SensorViz: Visualizing Sensor Data Across Different Stages of Prototyping Interactive Objects

Yoonji Kim[1,2], Junyi Zhu[1], Mihir Trivedi[1], Dishita G Turakhia[1], Ngai Hang Wu[1], Donghyeon Ko[1], Michael Wessely[1], Stefanie Mueller[1]
MIT CSAIL[1], Chung-Ang University[2]
yoonji.h.kim@gmail.com,{junyizhu;mihirt;dishita;crazywu}@mit.edu,donghyeon.ko@kaist.ac.kr,
{wessely;stefanie.mueller}@mit.edu

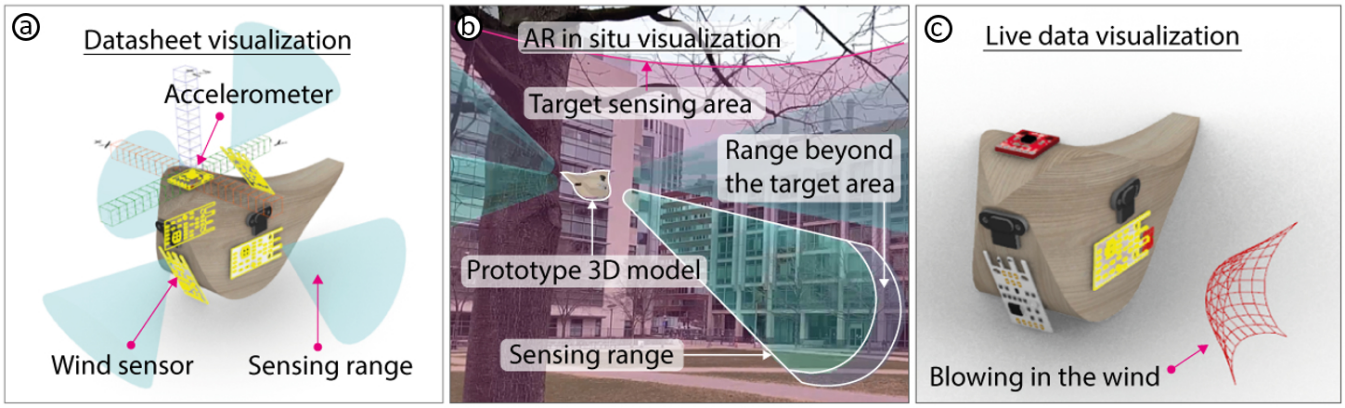

Figure 1: (a) Before buying sensors, makers can visualize sensor data from the datasheet to get a first idea of what sensors can sense. (b) Before physically building the prototype, makers can visualize sensor data in AR to see what sensors can sense in the context in which the prototype will be used. (c) After assembling the physical prototype, makers can visualize live data either in the 3D editor or via AR to verify that the sensors work as expected and to make further changes as needed.

ABSTRACT

In this paper, we propose SensorViz, a visualization tool that supports novice makers during different stages of prototyping with sensors. SensorViz provides three modes of visualization: (1) visualizing datasheet specifications before buying sensors, (2) visualizing sensor interaction with the environment via AR before building the physical prototype, and (3) visualizing live/recorded sensor data to test the assembled prototype. SensorViz includes a library of visualization primitives for different types of sensor data and a sensor database builder, which once a new sensor is added automatically creates a matching visualization by composing visualization primitives. Our user study with 12 makers shows that users are more effective in selecting sensors and configuring sensor layouts using SensorViz compared to traditional prototyping utilizing datasheets and manual testing on the prototype. Our post hoc interviews indicate that SensorViz reduces trial and error by allowing makers to explore sensor positions on the prototype early in the design process.

DIS '22, June 13–17, 2022, Virtual Event, Australia

ACM ISBN 978-1-4503-9358-4/22/06…$15.00
https://doi.org/10.1145/3532106.3533481

CCS CONCEPTS

• **Human-centered computing** → **User interface toolkits**.

KEYWORDS

sensor visualization, electronic prototyping, personal fabrication

ACM Reference Format:
Yoonji Kim[1,2], Junyi Zhu[1], Mihir Trivedi[1], Dishita G Turakhia[1], Ngai Hang Wu[1], Donghyeon Ko[1], Michael Wessely[1], Stefanie Mueller[1]. 2022. SensorViz: Visualizing Sensor Data Across Different Stages of Prototyping Interactive Objects. In *Designing Interactive Systems Conference (DIS '22), June 13–17, 2022, Virtual Event, Australia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3532106.3533481

1 INTRODUCTION

Over the past decades, the availability of sensors has increased substantially and enabled makers and designers to prototype interactive objects rapidly and at low cost. However, this large variety of components also comes with the challenge of selecting the right sensor from many similar components to find the one that best fits the use case at hand. Since each sensor has its own specification, understanding what each sensor can sense and how it will work on a prototype can be a time-consuming process.

In today's workflow, makers have two options to increase their understanding of how sensors behave. First, makers can consult each sensor's datasheet. However, extracting information from a datasheet can be cumbersome and confusing since the sensing information may be buried among the chip design, schematic, and other technical aspects. In addition, datasheets are not standardized, which makes comparing sensors with each other a difficult task. Furthermore, since datasheets are text documents, it can be hard to picture what sensors can sense when placed on a prototype.

Alternatively, makers can physically explore what sensors can sense by wiring them up. However, this comes with its own challenges since makers have to buy different sensors upfront to test which one works best for their prototype. They have to write code for each sensor before they can start assessing if the sensors are suitable. Further, the sensor data is either represented as printed text in a terminal or as a graph in a plotter tool, and not as spatial information in 3D.

Several tools for prototyping robot and aircraft applications can visualize sensor data in 3D. Gazebo in ROS [27] and SensorFusion [29], for instance, allow users to define sensors and their 3D position via scripts and then visualize the resulting sensor data. BlenSor [14] takes this a step further by allowing users to interact directly with the sensors without the need for programming. However, after defining the sensors, users have to recompile the 3D environment to see the visualization results, which does not allow for real-time exploration. While users can import 3D models, users cannot edit their geometry while the sensor visualization is on-going. Finally, while users can create custom sensors via scripts, users have to create their own visualizations for each sensor.

In this paper, we present SensorViz, a visualization tool that supports novice makers by visualizing sensor information at each stage of the prototyping process. Makers can change both the sensor layout and the prototype 3D model and see the visualization update in real-time. In the early stages of design, SensorViz supports makers in deciding which sensors to select and how to lay them out by visualizing information from the sensor's datasheet. Before makers fabricate the prototype, they can use SensorViz's AR visualization to see the prototype, sensors, and sensor data in the context of the environment. Finally, once makers assemble the prototype, they can visualize live sensor data to verify that the sensors work as expected. SensorViz automatically generates the visualizations for different types of sensor data based on its library of visualization primitives. Thus, to add a new sensor, users only have to provide its specification and a matching visualization is generated automatically. Note that the focus of our paper is on the visualization of sensor data. Generating mounts, splitting geometry, and routing wires has been done in prior work and is not our contribution.

In summary, we contribute:

- a formative study with 12 makers to understand challenges makers face when using sensors for interactive prototypes;
- a visualization tool that supports makers during the different stages of prototyping by visualizing sensor data from the datasheet, overlaying data onto the environment via AR, and displaying live/recorded usage data;
- a library of visualization primitives for different types of sensor data, and a sensor database builder, which once a new sensor is added automatically creates a matching visualization by composing visualization primitives;
- a user study with 12 participants showing that SensorViz significantly speeds up the prototyping process ($F_{(1,10)} = 7.61, p < 0.05$) while the resulting sensor layouts have significantly better coverage ($F_{(1,10)} = 62.61, p < 0.001$) of the target sensing area.

2 RELATED WORK

Our work is related to how makers prototype interactive objects with respect to placing electronics on 3D object geometries, and how makers are supported today in visualizing sensor information.

2.1 Prototyping Interactive Objects

HCI researchers recognized early that "prototyping is a key activity within the design of interactive systems" [8]. However, its complexity in programming and circuit design also comes with challenges for makers and designers (Booth et al. [5]). To assist makers with programming, several tools automatically generate code based on high-level input provided via a component diagram that the maker draws (Trigger-Action-Circuits [1]), via a configuration interface that the maker selects hardware behavior criteria (Adapt2Learn [31]), or via a conversation the maker has with a virtual agent (HeyTeddy [16]). To facilitate hardware prototyping, researchers developed tools that help makers iterate on circuit layouts before physically building them (VirtualComponent [17], VirtualWire [20]). In addition, tools, such as AutoFritz [22] and SchemaBoard [18], support makers in creating circuits by providing guidance where to place components onto the breadboard. To support beginners in learning how to assemble circuits, simplified hardware modules (BitBlox [10]) can also direct the user how to connect components. While these systems support makers in building circuitry and sensors, users still have to estimate sensor properties, such as the sensor's range by reading complicated sensor datasheets. In contrast, *SensorViz* enables an intuitive understanding of the sensor's capabilities with visualizations situated on the digital prototype.

2.2 Integrating Electronics with 3D Prototypes

Several researchers have explored how to facilitate prototyping of electronics in the context of a 3D model's geometry, for instance, by automatically packing the electronics inside the prototype and modifying it to contain mounts for the electronics (Ashbrook et al. [2]). SurfCuit [32] automatically creates a layout of electronic components based on the schematic and then generates the fabrication files that contain grooves for the wiring. PHUI-kit [15] lets makers place electronic components on the surface of a 3D model and generates custom mounts for holding them in place. Similarly, Plain2Fun[33] and MorphSensor [36] support users in placing electronic components onto 3D models and generate custom fabrication files. CurveBoards [35], in contrast, allow makers to assemble electronic components directly on the prototype by integrating breadboards into the prototype's surface. While all of these tools support makers in integrating electronics with prototypes, they do not visualize information related to the electronic components.

2.3 Visualization in Prototyping and Situated Visualization

Several tools visualize information related to electronic components. One line of work focuses on helping makers understand the internal state of their circuits. Spark [4] and LightUp [9] are tools that visualize moving electrons as dots in AR, which can help makers find wiring mistakes. Similarly, ToastBoard [11] and CurrentViz [34] visualize voltages and current flows by overlaying the information onto the breadboard. While these tools help makers debug their circuits, they are not designed to support makers in choosing which sensor is appropriate for their prototype. Other work visualizes sensor data as 2D graphs either for testing circuits (Scanalog [28], Bifröst [23]) or for finding thresholds for detecting user interaction (Astral [19]). In addition, TinkerCAD [3] provides a separate 2D electronic editor that shows information such as a distance sensor's field of view in 2D. However, these tools do not show spatial information to help makers place sensors on prototypes.

Another body of research explores situated visualization [7], e.g. in the context of pervasive displays [12], and AR prototyping [21]. In this field, several works explore how to support makers in better understanding what sensors can sense when placed on a prototype by visualizing spatial sensor data. TinkerCAD [3], for instance, provides a separate 2D electronic editor that shows information such as a distance sensor's field of view in 2D. To contextualize sensor data in a 3D environment, Radu et al. [25] use AR to overlay 3D graphics that visualize magnetic fields around coils and magnets. In Virtual Makerspaces [24], the authors also visualized circuit signals in AR. However, the work uses predefined graphics and is set up as a learning environment and not for prototyping.

Finally, expert tools for prototyping robot and aircraft sensing applications, such as Gazebo in ROS [27] and SensorFusion [29], have been developed. However, these tools are developed for engineering applications and require users to write scripts to define the sensors' positions and geometry. BlenSor [14] facilitates this process by allowing users to directly interact with the sensors. However, users have to recompile the environment after every change, which does not allow for real-time exploration. Further, users cannot modify the 3D model while the sensor visualization is on-going. Finally, while users can create custom sensors via scripts, users have to create their own visualizations for each sensor.

3 FORMATIVE STUDY

To better understand the issues novice makers face when using sensors to develop prototypes, we interviewed 12 makers (9m, 3f, aged 23-29 (M=25.6, SD=2.5)) who are students with engineering or design backgrounds from our institution. They had novice experience in prototyping with sensors (3-5 previous projects) but did not consider themselves experts and had not taken a class from us before. The participants reflected on their prior experiences with sensors via semi-structured interviews (30 min each) with 10 standardized questions. Participants were compensated with 10 USD in local currency. We analyzed our interviews by transcribing the audio recordings and then conducting open and axial coding. Participants shared that they encountered many difficulties when using sensors:

Difficulty in imagining what sensors can sense: Several makers stated that it was difficult to estimate what a sensor can sense if a prototype has not yet been built. P12 explained: "I made a light that moves according to a person's posture. I designed the model in several parts and printed it out. After I assembled it, I found that sometimes the distance sensor sees the motor-driven joint when the joint moves. The distance sensor's viewing angle was wider than I thought." P3 also stated that she had difficulties estimating the position of sensors due to the scale of the prototype, saying: "I made an interactive curtain that recognizes gestures, and it took a lot of time to attach the sensors, test, and adjust them. [...] Because of its large size, it was difficult to imagine where and how to install the sensors before building the prototype." P4 said that they had experienced a delay in the prototyping process since they had selected the wrong sensor and had to purchase a different one: "I first needed to see the sensor data log [...] to better understand what the sensor's can sense. I could not do other work until the newly ordered sensor arrived."

Re-printing time due to sensor position adjustment: Multiple makers stated that because they had difficulties picturing what the sensors can sense, they followed a trial-and-error process in which they repeatedly modified both the sensors' positions and the prototype. P8 said: "I walked around with the prototype to get raw data. [...] It didn't work as expected. I kept relocating the sensors and modified the model several times, which were cumbersome tasks." Modifying the 3D model and reprinting the prototype, however, slowed down the prototyping process. As P10 reported: "to change the sensor's position, it is often necessary to reprint the 3D object, but as the 3D printing time is long, the overall prototyping time is longer and more materials are used.". Some makers reported that they tried to use simulation tools. P2 had used MATLAB but stated "it was not much help for modeling and laying out the sensors on the prototype in real-time". P1 noted: "it [MATLAB] might be good for simulating sensor signals, but not suitable for prototyping."

Impact of prototype geometry on sensor choice and placement: Participants also pointed out that the prototype geometry influenced which sensors they chose and where they placed them. P11 stated: "I made a children's toothbrush for my graduation project. But when I printed it, the toothbrush handle was smaller than I thought, so I swapped the touch sensor with a photoresistor."

Using the fewest sensors to reduce possible errors: Several participants mentioned that they tried to use a minimum number of sensors to reduce cost and to prevent the circuit from becoming too complex. P11 stated: "I usually consider lowering the complexity of the circuit. Using more components increases the possibility of bugs both for software and hardware. So I try to cover the sensing area with fewer sensors. However, this is always accompanied by worries about blind spots."

Difficulties when selecting sensors based on datasheets: Several participants had difficulties when comparing sensors from different manufacturers. P12 commented: "It isn't easy to find what I need because a lot of data is explained [in words]. Moreover, the document format is all different for each manufacturer, so it is complicated." P6 reported that he only selects popular sensors because

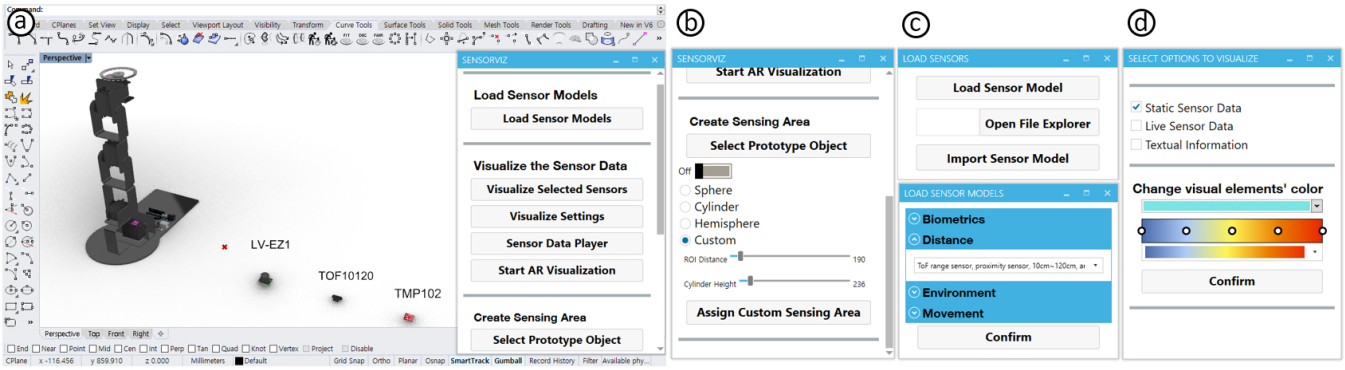

Figure 2: SensorViz user interface. (a) and (b) are the SensorViz main window. (b) Users can select the shape, size, and distance for the target sensing area. (c) Users can select and import sensor models in the list. (d) Users can select the visualization mode for static, real-time, textual, and color of the visualization.

it is easy to find information in online discussion boards, even if those sensors are not ideal for the use case.

In summary, the findings from our formative study highlight that makers experience difficulties when using sensors to create interactive prototypes due to missing contextualized information of what sensors see and how they fit onto the prototype geometry. We thus designed the SensorViz toolkit to help makers better understand what a sensor can sense in context of the environment and the object's geometry. To accomplish this, we developed a library of visualization primitives for different types of sensor data. The visualizations are overlaid either on the digital prototype in a 3D editor or the physical prototype in the real world through an AR visualization. Our sensor database builder allows makers to import sensors into our visualization environment without the need to read a datasheet. SensorViz also provides live and recorded sensor data visualization, which eases the difficulties of resolving errors by allowing users to test and debug their prototype in situ.

4 SENSORVIZ

SensorViz (Figure 2) is a visualization tool that supports novice makers during the different stages of prototyping with sensors by visualizing sensor data alongside the prototype geometry, both in a digital 3D editor and via an AR overlay. Makers define the position of their sensors by placing them onto the 3D model of the prototype. When makers change either the model or the sensors, they see the sensor visualization update in real-time, which allows for interactive exploration and fast design iteration to decide which sensors to use, how to lay them out, and how to adjust the model geometry.

4.1 Visualizations for Different Stages of Prototyping with Sensors

SensorViz is designed to support makers through three types of visualizations for different stages of prototyping with sensors.

Sensor Specifications from Datasheet: At the beginning of the prototyping process before makers buy sensors, SensorViz helps them to explore which sensors are most suitable for their prototype, how many they need, and where to place them by visualizing information from the sensors' data sheets, such as the min/max range, field of view, and resolution, in the 3D editor together with the prototype. Makers only have to select the sensor from the SensorViz database to load it into the 3D editor and see its sensor visualization. When makers change the position of the sensor on the virtual prototype, the visualization automatically updates in real-time. SensorViz also displays how well the sensors cover a specified sensing area. Makers define the area they want to sense; SensorViz then computes the intersection of the sensors' field of view with the area. While makers often have to guess the sensor's capabilities from abstract schematics and tables in its datasheet before buying the sensor, SensorViz offers a 3D visualization directly from the datasheet that enables makers to get a spatial understanding of the sensor's range and functionality in the early stages of prototyping.

Sensor Data Overlay over the Physical Prototype via AR: To support makers in evaluating their prototype in context before physically building it, SensorViz provides an AR overlay that can be displayed using either a tablet, smartphone, or head-mounted AR display (e.g., HoloLens). Makers start by opening the AR application (Fologram [13]) on their AR device. In the SensorViz UI, makers then select the 'Start AR' option to generate a QR code. The QR code can then be scanned with the AR application to synchronize the mobile AR device with the computer to start streaming data from SensorViz. The AR overlay shows the virtual prototype, the virtual sensors, and the corresponding sensor visualization. To ensure the correct size of the prototype and sensor visualization in AR, the AR application has a built-in feature to detect the size of objects in the environment and then scales the prototype in AR accordingly. To position the prototype in AR, the AR application automatically detects the ground plane in the environment, which SensorViz then sets to be equal to the ground plane in the 3D editor. The AR overlay is synchronized with the 3D editor, i.e., changes in the 3D editor are reflected in the AR overlay in real-time. Makers can toggle the visualization of each sensor on or off through the AR interface. Once makers fabricate the prototype and buy the sensors, they can use the AR overlay to help with assembly: The position of sensors is shown in the AR overlay, makers thus only have to align

the physical sensors with the virtual sensors to create a matching physical sensor layout.

Live and Recorded Sensor Data Visualization: To support makers in testing if their physical prototype works, SensorViz provides makers with functionality to visualize live data. Makers download the custom Arduino code (.ino) from the SensorViz editor and then upload it to an Arduino board connected to the sensor. Makers then select the sensor in SensorViz and choose the port from which the Arduino is streaming the sensor data. When SensorViz receives live data through the port, it visualizes it. SensorViz also allows makers to record data first before visualizing it. This can be useful for prototypes for which data collection may take a long time or that require data collection in the wild where makers do not have access to the 3D editor. To record sensor data, makers prepare the sensor using the same steps as for live data but use the 'record data' button in the SensorViz data recorder, which is implemented as a separate program that can run on a portable computer (Raspberry Pi). The recorded data is then saved as a file. Makers can then replay the recorded data using SensorViz to see it visualized. By replaying different parts of the data, makers can test how well the prototype works across different interaction scenarios.

The three visualizations support makers in the different stages of prototyping with sensors from exploring which sensors to use and how to lay them out before building the prototype, to further refining the sensor layout while visualizing the prototype in AR in the context in which it will be used, and finally streaming live data or collecting recorded data to verify that the prototype works as intended.

4.2 Prototyping Walkthrough

We next illustrate SensorViz's visualizations through the example of prototyping a wind chime in the form of a bird that hangs from a tree. We want to create a wind chime that uses changes in wind speed and wind direction to modulate sound; further, when a person is walking close to it (within 1.5 meters), it should play a special melody. The wind chime should also sense temperature, which we will display on an LED. We chose this example for our walkthrough since our target users are novice makers, such as students in an introductory electronics class who build simple prototypes. For instance, 300 students in our class used on average 2.38 sensors across 72 group projects. Schemaboard [18] also showed that 5,083 making projects used a median of 7 components, which included not only sensors but also buttons. The final prototype from our walkthrough has 6 sensors, which matches the complexity of the prototypes found in the study.

Trade-Off Between Different Sensors: We utilize wind sensors to sense wind speed and wind direction. In the SensorViz toolbar, we see that the wind sensor has a 60° field of view. Since we want to sense wind from any angle (360°), we select 6 wind sensors. We turn on the datasheet visualization, which shows each sensor's field-of-view, and use it to determine where to place the wind sensors to avoid overlapping sensing areas. We find that the geometry of the wind chime does not allow us to place the sensors to achieve full coverage (Figure 3a). We modify the geometry of

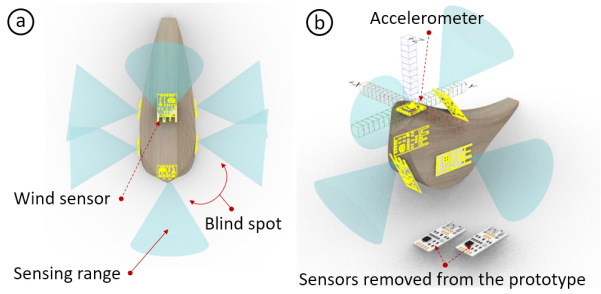

Figure 3: Placing sensor models and visualizing sensing range. (a) We place six wind sensors on the wind chime geometry, but still have blind spots. (b) We thus decide to remove two wind sensors and add an accelerometer to detect overall motion.

the wind chime but find it compromises the aesthetics too much. We therefore decide to use only 4 wind sensors, one for each cardinal wind direction, and to compensate for the 'blind spots' with an accelerometer that will detect motion caused by different wind speeds (Figure 3b).

Comparing Different Sensor Resolutions: When selecting the accelerometer, we see that there are three options: 0.01/mG, 0.4/mG, and 0.6/mG sensing resolution. Since we want the wind chime to be sensitive to even minor movements when the wind blows, we choose the accelerometer with the highest resolution. We then place the accelerometer onto a free spot on the wind chime.

Ensuring Sensing Area Coverage: Next, we choose a set of distance sensors to detect if a person is walking within 1.5 meters. Before positioning the sensors, we create a new sensing area around the wind chime by toggling on the 'Create Sensing Area' button. We select the cylinder from the list of sensing volumes and set its scale to a radius of 1.5m. As soon as we place each distance sensor on the wind chime, SensorViz visualizes which part of the sensing area each distance sensor covers (Figure 4a).

Modifying Prototype Geometry: While positioning one of the distance sensors, we notice that the wind chime geometry, i.e., the bird's tail, interferes with its field of view (Figure 4b). To solve

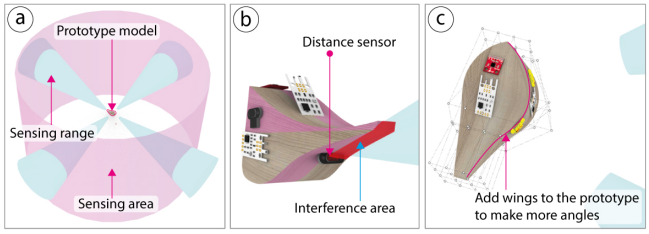

Figure 4: Visualizing sensing area coverage and interference: (a) Visualized sensing range and target area in SensorViz. (b) Finding overlaps between sensing range and prototype object. (c) Modifying prototype geometry to add angled surfaces for mounting the sensors in a tilted position.

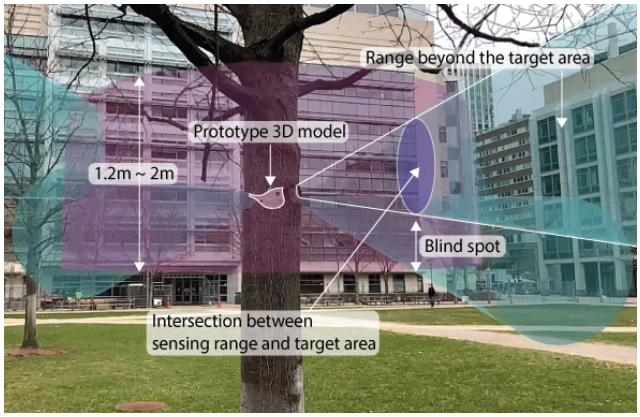

Figure 5: The AR visualization of our virtual wind chime hanging on the tree shows that the distance sensors cover would only detect adults and not small children walking by.

this issue, we first bend the bird's tail upward and then also curve the geometry by adding wings to the bird (Figure 4c), which allows us to tilt the distance sensor to avoid interference. We encounter a similar issue when placing the temperature sensor. We want to place the temperature sensor under the bird's tail to completely cover it to avoid direct sunlight. However, we notice that the tail is too narrow to fully cover the sensor. We therefore make the tail wider. To finalize our prototype, we also add a speaker for playing the sounds and an LED that we will use to display temperature.

Visualizing Virtual Prototype in Physical Context: Before we fabricate the wind chime, we visualize what its sensors can sense via AR in the context of the tree onto which it will be hung. We first change the position of our wind chime in the 3D editor to be at the level of the tree branch (1.5m). We then use our handheld tablet, open the AR application, and scan the QR code generated by SensorViz to sync our tablet with the SensorViz 3D editor. We hold our tablet up in front of the tree and confirm the detected ground plane, which then positions our wind chime at the height of the branch. Based on the AR visualization, we see that we need to adjust the angle of the distance sensors to point them further down to be able to also detect children within the sensor's field of view (Figure 5). We make the changes in SensorViz and confirm via AR that the sensing coverage is now appropriate.

Mounting Physical Sensors Based on Virtual Sensor Positions: After 3D printing the wind chime, we move on to mount the physical sensors onto the 3D printed prototype. We take each physical sensor and align it with the matching virtual AR sensor overlay.

Using Live Data: After we assemble the wind chime, we want to verify that it works as intended by interacting with it and visualizing the live data from the sensors. We first upload the code for the sensors onto a microcontroller and then select the port for streaming data in SensorViz. We then hang the wind chime outside onto the tree and walk around it. Since there is no wind, we blow air onto the wind chime from different directions. The distance sensors and wind sensors record data as expected, but we notice that the accelerometer values do not change much.

Using Recorded Data: Since we are unsure if the accelerometer behaves differently when the wind chime is exposed to actual wind rather than us blowing air on it, we decide to record sensor data for a full day using the SensorViz data recorder. Replaying the data later confirms that the accelerometer does pick up on the wind as intended. The recorded data also shows that the wind is coming mostly from two directions. We therefore take a note that for another prototype iteration, we will remove the other wind sensors to save cost and reduce the complexity of wiring.

5 LIBRARY OF VISUALIZATION PRIMITIVES

To visualize sensor information in a coherent manner, SensorViz contains a library of visualization primitives (Figure 6) that can be composed into more complex visualizations for various sensors.

5.1 Visualization Attributes

Dimensionality: We represent discrete data as a point and continuous data as a bar. We represent directional data as bars in different directions. For volumetric data, we display the data as a 3D volumetric shape (e.g., a cone or hemisphere). Dimensional information is rendered as a black outline with no infill.

Range: To visualize the range of a sensor, we use the min/max values from the datasheet as the bounds of the visualization. For continuous non-spatial and directional data, we define the lower bound of the visualization to be equal to the minimum value and the upper bound of the visualization to be equal to the maximum value specified in the datasheet. Thus, as a result, the bar of a temperature sensor with a range of 0-30°C is half as long as for a sensor with 0-60°C. To ensure that the visualization is not too large

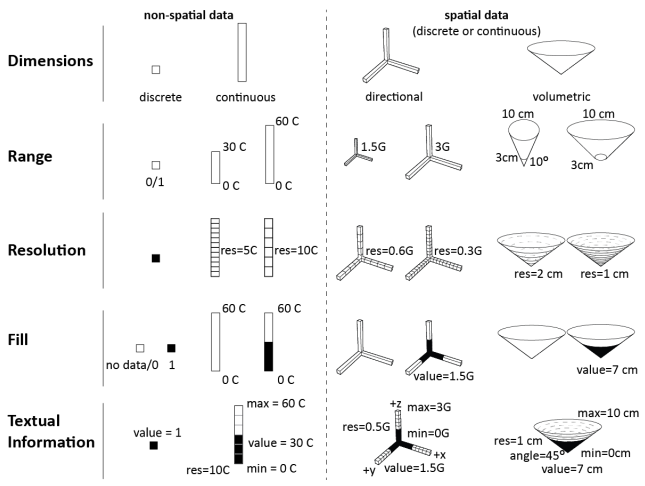

Figure 6: To visualize sensor information in a coherent manner, we created a library of visualization primitives. These primitives represent non-spatial and spatial data using attributes such as dimensions, range, resolution, fill, and textual information.

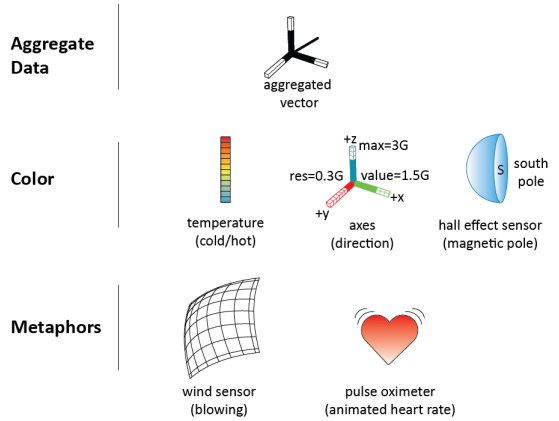

Figure 7: In addition to the visualization primitives, SensorViz also supports makers with additional visualization attributes, such as aggregate data, color, and metaphors.

and obstructing the view, SensorViz applies a scaling factor to the length of the bar that by default is the same for all sensors with the same measurement unit (e.g., a scaling factor of 0.1 for °C leads to visualizations of 3cm for 0-30°C and 6cm for 0-60°C). Makers can override the default scaling factor as needed. For spatial data, the visualization shows the lower and upper bounds of the sensing volume in the size as specified in the datasheet.

Resolution: We visualize resolution by splitting the bar or volume into segments, with smaller segments representing higher sensor resolutions than larger segments. We compute the segment size by dividing the sensor's output bit resolution by the range of the sensor. For instance, a 12-bit temperature sensor with 4096 different values for its sensor readings and a sensing range from -55 to 150°C has a resolution of 205/4096°C = 0.05°C. To ensure that the segments have an appropriate size in the visualization, SensorViz applies a scaling factor that by default is the same for all sensors that use the same measurement unit (e.g., a scaling factor of 2 for °C leads to visualizations with 40 segments (2/0.05) for the 12-bit sensor). Makers can override the scaling factor as needed.

Sensor Values: We visualize live sensor values by filling in the point, bar, or volume. For discrete values, if a 'zero' is read, the point is rendered in 0% opacity and if a 'one' is read, it is rendered in 100% opacity. Makers can differentiate between a 'zero' reading and 'no data' using the text label that shows the current value. For continuous values, the larger the received sensor values, the more of the bar or the volume is rendered in 100% opacity.

Textual information: We use labels to display the range (min/max), the resolution, and the current sensor value. For directional data, we also label the axes information, and for volumetric data, we label the angle of the field of view. Makers can choose in the user interface which information they want to display.

5.2 Additional Visualizations Attributes

In addition to the basic visualization primitives, SensorViz offers additional visualizations to support makers.

Aggregate Data: Many sensors measure multiple sensor values, which when aggregated provide higher-level information. For instance, accelerometers provide three separate values for acceleration in x, y, z but for many use cases seeing the aggregate value, i.e., the orientation in which the object is actually moving, is more helpful. SensorViz therefore provides a visualization that combines the three individual axis measurements into one directional vector.

Color Coding: Where appropriate, SensorViz uses color to facilitate makers' understanding of the sensor data and to help makers avoid mistakes. For instance, for temperature sensors, SensorViz colors the sensor data in blue for low temperatures and in red for high temperatures. To reduce potential mistakes when makers use digital hall effect sensors, SensorViz colors them in blue because they can only detect the red South pole of magnets. Finally, to prevent makers from confusing axes when prototyping with accelerometers, SensorViz colors the axes using the common coloring scheme of red for the x, green for the y, and blue for the z-axis.

Metaphors: Finally, metaphors can help makers assess what a sensor is for and if it works as expected. In SensorViz, makers have the option to activate metaphors for certain sensors to visualize high-level behavior. For instance, wind sensors in SensorViz are

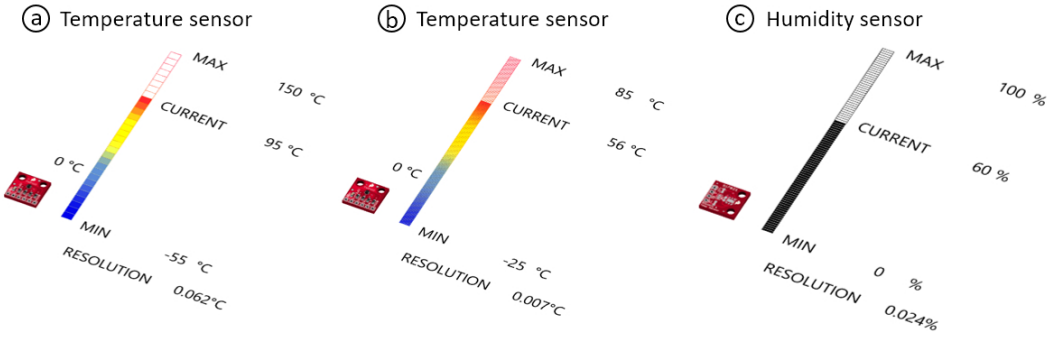

Figure 8: Composition of sensor data visualization for non-spatial sensors: (a,b) Two different temperature sensors, (c) humidity sensor.

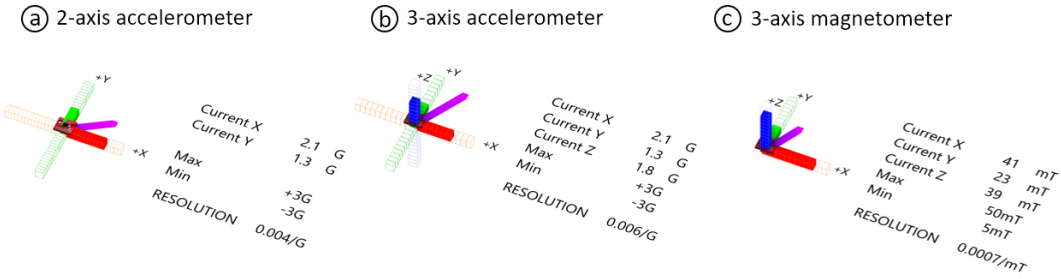

Figure 9: Composition of sensor data visualization for directional sensors: (a) 2-axis accelerometer, (b) 3-axis accelerometer, (c) 3-axis magnetometer.

visualized as a sail that blows in the wind when sensors report data, and pulse oximeters are visualized as a beating heart.

5.3 Composition of Visualization Primitives

Using the visualization primitives, SensorViz can represent various types of sensors.

Non-Spatial Sensors (Temperature/Humidity): Figure 8 shows the visualization of different non-spatial sensors, such as temperature and humidity sensors, all using the same visualization primitives. Since the analog data of the sensors is non-spatial, SensorViz represents it as a bar. Each sensor's bar is split into a number of segments based on the resolution of the sensor data. The incoming sensor signals are represented as opaque areas on the bar. The min/max values of the sensor and the currently read values are added as text labels to the bar. The humidity sensor uses the default visualization, i.e., displays sensor data in black (Figure 8b), while the temperature sensor data is colored in blue/red for cold/hot temperatures (Figure 8a).

Directional Sensors (Accelerometers/Magnetometers): Figure 9 shows the visualization of different directional sensors, such as 2-axis and 3-axis accelerometers and magnetometers, all using the same visualization primitives. The analog data for each of the axes is represented as a bar pointing in the direction from which the data is retrieved. Each axis is split into a number of segments based on the incoming data. The incoming sensor signals are represented on each axis as opaque overlays. The axis information (x,y,z), min/max possible readings, and the currently read value are added as text labels to the visualizations. The combined data is shown as an aggregate vector and each axis is colored based on its direction.

Volumetric Sensors (Hall Effect, Distance, Wind Sensor): Figure 10 shows different types of volumetric sensors, i.e., a hall effect, distance, and wind sensor. All sensors are represented with sensing volumes, i.e., the hall effect sensor as a sensing hemisphere, and the distance and wind sensor as sensing cones. For the hall effect and the distance sensor, the incoming data is visualized directly on the sensing volume, i.e., for the discrete hall effect sensor the hemisphere is either opaque or translucent (Figure 10a), and for the analog distance sensor the cone is made opaque up to the level of the sensor value (Figure 10b). While by default, the data from the wind sensor is also visualized on the cone, we override the default and display it as a bar since the amount of wind is non-spatial and thus does not refer to a specific distance from the sensor. The angle or radius of the sensor, min/max, and the currently read value are added as text labels to the visualizations. The hall effect sensor is colored 'blue' to show that it only senses the 'red' South pole.

5.4 Sensor Database Builder

Sensor visualizations are automatically created based on the information about each sensor in the SensorViz sensor database. The database currently has 19 sensors, which correspond to common sensors used by novice makers (i.e., 300 students in our class used only 19 different sensors across 72 group projects although they could buy any sensor). All the sensors in the database are saved in a

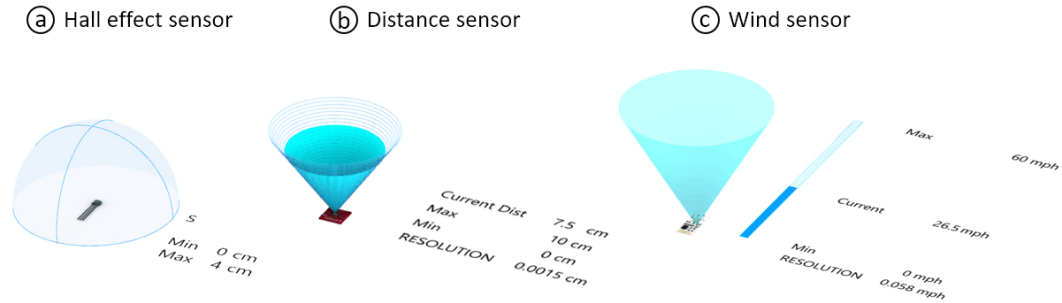

Figure 10: Composition of sensor data visualization for spatial sensors: (a) hall effect sensor, (b) distance sensor, (c) wind sensor.

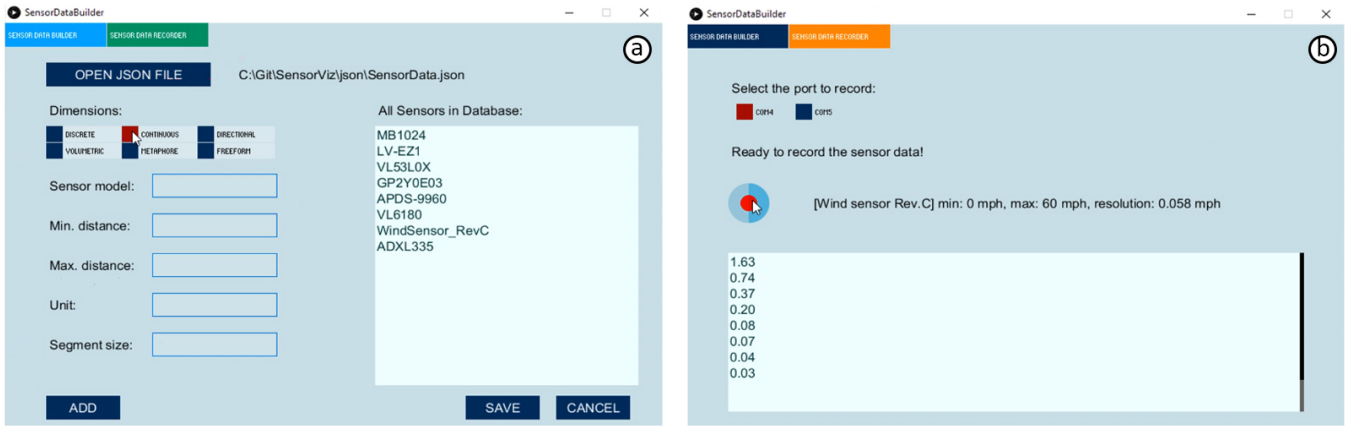

Figure 11: Sensor Database Builder: (a) Adding information from the datasheet. (b) Recording sensors' live data.

single file (.json). The file also contains a reference to the program (.ino) that is uploaded to an Arduino connected to the sensor for live data visualization.

Sensor Database Builder: To facilitate adding new sensors, we built a sensor database builder (Figure 11a). It is targeted at expert makers who are able to understand the information from the datasheet and supports them populating the database by specifying the require data that should get extracted from the datasheet. Once a maker added a sensor to the database, it becomes available in the SensorViz 3D editor. The sensor database can be shared with other makers by copying the .json file that contains all sensors and the .ino files that contain the code for each sensor. We next describe in more detail how expert makers can use the sensor database builder to add new sensors to SensorViz:

Adding Information from the Datasheet: To add a sensor to the SensorViz database, makers have to define if the sensor is a discrete/continuous sensor, if it contains directional information (x,y,z) or senses a volume (i.e., has an angle for its field of view). In addition, makers have to specify the range of the sensor data, the resolution of the sensor, and which textual information should be displayed. Furthermore, they need to specify if color coding should be used and if they want to provide a custom metaphor.

Automatically Generated Visualization: Once the maker saves the information, SensorViz automatically adds the sensor to the .json file and afterwards shows the sensor in the SensorViz sensor list. When makers load the sensor into the 3D editor, the visualization is automatically created by matching the information from the database to the visualization primitives. For the sensor's 3D model, SensorViz first searches its 3D model collection and if the sensor is not available, it uses a default sensor model. Makers can also specify a sensor 3D model file in case they have it available.

Writing the Code for Live Data Visualization: Makers can extend the visualization of the new sensor to include live data (Figure 11b). For this, makers first have to write the code for the sensor using SensorViz's code template (.ino). The template contains a returnSensorData() function, which prints the live sensor data to the serial monitor, allowing SensorViz to retrieve the data. When printing data to the serial monitor, makers need to prefix the sensor data with the attribute name from the .json file. Using this convention, both raw sensor values and aggregate data can be reported back to SensorViz. If sensors have multiple variables (accelerometer), SensorViz uses a ',' as data separator. Once makers upload the program, the live data visualization for the new sensor becomes available in SensorViz.

6 IMPLEMENTATION

Figure 12 shows the SensorViz system workflow. SensorViz builds on the MorphSensor 3D editing environment [36] in Rhino3D and is implemented as a Grasshopper plugin. To show SensorViz visualizations in AR, we use the Fologram plugin for Rhino3D. Our sensor data recorder is implemented as an executable file that can run on a portable computer (Raspberry Pi). The sensor database builder is implemented in Processing.

Loading Sensor Information from Sensor Database: When makers open the SensorViz editor, SensorViz retrieves all the sensor information from the SensorViz sensor database by parsing the JSON file. It also retrieves the sensor 3D model by matching the 3D model name in the JSON file with the sensor model titles in the Sparkfun 3D Model Component Library [26].

Visualizing Sensor Data: After loading the sensor specification, SensorViz creates the visualization by matching the sensor attributes with the visualization primitives. To position non-spatial data next to the sensor, SensorViz retrieves the 3D model's center and then offsets the visualization accordingly. To position spatial data, such as sensing volumes, in the direction the sensor is facing, SensorViz retrieves the normal vector of the plane of the sensor model. This information is provided by the Sparkfun 3D model library and always faces forward relative to the orientation of the sensor. To compute how to segment the visualization according to the sensor resolution, SensorViz first computes the sensor's resolution as described in section 'Visualization Attributes' and then maps it onto the bar or volume. To visualize the current sensor

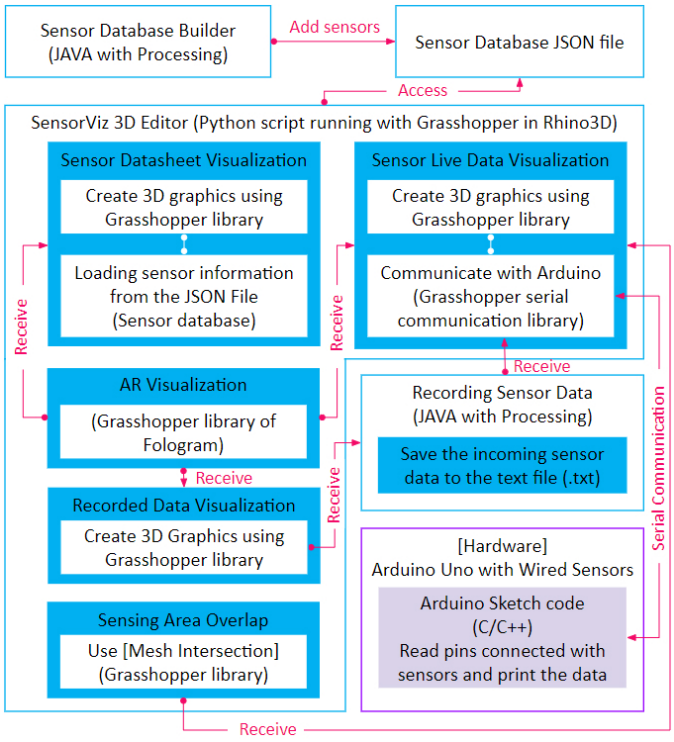

Figure 12: SensorViz system workflow.

value, SensorViz creates an additional geometry on top of the existing visualization primitive and sets its height to be the current sensor value and its appearance to be opaque. SensorViz displays text as 3D text labels and places them at different offsets from the sensor's center, based on the type of label. To color a visualization, SensorViz connects the geometry of the visualization to the 'Gradient' attribute in Grasshopper. To visualize metaphors, SensorViz takes as input a custom 3D geometry and then connects it to an animation that is driven by the sensor value.

AR visualization: To start the AR visualization in Fologram [13] from the SensorViz user interface, we wrote a custom python script. To visualize sensor data from Grasshopper in AR, SensorViz connects each visualization primitive's geometry to Fologram's 'Sync' module. Using 'Sync,' Fologram then transfers each change in the 3D editor to the AR overlay in real-time.

Reading Live Sensor Values: To receive live data from the microcontroller, SensorViz uses Grasshopper's Firefly plugin, which reads data from the serial monitor. Once data is incoming, SensorViz analyzes its prefix and then connects it to the corresponding attribute in the sensor's visualization. Finally, SensorViz connects the attribute to the geometry of the visualization primitive to display the changing data values.

Recording Sensing Data: When data needs to be recorded, SensorViz writes the incoming sensor values into a text file (.txt) together with a timestamp of when the data was received. To visualize the recorded data later, SensorViz reads the text file and visualizes the sensor data at the recorded time intervals.

Sensing Area Overlap: To compute how much a sensor's field of view and the sensing area overlap, SensorViz iterates over all sensors and uses Grasshopper's mesh intersection function to determine the amount of coverage.

7 USER STUDY

We conducted a user study to understand how SensorViz's visualization can help makers during the different stages of prototyping with sensors. We compared prototyping with the SensorViz visualizations (datasheet visualization, AR overlay, live data visualization) to a baseline condition, in which the SensorViz editor was provided but all visualizations were turned off and the participants had access to the sensors and their datasheets.

Participants: We recruited twelve novice makers, 10 male and 2 female, aged 24-30 years (M=26.8, SD=2.3), who are students with industrial design backgrounds from our institution. All of them had some experience prototyping with sensors, i.e., had completed 2-5 previous projects but did not consider themselves experts. Participants were compensated with 10 USD in local currency.

Conditions: In the SensorViz condition, participants were given the SensorViz 3D editor with all visualizations enabled, i.e., they were able to access the datasheet visualization, sensor data overlay over the physical prototype via AR, and live data visualization. In the baseline condition, participants were given the SensorViz editor with all visualizations turned off. In both conditions, participants were able to place digital sensor 3D models on the virtual prototype geometry. The study was run as a between-subjects study, and participants were randomly assigned to a condition.

Task: Participants were asked to build a smart lamp that can automatically turn on/off and direct its light toward the user. The lamp we gave participants already consisted of a robotic arm as the stand to orient the light and a ring-shaped LED strip for turning on/off the light. Participants were asked to extend the prototype to add four features: (1) the lamp orients itself to the user when the user is within 2 meters to the lamp, (2) the lamp lights up when the user is within a distance of 1 meter to the lamp, (3) the lamp gets brighter when the room gets darker, and (4) the light color changes according to the temperature in the room. To avoid falsely detecting the table, we asked participants to augment the lamp to sense within a range of 76cm - 150cm height (Figure 13a). In addition, since the lamp was sitting on a desk that was in a corner of a room, it only needed to sense the area towards the front and right side (90 degrees) (Figure 13b). For prototyping their solution, participants were asked to determine the best sensor layout, i.e., the layout that used the fewest sensors and had the best coverage.

We provided participants with six different distance sensors (APDS-9960, GP2Y0E03, LV-EZ1, TOF10120, VL53l0X, and VL6180), a temperature sensor (TMP102), and a photoresistor (CdS-5528). To reflect the different prototyping stages, we first simulated selecting sensors before buying them and before having them physically available. For this, the control group had access to regular datasheets

and the SensorViz group had access to the SensorViz datasheet visualization. After this, we simulated that the sensors had arrived and participants were now allowed to physically place sensors and access the live data visualization on the Serial monitor (control group) or for the SensorViz group inside the SensorViz editor and via AR overlay. Participants decided how they wanted to split their time between using the digital 3D editor and building the physical prototype. For the final deliverable, we asked participants to mirror their final design both on the digital and the physical prototype.

Resources: Participants in both conditions were provided with datasheets of each sensor. In addition, participants were allowed to search for information on the internet. For digital prototyping, we prepared a digital scene with the 3D model of the lamp on a table set up against the corner of a room that mirrored the physical setup. We also provided the eight different sensors in the SensorViz editor. For physical prototyping, we provided the physical lamp and the eight different sensors, three copies of each in case participants needed multiple sensors of the same type. The sensors were already wired to one microcontroller each (Arduino Uno) and had the code uploaded for providing both live data in the SensorViz editor and alternatively the Arduino serial monitor. To see multiple sensors' data at the same time, all Arduinos were connected simultaneously to a USB hub and participants were able to see the live data of each sensor by selecting the serial port for each Arduino. In the SensorViz condition, participants used a tablet in case they wanted to visualize sensor data via AR.

Study Procedure: We gave participants a short introduction to the editor they were using in their assigned condition and showed them the available resources. We explained the task with a picture that showed the target sensing area. Participants then built their prototypes for up to 120 minutes. Participants were allowed to end the task anytime they were satisfied with their prototype. At the end, we conducted a 30-minute semi-structured interview. The experiment took 2.5 hours.

Data Collection: To design and prototype interactive objects, makers iterate between exploring, implementing, and testing. We collected data about the time efficiency of prototyping to evaluate if SensorViz enhances the prototype's quality by helping makers to prototype faster, which allows them to produce more iterations in the same amount of time. Referencing evaluation factors presented in previous works [16, 17], we measured task completion time, time spent for selecting sensors, and the number of times participants switched sensors during the prototyping process. Participants were allowed to exchange sensors throughout the prototyping process until they found the best sensor layout for the task. We also collected every participants' digital 3D model and documented the sensor layout on the physical prototype. To evaluate how well the sensor layout covers the target sensing areas (i.e., at 1m distance and 2m distance from the lamp), we compute the coverage of each of the two sensing areas in m^2 using the position of the sensors in the digital editor. We also measured if the placement of the light sensor picked up only the ambient light or was influenced by the lamp light, i.e., if the light sensor had the same value independent of the lamp being on/off. Finally, we measured if the temperature sensor picked up only the heat from the room or also heat generated by the LED light ring of the lamp and the motors from the robotic lamp stand.

Quantitative Results: We analyzed the raw data using one-way ANOVA tests followed by Bonferroni correction post hoc analysis with α=0.05. Results are shown in Figure 14.

Prototyping Speed: All participants completed the task within 120 minutes. Overall, participants in the SensorViz group were faster than the baseline group with an average completion time of 34m 34s (SD: 10m 57s) versus 1h 5m 40s (SD: 25m 21s) ($F_{(1,10)} = 7.61, p < 0.05$) (Figure 14d). For selecting an initial set of sensors to start the first prototyping round, participants in the SensorViz group were also faster than the baseline group with an average of 6m 46s (SD: 3m 27s) versus 33m 50s (SD: 21m 53s) ($F_{(1,10)} = 8.95, p < 0.05$) (Figure 14e).

Number of Iterations for Selecting Sensors: Participants in the SensorViz group needed to iterate less on which sensors to use, i.e. they replaced 1 sensor ($M = 0.17, SD = 0.41$) vs. the baseline group

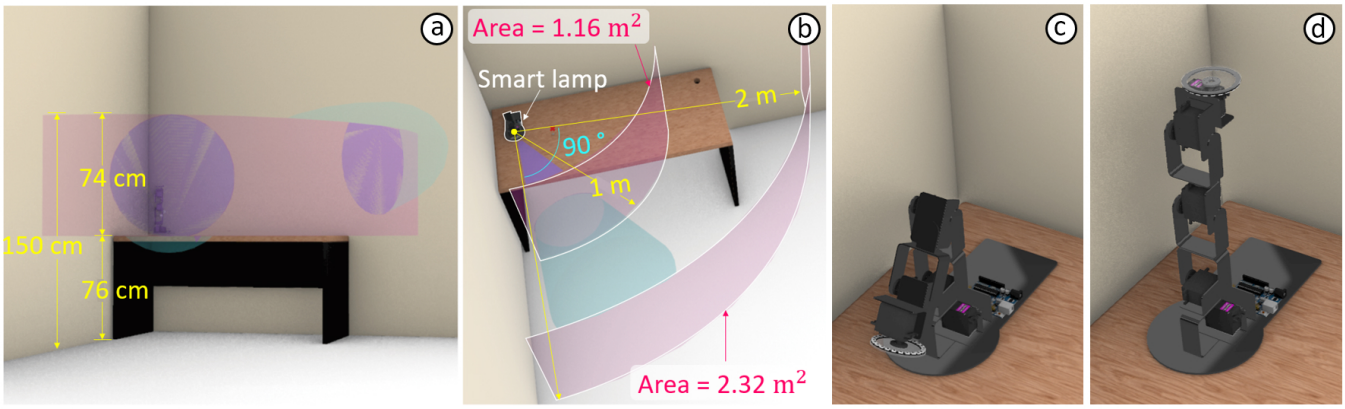

Figure 13: SensorViz evaluation study setup: (a) sensing area within a range of 76cm - 150cm height. (b) the lamp senses the area towards the front and right side (90 degrees). (c) the lamp's initial shape before sensing the user. (d) the lamp orients itself to the user when the user is within 2 meters to the lamp.

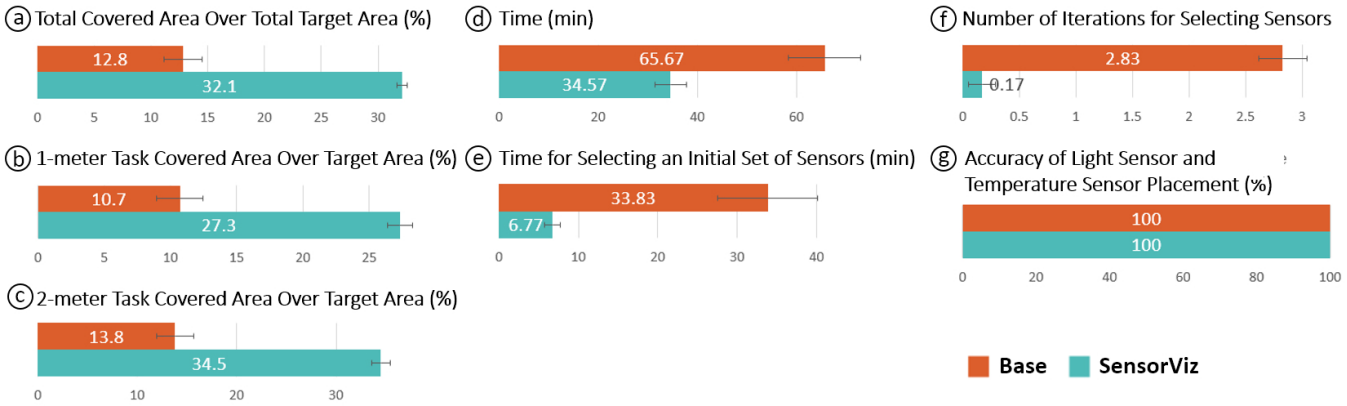

Figure 14: SensorViz study results: Overlap of sensing area with the sensor's field-of-view in (a) total, (b) 1-meter task, and (c) 2-meter task. Self-assessed (d) completion time and (e) initial sensor selection time. (f) Shows the number of iterations for selecting sensors and (g) the accuracy of the light sensor and temperature sensor placement.

replaced 17 sensors ($M = 2.83, SD = 0.75$) ($F_{(1,10)} = 58.18, p < 0.001$) (Figure 14f).

Coverage of Sensing Area: We measured the coverage of the two sensing areas (1 meter, 2 meters) with the distance sensors. In the 1-meter distance task (Figure 14b) where the target sensing area was $1.16m^2$, the SensorViz group had 27.34% coverage ($M = 0.32m^2, SD = 0.04m^2$) which is significantly better compared to the baseline group which had coverage of 10.74% ($M = 0.13m^2, SD = 0.07m^2$) ($F_{(1,10)} = 37.09, p < 0.001$). This was also true for the 2-meter task (Figure 14c) where the target sensing area was $2.32m^2$ with significant differences for the SensorViz group which had coverage of 34.49% ($M = 0.8m^2, SD = 0.08m^2$) vs. baseline group which had coverage of 13.79% ($M = 0.32m^2, SD = 0.15m^2$) ($F_{(1,10)} = 48.13, p < 0.001$). Thus, overall, SensorViz participants achieve a target sensing coverage of 32.1% vs. the baseline which achieved 12.8% (Figure 14a).

Number of Sensors: Though we did not restrict the number of sensors that participants could use, all the participants used the same number of sensors. Participants have used two sensors for the 1-meter task, two sensors for the 2-meter task, a temperature sensor, and a photoresistor.

Placement of Light Sensor and Temperature Sensor: We found that there was no difference in how accurately participants in both conditions placed the light and the temperature sensor. In both conditions, for all participants, the light sensor did not pick up any of the lamp light and the temperature sensor did not pick up any of the heat from the motors or the lamp (Figure 14g).

In summary, for the spatial distance sensors, SensorViz participants achieved 252% higher coverage as in the baseline condition (254.5% for 1 meter, and 250.1% for 2 meters), but for the non-spatial light and temperature sensors, there was no difference between SensorViz and the baseline. Finally, participants who used SensorViz placed sensors 190% faster than the baseline condition.

Qualitative Results: We analyzed our post-study interviews by transcribing the audio/video recordings (total: 3 h 35 min material) and then conducting open and axial coding. In the semi-structured interviews, we asked participants to reflect on their prototyping process, i.e., what was the most challenging part and how they overcame the difficulties. We also inquired about the use of each of the provided visualizations and other provided resources.

Simplifying the prototyping process by minimizing trial and error: Participants in the SensorViz group highlighted that SensorViz helped them to reduce the number of iterations on the physical prototype. For instance, P11 stated: "in the process of choosing a location, we can [...] virtually attach the sensor and test it, so it can greatly reduce the [physical] prototyping process that actually takes the most amount of time." Similarly, P2 stated: "before testing the sensor, the process to look at the document [data sheet] and choose it is long, and trial-and-error happens, but I think this [SensorViz] will allow trial-and-errors to decrease a lot."

Preventing Late Model Changes: Participants pointed out that it was helpful that they were able to modify the prototype geometry in the 3D editor while also seeing the sensor information. Participants noted that this feature was particularly useful to decide on sensor placement "before the prototype's shape is finalized" (P5) and that "placing the sensor and editing the model early on" (P5) helped to prevent further changes down the line when the prototype was already 3D printed. Another participant said "it was good that in the beginning of prototyping [...], we could edit the model while looking at the sensor location" (P7).

Beneficial to collaborative prototyping: Participants (P10, P11) also reported SensorViz could aid makers in collaborating with others, saying: "working with a team, we could end up dividing the hardware and model designs. But with this [SensorViz], we do not have to wait until the hardware is made, so it is helpful that we could try modeling first by visualizing the sensor in the software" (P10). P11 noted: "When collaborating, there are many times, especially when sharing the work of making the hardware, when you have to wait for the rest of it to be complete, but for this

[SensorViz] it was good that none of that was necessary and we could try it immediately."

Advantages of different sensor visualizations during prototyping: Participants mentioned that SensorViz's visualizations aided them in various ways in the prototyping process. P8 and P11 noted that SensorViz' datasheet visualization "was helpful [...] because we could see the sensor shape or the [sensing] area directly from the software" (P11) and allowed them to "easily see the blind spots" (P8). P8 also stated that the live data visualization was useful for testing sensors, saying: "In particular, when testing if this really detected in this area, the function that showed the live data showed the space immediately, so it was used very effectively". Participants who used the AR visualization to place sensors on the physical prototype reported that "It was nice to be able to attach it to the exact location [...] by looking at it using AR" (P7) and suggested using an AR headset instead of the tablet to have both hands free for mounting the sensors: "because it was a tablet, it was uncomfortable to try to attach it [the sensor] with one hand." Moreover, participants (P2, P8, P11) appreciated that the AR visualization assisted them in testing the sensors in context, saying: "what I liked most was the target area display using AR, which I think was the functionality I used most enthusiastically [...] I see 1 meter after setting it as the target area, and if the sensor detection area covers it or not [...] the test was much easier and precisely controllable" (P8). "Because the detection range is shown by AR, [...] I can check whether there is a collision between the sensor's FOV and target area [...] accuracy is quite important for shape-changing objects, so this visual feedback was very helpful" (P2).

Archiving the prototyping progress: P5 stated that SensorViz allows users to save the history of their prototyping progress, saying: "since this is a graphic editing tool, it seems like it would be good to archive the work in the middle."

Non-spatial sensors' visualization: Participants (P5, P7, P11) reported that the photoresistor and the temperature sensor's "data is simple enough to use the serial to look at it" (P5), still SensorViz was beneficial because "it is intuitive to have the information floated right above the sensor" (P11). P7 also mentioned the inconvenience of reading data from the serial monitor, saying: "the text moves quickly upward and the different sensors show up alternating, so it is sometimes difficult to follow."

In summary, our user study showed that SensorViz speeded up the prototyping process by minimizing trial and error in selecting and testing sensors. For the non-spatial sensors, though there was no significant difference between groups, still participants appreciated displaying the sensor data next to the sensor. Also, there is the opportunity of assisting users in collaborative prototyping and preventing late model changes by enabling users to modify the prototype geometry in the 3D editor while seeing the sensor visualization.

8 LIMITATIONS AND FUTURE WORK

There are several avenues to further improve SensorViz.

Validating the Design Space with Additional Sensors: Currently, SensorViz only supports specific types of sensors (temperature, humidity, distance, hall effect, and wind sensors, as well as accelerometers and gyroscopes). While these sensors are representatives of particular sensor categories (non-spatial, directional, and spatial sensors), adding more sensors will help to further validate the design space of visualization primitives. In addition, although the visualization primitives and visualization have been designed to support the most type of sensors used, the scalability of sensor visualization has not been tested. For future work, we will study the scalability of sensor visualization with our suggested sensor visualization primitives.

Automatically Processing Datasheets: When makers want to add a new sensor to SensorViz, they currently have to manually transfer the data from the datasheet. For future work, we plan to automatically extract this information from the pdfs of the datasheets.

Manual vs. Automatic Sensor Placement: We initially considered automating the sensor placement. However, we found that sensor placement is a task that requires user input since it also affects the design of the prototype. While algorithms can suggest optimal placement with respect to sensing coverage, taking aesthetics into account is difficult to automate. Algorithms would also require knowing the final object geometry, whereas makers may adjust sensors and object geometry in tandem to achieve the desired design.

Interactivity of AR visualization: While makers can visualize sensor data in AR and toggle the visualizations of each sensor on/off, adjustments to the sensor layout can only be made through the SensorViz 3D editor. For future work, we will add bidirectional interaction so that changes in AR are reflected in the 3D editor as well.

Visualizing Effects of Electronic Components on Sensors: Electronic components, such as capacitors added to sensors, can change the sensed values. We did not include a feature that shows the combined effect of electronic components since our target users are novice makers. For instance, all students in our introductory electronics class used standalone sensor modules that did not require extra filtering with capacitors.

Simulating Environmental Data: Our paper focuses on developing sensor visualizations for the different stages of prototyping. Since the visualization primitives for the sensors are the same for live data as for simulation, we did not include a simulation feature in our visualization tool. However, a feature for simulating sensor data can be added as future work.

Integrating Circuit Layout Functionality: Participants in our user study pointed out that they would like to not only place sensors but also to build the entire circuit as part of the editing process. For future work, we will add functionality that allows makers to place the sensors and also build the entire circuit on the 3D geometry (e.g., SurfCuit [32], MorphSensor [36]).

Social Aspects of Prototyping: Some participants in our formative study mentioned that they frequently asked other makers to solve issues collaboratively while prototyping with sensors. Previous HCI research explored the collaborative aspect for prototyping [6, 30]

and we plan to investigate in future work how to support collaborative prototyping with our system by facilitating conversation between makers about various sensor choices and placement options.

9 CONCLUSION

In this paper, we presented SensorViz, an interactive visualization tool that supports makers with different visualizations of sensor data throughout the various stages of the prototyping process. We discussed the results from our formative study with 12 makers that showed makers experience difficulties when creating prototypes using sensors. We then demonstrated how SensorViz addresses these challenges by visualizing information from the sensor's datasheet, overlaying sensor information, and providing live sensor data. We discussed how our library of visualization primitives, together with our sensor database builder, allows makers to add new sensors. We then reported results from our user study that showed that SensorViz significantly reduces prototyping time while also enabling makers to place sensors more effectively, in a way that they cover a larger portion of the target sensing area. For future work, we plan to improve the system by automatically parsing datasheets and enabling changes in the sensor layout directly in AR.

ACKNOWLEDGMENTS

This work has supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1C1C101196211).

REFERENCES

[1] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 331–342. https://doi.org/10.1145/3126594.3126637

[2] Daniel Ashbrook, Shitao Stan Guo, and Alan Lambie. 2016. Towards Augmented Fabrication: Combining Fabricated and Existing Objects. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 1510–1518. https://doi.org/10.1145/2851581.2892509

[3] AUTODESK. 2019. *TINKERCAD*. https://www.tinkercad.com/

[4] Elham Beheshti, David Kim, Gabrielle Ecanow, and Michael S. Horn. 2017. Looking Inside the Wires: Understanding Museum Visitor Learning with an Augmented Circuit Exhibit. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1583–1594. https://doi.org/10.1145/3025453.3025479

[5] Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. 2016. Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3485–3497. https://doi.org/10.1145/2858036.2858533

[6] Adrien Bousseau, Theophanis Tsandilas, Lora Oehlberg, and Wendy E. Mackay. 2016. How Novices Sketch and Prototype Hand-Fabricated Objects. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 397–408. https://doi.org/10.1145/2858036.2858159

[7] N Bressa, H Korsgaard, A Tabard, S Houben, and J Vermeulen. 2022. What's the Situation with Situated Visualization? A Survey and Perspectives on Situatedness. In *IEEE Trans Vis Comput Graph.* (Toronto, Ontario, Canada) *(vol. 28,1)*. 107–117. https://doi.org/10.1109/TVCG.2021.3114835

[8] Marion Buchenau and Jane Fulton Suri. 2000. Experience Prototyping. In *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques* (New York City, New York, USA) *(DIS '00)*. Association for Computing Machinery, New York, NY, USA, 424–433. https://doi.org/10.1145/347642.347802

[9] Joshua Chan, Tarun Pondicherry, and Paulo Blikstein. 2013. LightUp: An Augmented, Learning Platform for Electronics. In *Proceedings of the 12th International Conference on Interaction Design and Children* (New York, New York, USA) *(IDC '13)*. Association for Computing Machinery, New York, NY, USA, 491–494. https://doi.org/10.1145/2485760.2485812

[10] Kayla DesPortes, Aditya Anupam, Neeti Pathak, and Betsy DiSalvo. 2016. BitBlox: A Redesign of the Breadboard. In *Proceedings of the The 15th International Conference on Interaction Design and Children* (Manchester, United Kingdom) *(IDC '16)*. Association for Computing Machinery, New York, NY, USA, 255–261. https://doi.org/10.1145/2930674.2930708

[11] Daniel Drew, Julie L. Newcomb, William McGrath, Filip Maksimovic, David Mellis, and Björn Hartmann. 2016. The Toastboard: Ubiquitous Instrumentation and Automated Checking of Breadboarded Circuits. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 677–686. https://doi.org/10.1145/2984511.2984566

[12] Alix Ducros, Clemens N. Klokmose, and Aurélien Tabard. 2019. Situated Sketching and Enactment for Pervasive Displays. In *Proceedings of the 2019 ACM International Conference on Interactive Surfaces and Spaces* (Daejeon, Republic of Korea) *(ISS '19)*. Association for Computing Machinery, New York, NY, USA, 217–228. https://doi.org/10.1145/3343055.3359702

[13] Fologram. 2021. *Fologram*. https://www.fologram.com

[14] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. 2011. BlenSor: Blender sensor simulation toolbox. In *International Symposium on Visual Computing*. Springer, 199–208. https://doi.org/10.1007/978-3-642-24031-7_20

[15] Michael D. Jones, Zann Anderson, Casey Walker, and Kevin Seppi. 2018. PHUI-Kit: Interface Layout and Fabrication on Curved 3D Printed Objects. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3173574.3173684

[16] Yoonji Kim, Youngkyung Choi, Daye Kang, Minkyeong Lee, Tek-Jin Nam, and Andrea Bianchi. 2019. HeyTeddy: Conversational Test-Driven Development for Physical Computing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 4, Article 139 (dec 2019), 21 pages. https://doi.org/10.1145/3369838

[17] Yoonji Kim, Youngkyung Choi, Hyein Lee, Geehyuk Lee, and Andrea Bianchi. 2019. VirtualComponent: A Mixed-Reality Tool for Designing and Tuning Breadboarded Circuits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300407

[18] Yoonji Kim, Hyein Lee, Ramkrishna Prasad, Seungwoo Je, Youngkyung Choi, Daniel Ashbrook, Ian Oakley, and Andrea Bianchi. 2020. *SchemaBoard: Supporting Correct Assembly of Schematic Circuits Using Dynamic In-Situ Visualization*. Association for Computing Machinery, New York, NY, USA, 987–998. https://doi.org/10.1145/3379337.3415887

[19] David Ledo, Jo Vermeulen, Sheelagh Carpendale, Saul Greenberg, Lora Oehlberg, and Sebastian Boring. 2019. Astral: Prototyping Mobile and Smart Object Interactive Behaviours Using Familiar Applications. In *Proceedings of the 2019 on Designing Interactive Systems Conference* (San Diego, CA, USA) *(DIS '19)*. Association for Computing Machinery, New York, NY, USA, 711–724. https://doi.org/10.1145/3322276.3322329

[20] Woojin Lee, Ramkrishna Prasad, Seungwoo Je, Yoonji Kim, Ian Oakley, Daniel Ashbrook, and Andrea Bianchi. 2021. VirtualWire: Supporting Rapid Prototyping with Instant Reconfigurations of Wires in Breadboarded Circuits. In *Proceedings of the Fifteenth International Conference on Tangible, Embedded, and Embodied Interaction* (Salzburg, Austria) *(TEI '21)*. Association for Computing Machinery, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/3430524.3440623

[21] Germán Leiva, Jens Emil Grønbæk, Clemens Nylandsted Klokmose, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2021. Rapido: Prototyping Interactive AR Experiences through Programming by Demonstration. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 626–637. https://doi.org/10.1145/3472749.3474774

[22] Jo-Yu Lo, Da-Yuan Huang, Tzu-Sheng Kuo, Chen-Kuo Sun, Jun Gong, Teddy Seyed, Xing-Dong Yang, and Bing-Yu Chen. 2019. AutoFritz: Autocomplete for Prototyping Virtual Breadboard Circuits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300633

[23] Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. BifröSt: Visualizing and Checking Behavior of Embedded Systems across Hardware and Software. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 299–310. https://doi.org/10.1145/3126594.3126658

[24] Iulian Radu, Tugce Joy, and Bertrand Schneider. 2021. Virtual Makerspaces: Merging AR/VR/MR to Enable Remote Collaborations in Physical Maker Activities. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, Article 202, 5 pages. https://doi.org/10.1145/3411763.3451561

[25] Iulian Radu and Bertrand Schneider. 2019. What Can We Learn from Augmented Reality (AR)? Benefits and Drawbacks of AR for Inquiry-Based Learning of Physics. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300774

[26] Sparkfun. 2019. *Sparkfun 3D Model Component Library*. https://github.com/sparkfun/3D_Models

[27] Stanford Artificial Intelligence Laboratory et al. [n.d.]. *Robotic Operating System*. https://www.ros.org

[28] Evan Strasnick, Maneesh Agrawala, and Sean Follmer. 2017. Scanalog: Interactive Design and Debugging of Analog Circuits with Programmable Hardware. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 321–330. https://doi.org/10.1145/3126594.3126618

[29] Inc. The MathWorks. 1994. *Sensor Fusion and Tracking*. https://www.mathworks.com/products/sensor-fusion-and-tracking.html

[30] Austin L. Toombs. 2017. Hackerspace Tropes, Identities, and Community Values. In *Proceedings of the 2017 Conference on Designing Interactive Systems* (Edinburgh, United Kingdom) *(DIS '17)*. Association for Computing Machinery, New York, NY, USA, 1079–1091. https://doi.org/10.1145/3064663.3064760

[31] Dishita G Turakhia, Andrew Wong, Yini Qi, Lotta-Gili Blumberg, Yoonji Kim, and Stefanie Mueller. 2021. Adapt2Learn: A Toolkit for Configuring the Learning Algorithm for Adaptive Physical Tools for Motor-Skill Learning. In *Designing Interactive Systems Conference 2021* (Virtual Event, USA) *(DIS '21)*. Association for Computing Machinery, New York, NY, USA, 1301–1312. https://doi.org/10.1145/3461778.3462128

[32] Nobuyuki Umetani and Ryan Schmidt. 2017. SurfCuit: Surface-Mounted Circuits on 3D Prints. *IEEE Computer Graphics and Applications* 37, 3 (2017), 52–60. https://doi.org/10.1109/MCG.2017.40

[33] Tianyi Wang, Ke Huo, Pratik Chawla, Guiming Chen, Siddharth Banerjee, and Karthik Ramani. 2018. Plain2Fun: Augmenting Ordinary Objects with Surface Painted Circuits. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI EA '18)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3170427.3188655

[34] Te-Yen Wu, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin, and Mike Y. Chen. 2017. CurrentViz: Sensing and Visualizing Electric Current Flows of Breadboarded Circuits. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 343–349. https://doi.org/10.1145/3126594.3126646

[35] Junyi Zhu, Lotta-Gili Blumberg, Yunyi Zhu, Martin Nisser, Ethan Levi Carlson, Xin Wen, Kevin Shum, Jessica Ayeley Quaye, and Stefanie Mueller. 2020. *CurveBoards: Integrating Breadboards into Physical Objects to Prototype Function in the Context of Form*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376617

[36] Junyi Zhu, Yunyi Zhu, Jiaming Cui, Leon Cheng, Jackson Snowden, Mark Chounlakone, Michael Wessely, and Stefanie Mueller. 2020. *MorphSensor: A 3D Electronic Design Tool for Reforming Sensor Modules*. Association for Computing Machinery, New York, NY, USA, 541–553. https://doi.org/10.1145/3379337.3415898