute new compatible sensors into the LegoSENSE ecosystem.

	Engineers	Scientists	Middle School Students	Developers
Clarity	9.3	8.9	8.1	9.4
Usability	8.9	9.2	8.4	9.0
Customizability	8.0	9.0	9.3	8.9
Reliability	8.7	8.6	9.1	9.2
Aesthetics	8.4	8.8	8.2	8.3

Table 5: Survey responses from four groups of users who have independently used LegoSENSE for their own applications. Scores range from 1 (worst) to 10 (best).

6 EDUCATION AND COMMUNITY

In addition to the scenarios described in Section 5, we have begun to distribute LegoSENSE to a diverse set of users and research groups for a variety of projects. There are three groups (**G1** to **G3**) of researchers we have distributed LegoSENSE to incorporate into their own projects. **G1** is groups of researchers in the school of engineering who are working on creating systems for smarter homes and monitoring mental health using ambient sensors. These end users have extensive experience in programming and embedded platforms design. **G2** is researchers in the school of arts and sciences who are deploying LegoSENSE into numerous nursing homes to monitor the health of elderly citizens and the spread of COVID-19. These users come from a scientific background, but do not have any experience in embedded platform design. **G3** includes 4 developers (graduate school students and researchers) who are interested in designing their own module in addition to `M1` to `M8` provided by LegoSENSE.

Additionally, during several workshops within our local community, we used LegoSENSE to teach middle school students the basics of programming, IoT, and machine intelligence (**G4**). During each 40 minute workshop, we allowed students to program several applications, including detecting changes in temperature, fall detection, and air quality monitoring. In total, so far, approximately 50 students have used LegoSENSE to develop various of applications. We also measured the average number of applications each student was able to successfully program in 40 minutes, with no prior background, which was 1.4. This is on average faster than an experienced user deploying an application using discrete sensors (Section 5).

We surveyed all four groups (engineers, scientists, middle school students, and developers) about their experience in using LegoSENSE for their use cases using the same matrices described in Section 5.2. The average results are tabulated in Table 5. We see that even among middle school students, LegoSENSE was very easy to use and reliable, averaging close to 8.7 in all areas. We see that the engineers (**G2**) scored customizability the lowest out of the three groups because LegoSENSE only supports eight sensors, and their work focuses heavily on exploring a wide range of sensing modalities in a variety of contexts.

However, the developers, **G4**, finds it's easily to follow LegoSENSE's open-source documentation to design and develop their own sensor modules. For example, one developer customized a soil moisture and a soil pH sensor for her smart irrigation research. And she continuously operated 4 sets of LegoSENSE with `M1`, `M5`, and her sensors boards for 6 weeks. Two developers came up with sensor modules for assembly line quality monitoring. The sensor module development time for every developer varies according to their previous experiences in circuit design. But they all agree that "*instructions are easy to follow*" and "*way quicker than building from scratch*". We are currently expanding our library of available sensors and have open-sourced the design of LegoSENSE to allow anyone from the community to make any additional sensors compatible with LegoSENSE. In light of the aforementioned scenarios, it is possible to continuously and organically expand the number of sensors and actuators that integrate with LegoSENSE,

while simultaneously ensuring that the software and hardware of the open-sourced platform remain up-to-date.

7 DISCUSSION

Yet another Hardware Platform? Many hardware systems in our research community are built on open-source hardware designs provided by electronics retailers such as SparkFun [12] and Adafruit [21]. Some of them also have their own "plug and play" connection system to simplify the hardware connection and hands-on tutorials to help users set up the sensors. However, as discussed in Section 2, these systems have limited support for simultaneously active sensors and still require users to program. On the other hand, LegoSENSE is designed to be expandable and highly integrated. LegoSENSE is a generic platform that works for most existing sensors in the market by supporting most common communication protocols. We also designed and implemented complete hardware and software for LegoSENSE to support "plug and play" and "mix and match" right out of the box. By adopting an open-source approach, we hope to encourage the community to contribute to the platform's hardware capabilities and software functionalities. We believe that our open-source initiative have the potential to create an ecosystem of collaborative innovation, allowing for more rapid progress and wider adoption.

LegoSENSE Ecosystem. As we open-source LegoSENSE, we envision people and/or companies designing and publishing their own modules. The more modules developed, the more versatile LegoSENSE is, and the more applications LegoSENSE can serve. With LegoSENSE ecosystem developed, users can access/sense the world by developing their own, or purchasing the desired modules and snapping them onto the carrier board. LegoSENSE lowers the barrier for lay persons and researchers in other fields, who don't have expertise in IoT and sensing platform deployment, for their own applications and research. In addition, we are working on creating an online forum for developers and users to exchange their designs and brainstorm on new modules.

Integration with Other Applications. LegoSENSE can be directly applied to other applications as a tool and can be used as a data acquisition platform as described in Section 5. Moreover, since LegoSENSE is based on Raspberry Pi [14] with Raspberry Pi OS (formerly Raspbian, a Debian-based operating system for Raspberry Pi), people can also run software application with just LegoSENSE. Web dashboard for end user is an example that benefits from the real OS platform. Besides general-purpose input/output, Raspberry Pi also offers several other ports such as USB and HDMI. LegoSENSE can be further used for prototyping IoT edge devices. For example, by connecting a thermal camera and Google accelerator [27], it is possible to reproduce a LegoSENSE-version of temperature monitoring system for COVID.

Power Consumption. The increased power consumption of LegoSENSE on top of a system built with discrete components on a Raspberry Pi comes entirely from additional circuitry to enable the plug-and-play functionality: the multiplexer on the carrier board and EEPROMs on the modules (both 10s of μA). The Raspberry Pi draws current on the order of Amps. Because both the multiplexer and EEPROM on the carrier board are orders of magnitude less than the current draw from the Raspberry Pi, the increased power consumption is negligible compared to the benefits that LegoSENSE provides. Even lower power-consuming microcontrollers, such as those from the Arduino ecosystem, draw on order of tens of mA when active; if LegoSENSE was implemented on these lower-power microcontrollers, it would still add negligible power consumption. However, the primary goal of LegoSENSE is to enable ease of use for any person, regardless of their technical background. As such, the benefits provided by Raspberry Pi, a hardware platform with a real OS that can support the software system in Section 3, outweighs its high power consumption.

Limitations and Future Work (i) *Support for actuators.*: The current LegoSENSE ecosystem supports and can be easily expanded to a wide variety of sensors. We also plan to support modules with different actuators, and add interfaces for users to control them with minimum setup. We also plan to implement easily programmable connections and triggers between sensors and actuators for fully automated systems. (ii) *Coexist with other shields*: Although LegoSENSE ecosystem supports a wide variety of sensors, in order to build on the modular nature of LegoSENSE, users may have existing Raspberry Pi hats/shields with sensors they want to use. There are multiple reasons a user may still want to use the hat/shield: the module for the same sensor may not exist yet; users simply do not want to invest in additional funds; the device, such as a network adapter or a GPS receiver, does not fit in the surface area designated to a module. We plan to support the configuration where LegoSENSE carrier board coexists with other shields. Specifically, a configuration where the hat sits on top of the Raspberry Pi, and the LegoSENSE carrier board sits on top of the hat. We plan to integrate various Raspberry Pi shields into LegoSENSE's software system and treat them the same as a LegoSENSE module.

8 CONCLUSION

The Internet of Things has led to the tremendous growth in intelligent sensors and their applications in a wide range of domains. However, designing and deploying such sensing systems are often expensive, time-consuming, and/or require a strong technical background. In this work, we present LegoSENSE, an open and modular platform for enabling rapid deployment of customized sensing solutions. LegoSENSE does not require users to perform coding, hardware design, or wiring, which greatly improves its usability and accessibility over existing modular sensing solutions. Through a series of real-world deployments and user studies, we show that LegoSENSE reduces the development time of applications and sensor deployments, compared to configuring existing sensing solutions, by up to nine times.

ACKNOWLEDGMENTS

This research was partially supported by the National Science Foundation under Grant Numbers CNS-1704899, CNS-1815274, CNS-1943396, CNS-1837022, and CMMI-2218809, as well as COGNISENSE, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Columbia University, NSF, SRC, DARPA, or the U.S. Government or any of its agencies.

REFERENCES

[1] Advanced Monolithic Systems, Inc. August 2009. *Datasheet: AMS1117 1A Low Dropout Voltage Regulator.* Advanced Monolithic Systems, Inc. http://www.advanced-monolithic.com/pdf/ds1117.pdf

[2] Amphenol ICC August 2020. *Datasheet: Amphenol FCI 91911-31321LF.* Amphenol ICC. https://cdn.amphenol-cs.com/media/wysiwyg/files/drawing/91900.pdf

[3] Amphenol ICC August 2020. *Datasheet: Amphenol FCI 91931-31121LF.* Amphenol ICC. https://cdn.amphenol-cs.com/media/wysiwyg/files/drawing/91900.pdf

[4] ams OSRAM January 2020. *Datasheet: TCS3472FN Color Light-to-Digital Converter with IR Filter.* ams OSRAM. https://ams.com/documents/20143/36005/TCS3472_DS000390_3-00.pdf

[5] Arduino. 2022. Arduino. https://www.arduino.cc/. Accessed: 2022-01-25.

[6] ASAIR 2020. *Datasheet: AHT20 Temperature and Humidity Sensor.* ASAIR. https://cdn-learn.adafruit.com/assets/assets/000/091/676/original/AHT20-datasheet-2020-4-16.pdf

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

[8] Ingrid V. E. Carlier, Denise Meuldijk, Irene M. van Vliet, Esther M. van Fenema, Nic J.A. van der Wee, F.G. Zitman, Frans Zitman, and Frans G. Zitman. 2012. Routine outcome monitoring and feedback on physical or mental health status: evidence and theory. *Journal of Evaluation in Clinical Practice* (2012). https://doi.org/10.1111/j.1365-2753.2010.01543.x

[9] David Culler, Jason Hill, Mike Horton, Kris Pister, Robert Szewczyk, and Alec Wood. 2002. Mica: The commercialization of microsensor motes. *Sensors (Apr. 1, 2002)* (2002), 1–5.

[10] Daniel de Godoy, Bashima Islam, Stephen Xia, Tamzeed Islam, Rishikanth Chandrasekaran, Yen-Chun Chen, Yen-Chun Chen, Yen-Chun Chen, Shahriar Nirjon, Peter R. Kinget, and Xiaofan Jiang. 2018. PAWS: A Wearable Acoustic System for Pedestrian Safety. *null* (2018). https://doi.org/10.1109/iotdi.2018.00031

[11] N. Edmonds, D. Stark, and J. Davis. 2005. MASS: modular architecture for sensor systems. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* 393–397. https://doi.org/10.1109/IPSN.2005.1440955

[12] SparkFun Electronics. 2022. Sparkfun Sensors. https://www.sparkfun.com/categories/23. Accessed: 2022-01-25.

[13] Noura Farra, Bilal El-Sayed, Nadine Moacdieh, Hazem Hajj, Ziad Hajj, and Rachid Haidar. 2011. A Mobile Sensing and Imaging System for Real-Time Monitoring of Spine Health. *Journal of Medical Imaging and Health Informatics* (2011). https://doi.org/10.1166/jmihi.2011.1034

[14] Raspberry Pi Foundation. 2022. Raspberry Pi. https://www.raspberrypi.com/. Accessed: 2022-01-25.

[15] FUSE. 2020. FUSE - The Linux Kernel documentation. https://www.kernel.org/doc/html/latest/filesystems/fuse.html. Accessed: 2022-04-25.

[16] fusepy. 2018. fusepy: Simple ctypes bindings for FUSE. https://github.com/fusepy/fusepy.

[17] Goertek March 2016, Revised January 2018. *Datasheet: SPL06-001 Digital pressure sensor.* Goertek. https://datasheet.lcsc.com/lcsc/2101201914_Goertek-SPL06-001_C2684428.pdf

[18] Sean Harte, Brendan O'Flynn, Rafael V. Martinez-Catala, and Emanuel M. Popovici. 2007. Design and implementation of a miniaturised, low power wireless sensor node. In *2007 18th European Conference on Circuit Theory and Design.* 894–897. https://doi.org/10.1109/ECCTD.2007.4529741

[19] Kaiyuan Hou, Yanchen Liu, Peter Wei, Chenye Yang, Hengjiu Kang, Stephen Xia, Teresa Spada, Andrew Rundle, and Xiaofan Jiang. 2022. A Low-Cost In-situ System for Continuous Multi-Person Fever Screening. *International Symposium on Information Processing in Sensor Networks* (2022). https://doi.org/10.1109/ipsn54338.2022.00009

[20] Kaiyuan Hou, S. Xia, Junyi Wu, Minghui Zhao, Emily Bejerano, and Xiaofan Jiang. 2022. AI Stethoscope for Home Self-Diagnosis with AR Guidance. *ACM International Conference on Embedded Networked Sensor Systems* (2022). https://doi.org/10.1145/3560905.3568082

[21] Adafruit Industries. 2022. Adafruit Sensors. https://www.adafruit.com/category/35. Accessed: 2022-01-25.

[22] Blaise Kelly, Danilo Hollosi, Philippe Cousin, Sergio Leal, Branislav Iglar, and Andrea Cavallaro. 2014. Application of Acoustic Sensing Technology for Improving Building Energy Efficiency. *Procedia Computer Science* (2014). https://doi.org/10.1016/j.procs.2014.05.474

[23] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. 2005. *TinyOS: An Operating System for Sensor Networks.* Springer Berlin Heidelberg, Berlin, Heidelberg, 115–148. https://doi.org/10.1007/3-540-27139-2_7

[24] Yanchen Liu, Jingping Nie, Stephen Xia, Jiajing Sun, Peter Wei, and Xiaofan Jiang. 2022. SoFIT: Self-Orienting Camera Network for Floor Mapping and Indoor Tracking. *International Conference on Distributed Computing in Sensor Systems* (2022). https://doi.org/10.1109/dcoss54816.2022.00029

[25] Yanchen Liu, S. Xia, Jingping Nie, Peter Wei, Zhan Shu, Jeffrey Andrew Chang, and Xiaofan Jiang. 2022. AiMSE: Toward an AI-Based Online Mental Status Examination. *IEEE pervasive computing* (2022). https://doi.org/10.1109/mprv.2022.3172419

[26] Yanchen Liu, Minghui Zhao, Stephen Xia, Eugene Wu, and Xiaofan Jiang. 2022. A sensorless drone-based system for mapping indoor 3D airflow gradients: demo abstract. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services.* 634–635.

[27] Google LLC. 2022. USB Accelerator. https://coral.ai/products/accelerator/. Accessed: 2022-01-25.

[28] David G Luglio, Maria Katsigeorgis, Jade Hess, Rebecca Kim, John Adragna, Amna Raja, Colin Gordon, Jonathan Fine, George Thurston, Terry Gordon, et al. 2021. PM 2.5 concentration and composition in subway systems in the Northeastern United States. *Environmental health perspectives* 129, 2 (2021), 027001.

[29] Maxim integrated January 2015. *Datasheet: MAX31855 Cold-Junction Compensated Thermocouple-to-Digital Converter.* Maxim integrated. https://datasheets.maximintegrated.com/en/ds/MAX31855.pdf

[30] Konstantin Mikhaylov and Martti Huttunen. 2014. Modular wireless sensor and Actuator Network Nodes with Plug-and-Play module connection. In *SENSORS, 2014 IEEE.* 470–473. https://doi.org/10.1109/ICSENS.2014.6985037

[31] Konstantin Mikhaylov, Tomi Pitkaäho, and Jouni Tervonen. 2013. Plug-and-Play Mechanism for Plain Transducers with Wired Digital Interfaces Attached to Wireless Sensor Network Nodes. *Int. J. Sen. Netw.* 14, 1 (sep 2013), 50–63. https://doi.org/10.1504/IJSNET.2013.056336

[32] Modern Device 2022. *Modern Device Wind Sensor.* Modern Device. https://moderndevice.com/product/wind-sensor/ Accessed: 2022-01-25.

[33] Jingping Nie, Yanchen Liu, Yanchen Liu, Yigong Hu, Yuanyuting Wang, Stephen Xia, Matthias Preindl, Xiaofan Jiang, and Xiaofan Jiang. 2021. SPIDERS+: A light-weight, wireless, and low-cost glasses-based wearable platform for emotion sensing and bio-signal acquisition. *Pervasive and Mobile Computing* (2021). https://doi.org/10.1016/j.pmcj.2021.101424

[34] Jingping Nie, Minghui Zhao, Stephen Xia, Xinghua Sun, Hanya Shao, Yuang Fan, Matthias Preindl, and Xiaofan Jiang. 2022. AI Therapist for Daily Functioning Assessment and Intervention Using Smart Home Devices. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems.* 764–765.

[35] Brendan O'Flynn, S. Bellis, K. Delaney, J. Barton, S. C. O'Mathuna, Andre Melon Barroso, J. Benson, U. Roedig, and C. Sreenan. 2005. The Development of a Novel Minaturized Modular Platform for Wireless Sensor Networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks* (Los Angeles, California) *(IPSN '05).* IEEE Press, 49–es.

[36] Plantower June 2016. *Datasheet: PMS5003 Digital universal particle concentration sensor.* Plantower. https://www.aqmd.gov/docs/default-source/aq-spec/resources-page/plantower-pms5003-manual_v2-3.pdf

[37] Armin Ronacher. 2022. Flask Documentation. https://flask.palletsprojects.com/en/2.1.x/. Accessed: 2022-01-25.

[38] Sensirion The Sensor Company August 2017. *Datasheet: SGP30 Sensirion Gas Platform.* Sensirion The Sensor Company. https://www.mouser.com/pdfdocs/Sensirion_Gas_Sensors_SGP30_Datasheet_EN-1148053.pdf

[39] JJS Technical Services. 2022. Honeywell Analytics AirAlert 96d Multi-Channel Readout Gas Detection Controller with Data Logger - AA96D-DLC. https://www.jjstech.com/aa96d-dlc.html.

[40] JJS Technical Services. 2022. Honeywell Analytics FS7 Multi-Spectrum Fire and Flame Detector, Non-Latching Alert and Alarm Relays,Viton O-ring - FS7-2173-2RP. https://www.jjstech.com/fs7-2173-2rp.html.

[41] Shimmer. 2022. Shimmer Consensys. https://shimmersensing.com/. Accessed: 2022-01-25.

[42] Libelium Comunicaciones Distribuidas S.L. 2022. Libelium IoT Solutions. https://www.libelium.com/. Accessed: 2022-01-25.

[43] STMicroelectronics April 2022. *Datasheet: VL53L1X, A new generation, long distance ranging Time-of-Flight sensor based on ST's FlightSense™ technology.* STMicroelectronics. https://www.st.com/resource/en/datasheet/vl53l1x.pdf Version 7.0.

[44] STMicroelectronics October 2017. *Datasheet: 1-Kbit and 2-Kbit serial I²C bus EEPROMs.* STMicroelectronics. https://www.st.com/resource/en/datasheet/m24c02-r.pdf

[45] TDK April 2015. *Datasheet: ICS-40180 RF-Hardened, Low-Noise Microphone with Bottom Port and Analog Output.* TDK. https://invensense.tdk.com/wp-content/uploads/2015/02/DS-000021-v1.22.pdf

[46] TDK February 2015. *Datasheet: MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Devices.* TDK. https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[47] Texas Instruments January 2018. *Datasheet: ADS111x Ultra-Small, Low-Power, I2C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator.* Texas Instruments. https://www.ti.com/lit/ds/symlink/ads1115.pdf

[48] Texas Instruments November 2019. *Datasheet: TCA9548A Low-Voltage 8-Channel I2C Switch with Reset.* Texas Instruments. https://www.ti.com/lit/ds/symlink/tca9548a.pdf

[49] Mélodie Vidal, Jayson Turner, Andreas Bulling, and Hans Gellersen. 2012. Wearable eye tracking for mental health monitoring. *Computer Communications* (2012). https://doi.org/10.1016/j.comcom.2011.11.002

[50] Tianyu Wang, Giuseppe Cardone, Antonio Corradi, Lorenzo Torresani, and Andrew T Campbell. 2012. Walksafe: a pedestrian safety app for mobile phone users who walk and talk while crossing roads. In *Proceedings of the twelfth workshop on mobile computing systems & applications*. 1–6.

[51] Peter Wei, Stephen Xia, and Xiaofan Jiang. 2018. Energy saving recommendations and user location modeling in commercial buildings. In *Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization*. 3–11.

[52] Stephen Xia, Rishikanth Chandrasekaran, Yanchen Liu, Chenye Yang, Tajana Rosing, Xiaofan Jiang, and Xiaofan Jiang. 2021. A Drone-based System for Intelligent and Autonomous Homes. *ACM International Conference on Embedded Networked Sensor Systems* (2021). https://doi.org/10.1145/3485730.3492881

[53] Stephen Xia and Xiaofan Jiang. 2020. PAMS: Improving Privacy in Audio-Based Mobile Systems. *AIChallengeIoT@SenSys* (2020). https://doi.org/10.1145/3417313.3429383

[54] Wei-Ying Yi, Kwong-Sak Leung, and Yee Leung. 2018. A Modular Plug-And-Play Sensor System for Urban Air Pollution Monitoring: Design, Implementation and Evaluation. *Sensors* 18, 1 (2018). https://doi.org/10.3390/s18010007

[55] Andrea Zanella, Nicola Bui, Angelo Castellani, Aldo Castellani, Lorenzo Vangelista, and Michele Zorzi. 2014. Internet of Things for Smart Cities. *IEEE Internet of Things Journal* (2014). https://doi.org/10.1109/jiot.2014.2306328

[56] Minghui Zhao, Yanchen Liu, Avik Dhupar, Kaiyuan Hou, Stephen Xia, and Xiaofan Jiang. 2022. A modular and reconfigurable sensing and actuation platform for smarter environments and drones: demo abstract. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 626–627.