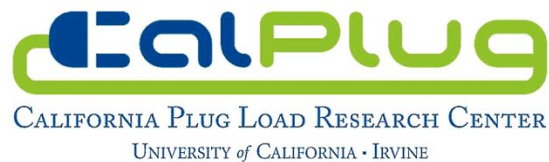
	PlugLoad Perceptoscope		Page 1
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

PlugLoad Perceptoscope



CSE Senior Design Project 2018

Team Members: Nikita Tsvetkov, Kim Phan Truong, Viet Ly

Faculty Members: Sergio Gago, Joy Pixley, Michael Klopfer, G.P. Li




4100 Calit2 Building

University of California, Irvine

Irvine, CA 92697-2800


Office: 949.824.9073

	PlugLoad Perceptoscope		Page 2
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

calplug.uci.edu | | www.calplug.org

Contents

PlugLoad Perceptoscope	1
Purpose	3
Introduction	3
Descriptions of Solution	3
Overview	3
Specific Goals	4
Results and Current Limitations	5
Approach and Technically Detailed Expansion of Overview	5
ARKit and AR Visualization on iOS Devices	5
Smart Meters	6
Object Recognition and Mapping	6
AWS Backend	7
Instructions for Use	8
Possible Extensions	22
Conclusions	22

	PlugLoad Perceptoscope		Page 3
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Purpose

The modern home has many electronic devices. Internet of Things (IOT) and smart home trends are creating new devices to be added to the consumer's home. People are now faced with a new problem of understanding their energy usage. This usage often looks like a black box system, displayed as a utility bill at the end of every month. If people understand their energy usage, then they can make decisions which save them money, and reduce environmental impact. Our solution tackles the problem by allowing people to visualize the energy usage in their home. This user-friendly visualization allows people to make informed decisions about their devices. Once a person knows how much energy is used by which device at what time, then the person can turn it off accordingly to save energy. Our solution leverages data collection by smart meters and an Augmented Reality (AR) display to visualize energy usage. This document describes what our project is, what problems it solves, how it works, the data flow, functionality, features, and examples of basic operation.


Introduction

Our solution helps people understand their energy usage. By facilitating this understanding of energy, we allow people to save money and our planet. Humans are visual creatures. By creating a system which allows people to look at their devices, and see their energy usage directly, we are creating an easy and exciting way for consumers to be more conservative about their energy consumption. Our system was also engineered with extensibility in mind. This system can be repurposed to allow any display of arbitrary sensor data. There are four crucial technological components that drive this project: an AR display, energy data collection via smart meters, object mapping and recognition, and backend services. These components combine together to create an energy audit system, which allows users to simply look at their devices directly, and receive meaningful visual information about that device's energy usage. These components will be described at a high-level in the Overview section.

Descriptions of Solution

Overview

We present the information to the user as an AR display on Apple's iOS devices, such as the iPad. This method has the advantage of being cheap. Many people already own an iPad, and other AR devices are significantly more expensive. In AR, virtual images are overlaid on top of real images captured by a device's camera. When a user looks at a device in their home, we overlay a plot of energy usage on top of that device. The plot shows the device's instantaneous demand in watts over the last 24 hours of the device's operation. We believe that AR is an intuitive way for the user to understand their energy usage, as opposed to looking at device usage metrics on their desktop computer. By leveraging Apple's ARKit technology, we allow our solution to be more extensible, and more powerful, since we accelerating development by using a free and advanced multi-featured framework.

	PlugLoad Perceptoscope		Page 4
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

The energy usage data is collected by devices called smart meters. By plugging a smart meter into a socket, and then connecting a device to a smart meter, we can observe energy usage with detail and efficiency. These plugs use wireless networks and the MQTT protocol to send their data. This data is collected and aggregated by CalPlug, and our solution visualizes this data. The smart meters all have a unique id, whose importance is explained next.

When we plug a device into a smart meter, we are creating an association between the device and the smart meter. When we look at the device, we wish to visualize the data collected by the connected smart meter. We solve this problem of association by teaching our system about these associations. We train our system about these associations with one-time use QR codes. The smart meters each have a unique id. This unique id is imprinted on the QR code, which the AR tablet recognizes. After the QR code is recognized, the AR tablet uses visual recognition to identify the device. Once the device has been identified, then an association is created between the smart meter and the device which is plugged into it. From that point, the AR device is able to visualize the energy usage data, by looking directly at the energy consuming device.

We need a place to store these associations, so therefore we use Amazon Web Services (AWS) to store the data. By using AWS, we forward the work of conserving data integrity and providing scalability to Amazon. We also use AWS to perform the actual task of generating the energy visualization plots, and interfacing with the CalPlug MongoDB database. The AWS backend is the invisible spine which connects all of our components together.


Specific Goals

As stated previously, the four main components we chose to implement our solution are an AR display, smart meters, object mapping and recognition, and AWS backend services. There are many ways to combine and utilize these components. We agreed upon the following requirements to narrow our scope, to be achievable in the context of a senior project.

Our system should be able to:

1. Use QR codes as a persistent training mechanism to create object mappings between smart meter ids and devices within a room.
2. Once trained, be able to use visual object recognition augmented by spatial tracking to identify objects.
3. After the recognition process, present a visual display of data associated with that device. In the context of CalPlug, this display is a plot of energy usage data gathered by a smart meter.

The next section will discuss our implementation of these specific goals, and the limitations of our system in supporting them.

	PlugLoad Perceptoscope		Page 5
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Results and Current Limitations

Our iPad device is able to scan QR codes which are associated to smart meter unique ids. Afterwards, it is able to create an association between this id and the recognized object. This association is stored in our AWS backend. The AR tablet can then display a plot, after visually recognizing a device without a QR code.

A MobileNet pretrained model is used to perform the device recognition. Due to time constraints, we assume that there are only 1 of each unique object in the room, e.g. there is only one home-theatre system in a room. The pool of recognized objects is also limited to the 1000+ objects that MobileNet knows about.

The AR tablet uses the Simultaneous Localization and Mapping (SLAM) features of ARKit to try to place the plot and device recognition point onto the device properly in space. The SAML tracking compensates for the limitations of using a classification based image recognition model (MobileNet). This classification model is fast (~30 frames per second) but it lacks the ability to create a bounding box around the detected object. We tried using a feature based recognition model called TinyYOLO. This model can place bounding boxes, but it performs at 15 frames per second, and only identifies about 20 different objects. It takes an Nvidia Titan graphics card to run this model at over 30 frames per second. The SLAM tracking of ARKit is also what allows the plots to stay around the object in space, and behave as if they are part of the environment. The users can also use gestures to move the plots around in space, in order to make the display clearer. The plots can also be remapped without a QR code, once they have already been mapped.

We also use our AWS backend to collect data from CalPlug's MongoDB aggregate collection of smart meter data. We store this data, so that we do not need to rely on a consistent connection between our system, and CalPlug's DB. We can use AWS ElasticSearch to quickly parse the data, and transform it into a visual plot using matplotlib and AWS Lambda. This transformation is done on the backend, and the final plot is transferred through the AWS API Gateway to the tablet.

Approach and Technically Detailed Expansion of Overview

The following sections will detail the four major components mentioned previously.

ARKit and AR Visualization on iOS Devices

We use the Apple ARKit for our visual display. ARKit is only available on iOS device with an M9 processor or better. We use a Xamarin version of the ARKit framework for iOS devices. Xamarin is used, so that development can be done in C# on a Windows computer. An Apple computer is still necessary to compile the code. However, with this setup, we can have multiple developers writing C# code on Windows Machines, but all compiling on 1 Apple computer over SSH.

There are four main components of the AR visualization code: QR recognition, image recognition, association storage, and the visual display. QR image recognition is done via the Apple Vision Framework by requesting analysis on each image frame captured during the AR session.


	PlugLoad Perceptoscope		Page 6
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	


Image recognition is done with the Apple CoreML framework and a pretrained model. Each image frame captured during an AR session has a CoreML analysis performed on it. Association storage is done over an HTTPS connection to the AWS API Gateway endpoint. This endpoint is used to store and retrieve associations between QR codes, and MobileNet objects. The associations are stored serialized as HTTPS JSON request bodies. The visual display is a conjunction of ARKit and SceneKit objects. These objects exist in a scene graph, and are manipulated spatially using ARKit feature point detection. The plots are png images fetched over HTTPS from the AWS API Gateway, and are generated on the backend side via an AWS Lambda service, which uses other services to assist it.

Smart Meters

Smart meters is how we are able to collect energy usage data for visualization. The purpose of this document is not to explain how these meters work, but how we interface with them. CalPlug has a system, where these meters communicate over MQTT to push data to their MongoDB server. The CalPlug system aggregates two different types of smart meter networks into one database. We are able to save development time, by polling their database solution, rather than using MQTT directly. We poll this database with a python script. This python script is able to query the latest sensor data that we are missing, based on the timestamp field. The python script resides in an EC2 Virtual Machine. After polling the new data, the script then dumps it over HTTPS via AWS API Gateway. This method servers as an abstraction layer, which allows to place arbitrary sensor data into our AWS backend, via AWS API Gateway endpoints.

Object Recognition and Mapping

We use a pretrained image classification model called MobileNet. This model has very fast performance, while maintaining a reasonable accuracy. One way to judge the prospective performance of a CoreML pre-trained model is the size of the model, and this model is one of the smallest. We hypothesize that given enough time, it would be possible to train a special neural network, which is more suitable for our use cases. However, the training of that network would be a senior project of its own. As mentioned earlier, we tried using a different kind of model, which is a feature based recognition model called TinyYOLO. It uses a VNCoreMLFeatureValueObservation class rather than a VNClassificationObservation class. The feature observation class is capable of generating bounding boxes around the object. However, our performance was much worse with this model, and the number of objects we could detect were limited.

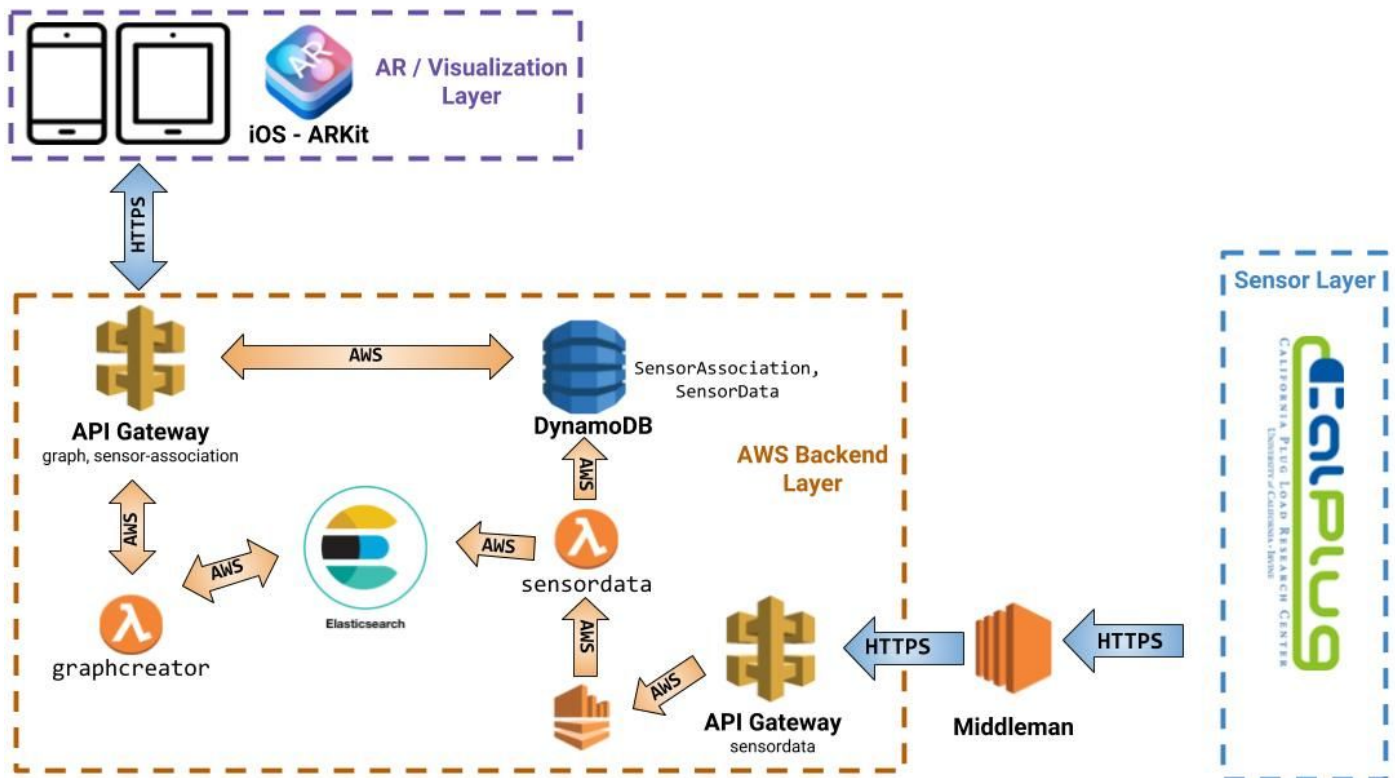
	PlugLoad Perceptoscope		Page 7
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	


AWS Backend

Our backend was engineered with extensibility and scalability in mind. The following AWS structure diagram is serialized as an AWS Cloud Formation template, so that easy deployment of infrastructure can be made by anyone with an AWS account. In the diagram below, the purple colored layer is the AR tablet which communicates over HTTPS with the AWS layer, in order to fetch the plot images, and store and retrieve the visually recognized object association mappings. The sensor layer, which is represented by the blue colored CalPlug layer, is the data source that we retrieve sensor data from.

We use API Gateway to create a REST API for our backend. The API Gateway for the AR Layer has three endpoint HTTPS methods. There is a GET graph, a POST sensor-association, and a GET sensor-associations. The graph is sent as an HTTPS body png image, and the sensor associations are JSON HTTPS bodies. The API Gateway for the sensor layer takes in a JSON HTTPS POST body. Once we data passes these gateways, it is in our innermost service layer.

The innermost layer utilizes several AWS services: DynamoDB, Lambda, and Elasticsearch. DynamoDB and Elasticsearch are used to store and query data; DynamoDB serves as a data warehouse, while Elasticsearch is used to query data quickly and easily without a schema. Finally, Lambda is used to transform data between services or integrate with the API Gateway.



	PlugLoad Perceptoscope		Page 8
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Instructions for Use


First, one has to follow both the front_end and back_end guides available on GitHub in order to setup the system.

GitHub: <https://github.com/CalPlug/EnergyDataVisualizer>

front_end: https://github.com/CalPlug/EnergyDataVisualizer/tree/master/front_end

back_end: https://github.com/CalPlug/EnergyDataVisualizer/tree/master/back_end


Once the system is ready, the following steps can be followed to demonstrate a complete usage of all the system's features.

	PlugLoad Perceptoscope		Page 9
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 1 - QR Code Preparation

Prepare the QR codes which simply encode the unique id of a smart meter.




	PlugLoad Perceptoscope		Page 10
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 2 - QR Code Analysis

Bring the tablet closer to the QR code, with the code in the center of the screen.




	PlugLoad Perceptoscope		Page 11
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 3 - Plot Loading

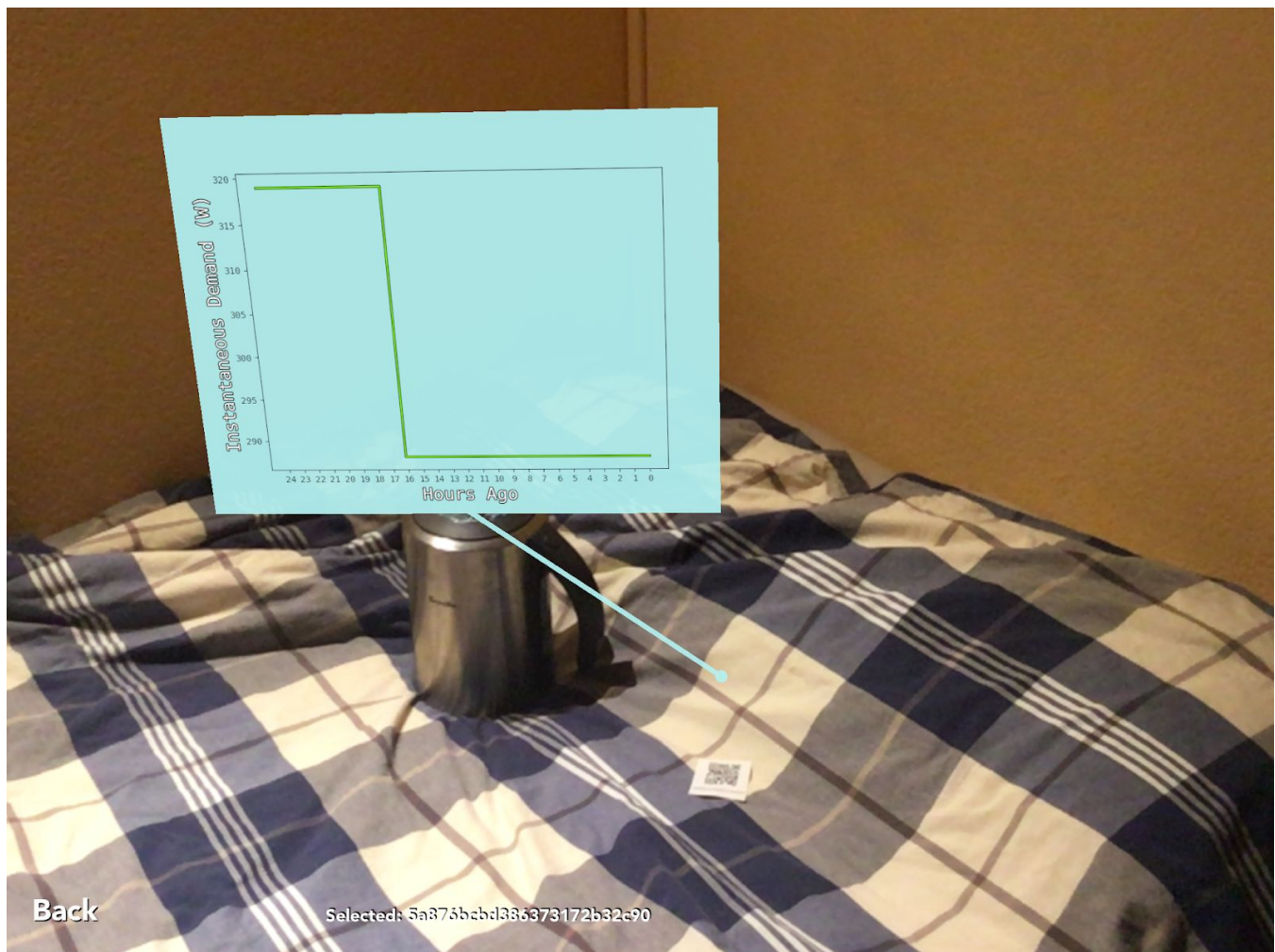
When the QR code is recognized, a loading indicator will popup at the bottom left of the screen.




	PlugLoad Perceptoscope		Page 12
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 4 - Plot Display

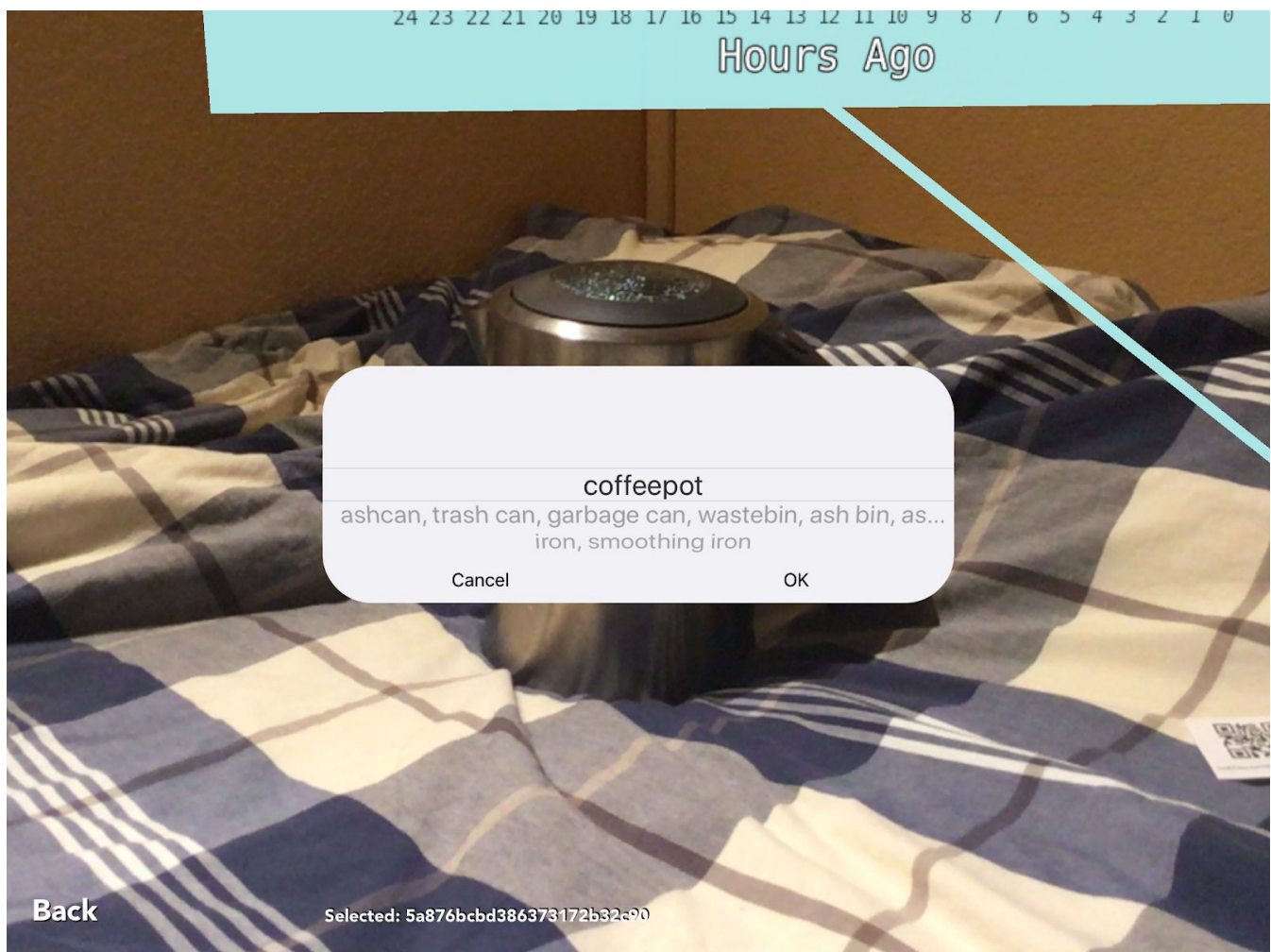
The energy visualization plot for that smart meter has now been loaded, and is ready to be mapped. Notice that the id of the smart meter shows up on the bottom of the screen. There is currently no object association for it because there is no text displayed next to the unique id. The object recognition point and line should be pointing at the QR code.




	PlugLoad Perceptoscope		Page 13
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 5 - Object Mapping

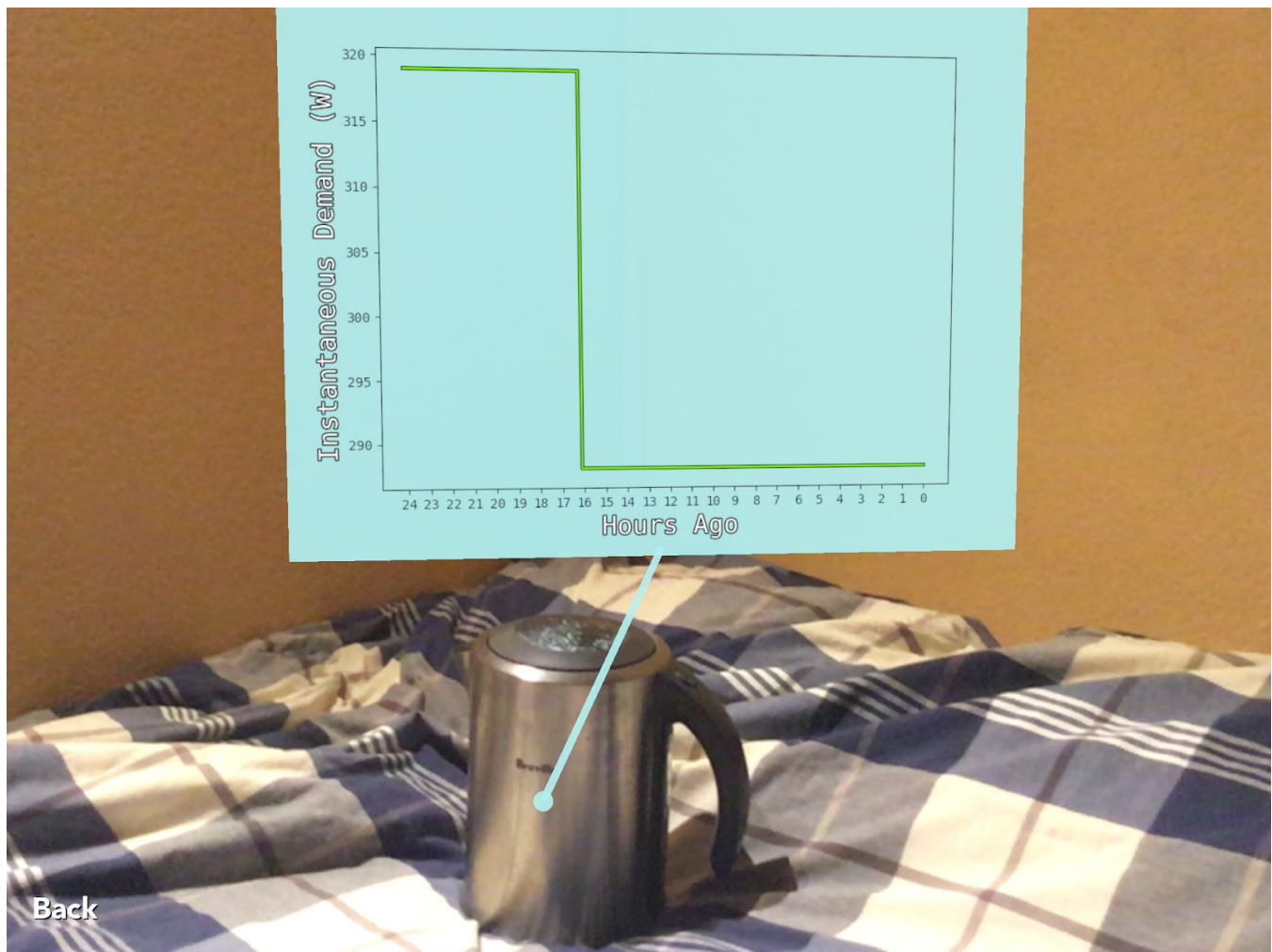
You can now create an object mapping to a visually recognizable device. Perform the mapping by looking at the desired device, and then tap on your screen with one finger. A menu with a dial-like selection dialogue will popup. This menu lists the top image recognition guesses made. To get a list of new guesses, tap the screen again without selecting anything. The Cancel button can be used to go back without mapping anything. Hold your finger on the selection dial to select the desired device. You can also press OK in order to choose the device that is currently selected.




	PlugLoad Perceptoscope		Page 14
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 6 - Mapping Confirmation

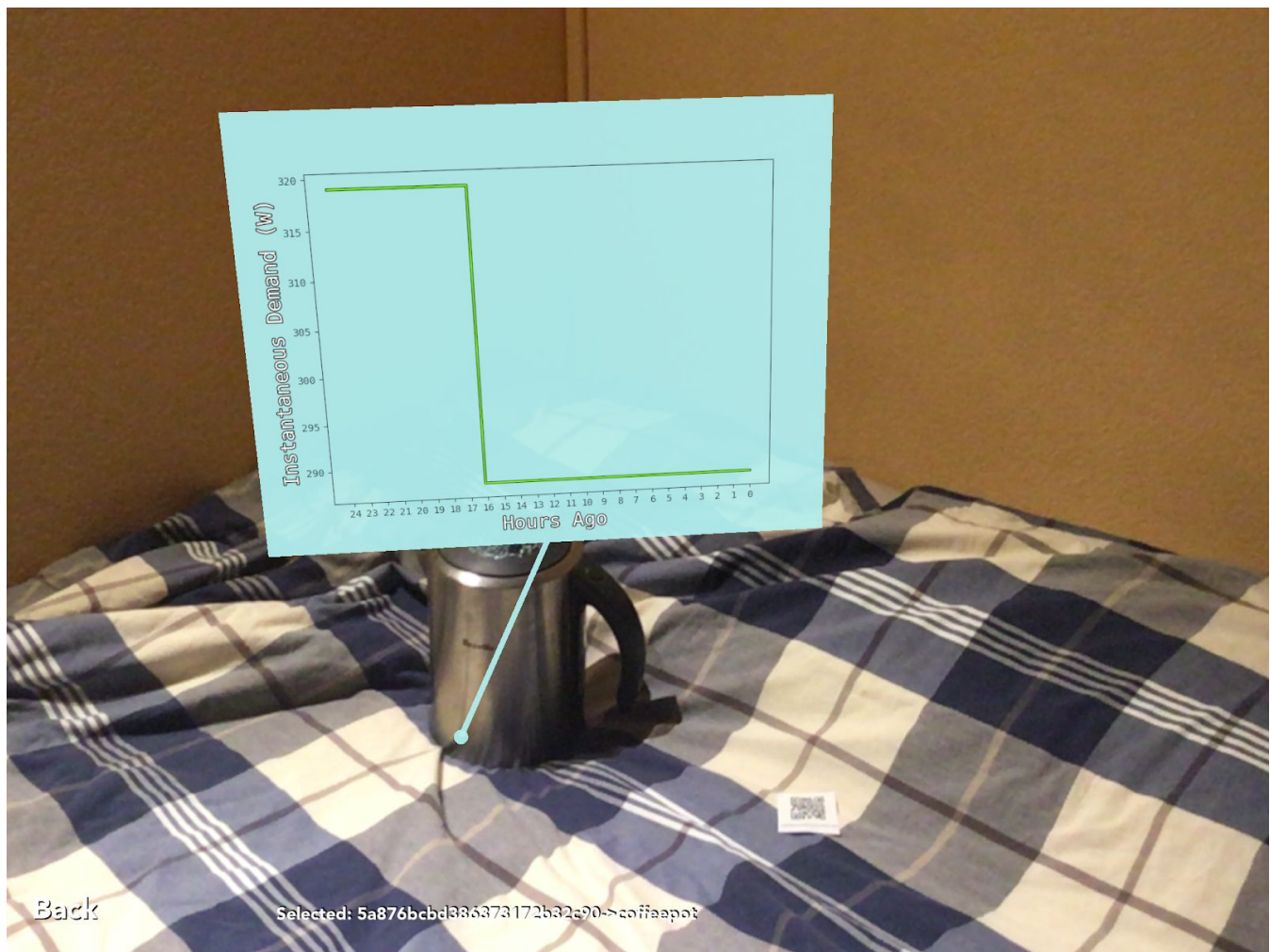
As soon as you choose the device name, the mapping has been performed, and persisted in our backend database. Notice that there is no longer any smart meter id selection text on the screen. The object recognition point and line should now point to the newly mapped device, instead of the QR code.




	PlugLoad Perceptoscope		Page 15
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 7 - Mapping Confirmation and Reselection

You can now do a single finger tap on the displayed plot in order to select it. Selecting the plot allows you to confirm that it has been mapped properly. Notice that at the bottom left of the screen, the text now reflects that the id has been mapped to an object. Because the id is now in the selected state, a remapping can be performed.




	PlugLoad Perceptoscope		Page 16
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 8 - Optional Remapping

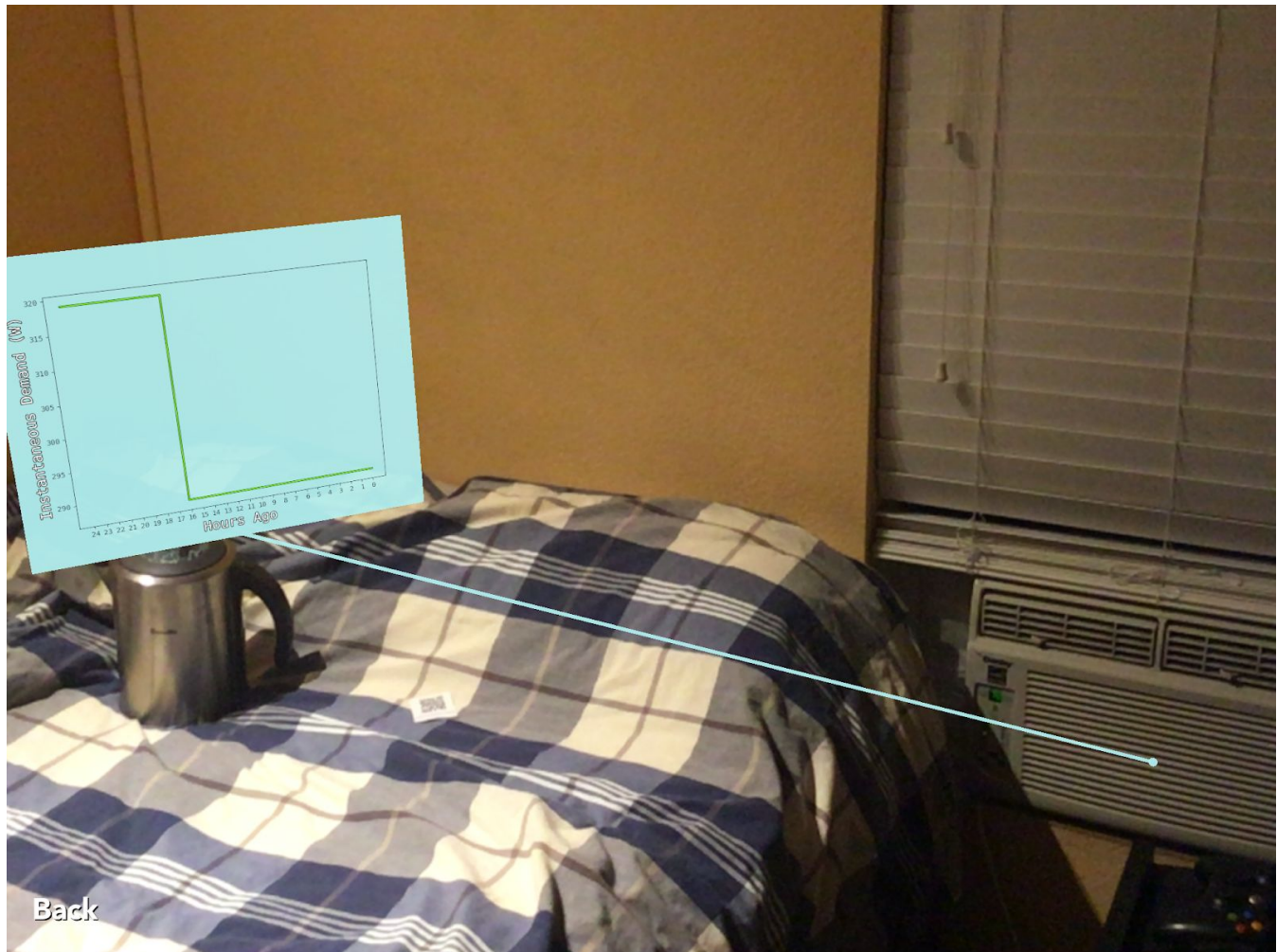
To remap the object, all you have to do is look at another device and tap to get the selection dial, just like in Step 5.




	PlugLoad Perceptoscope		Page 17
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 9 - Remapping Confirmation

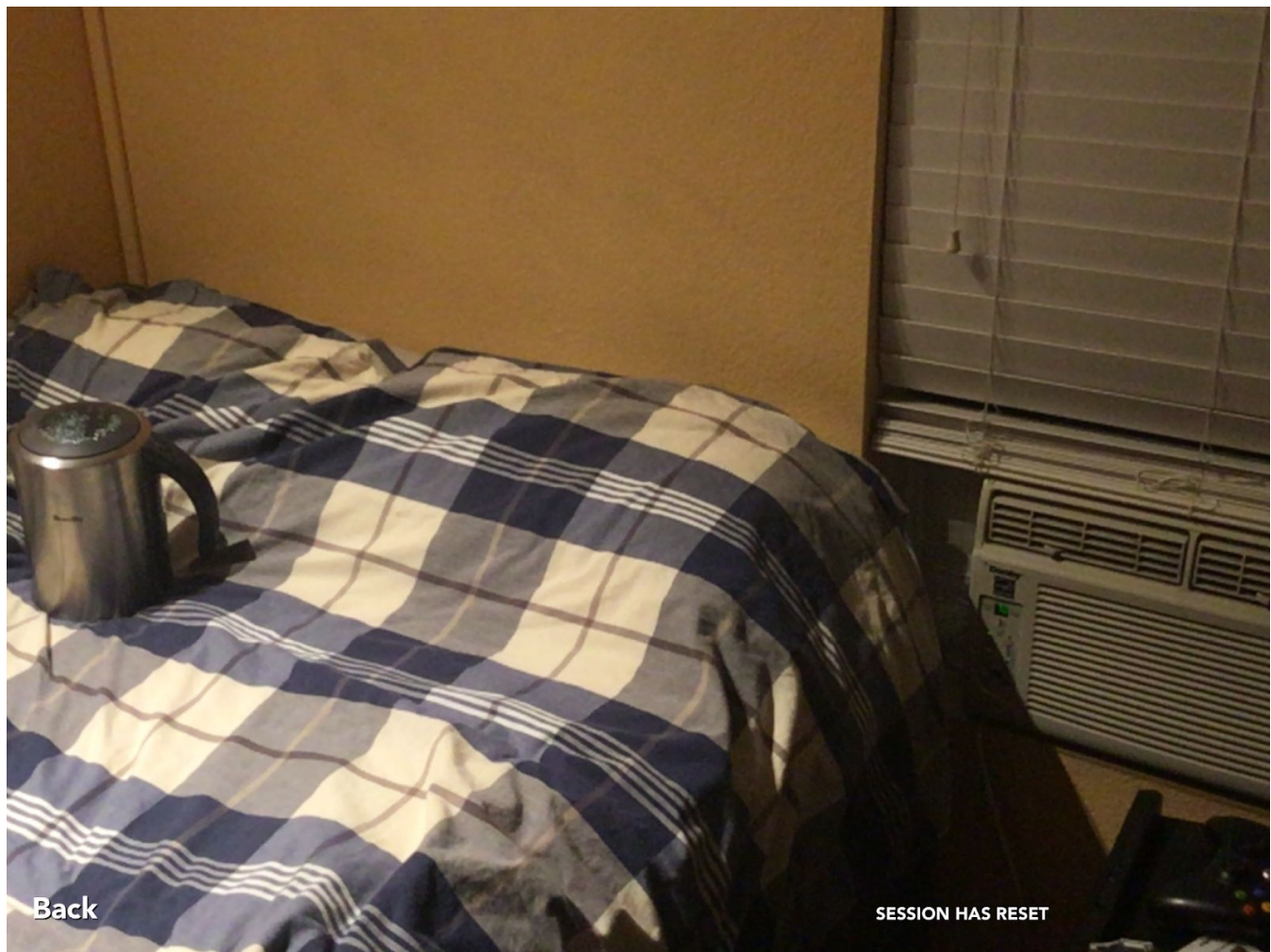
Once a device name has been chosen, the mapping has been performed just like in Step 6. The object recognition point and line should reflect the re-mapping.




	PlugLoad Perceptoscope		Page 18
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 10 - Object Recognition 1

You can now press the back button to start a new session or kill the application or restart your tablet. When you relaunch the application no plots should be visible, but our previous mappings should have persisted.




	PlugLoad Perceptoscope		Page 19
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 11 - Object Recognition 2

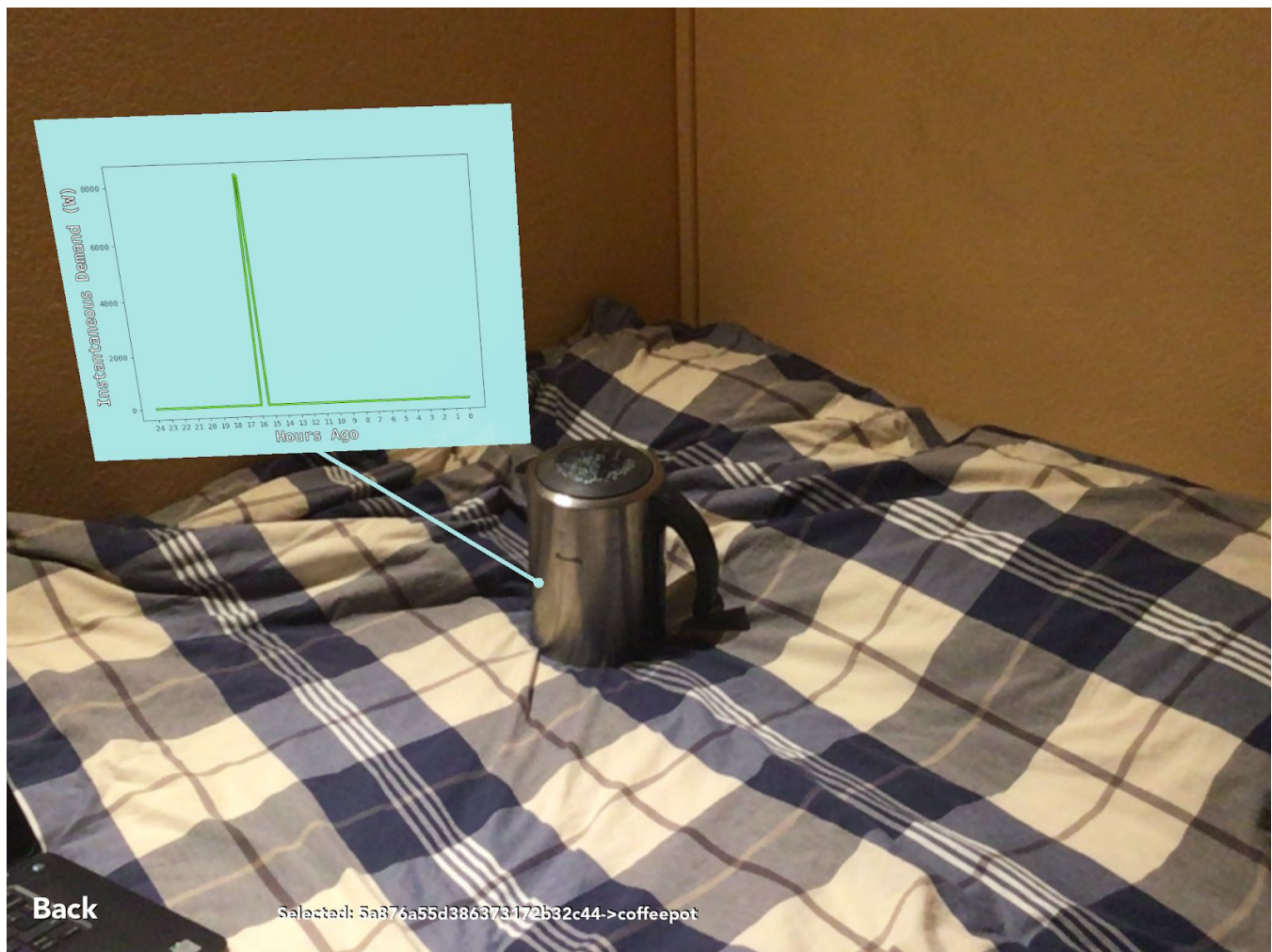
In order to visualize energy usage by a device, simply bring your tablet closer to the device, with the device in the center of the screen. You should get a loading indicator at the bottom left of the screen once the device has been recognized.




	PlugLoad Perceptoscope		Page 20
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 11 - Energy Usage Visualization

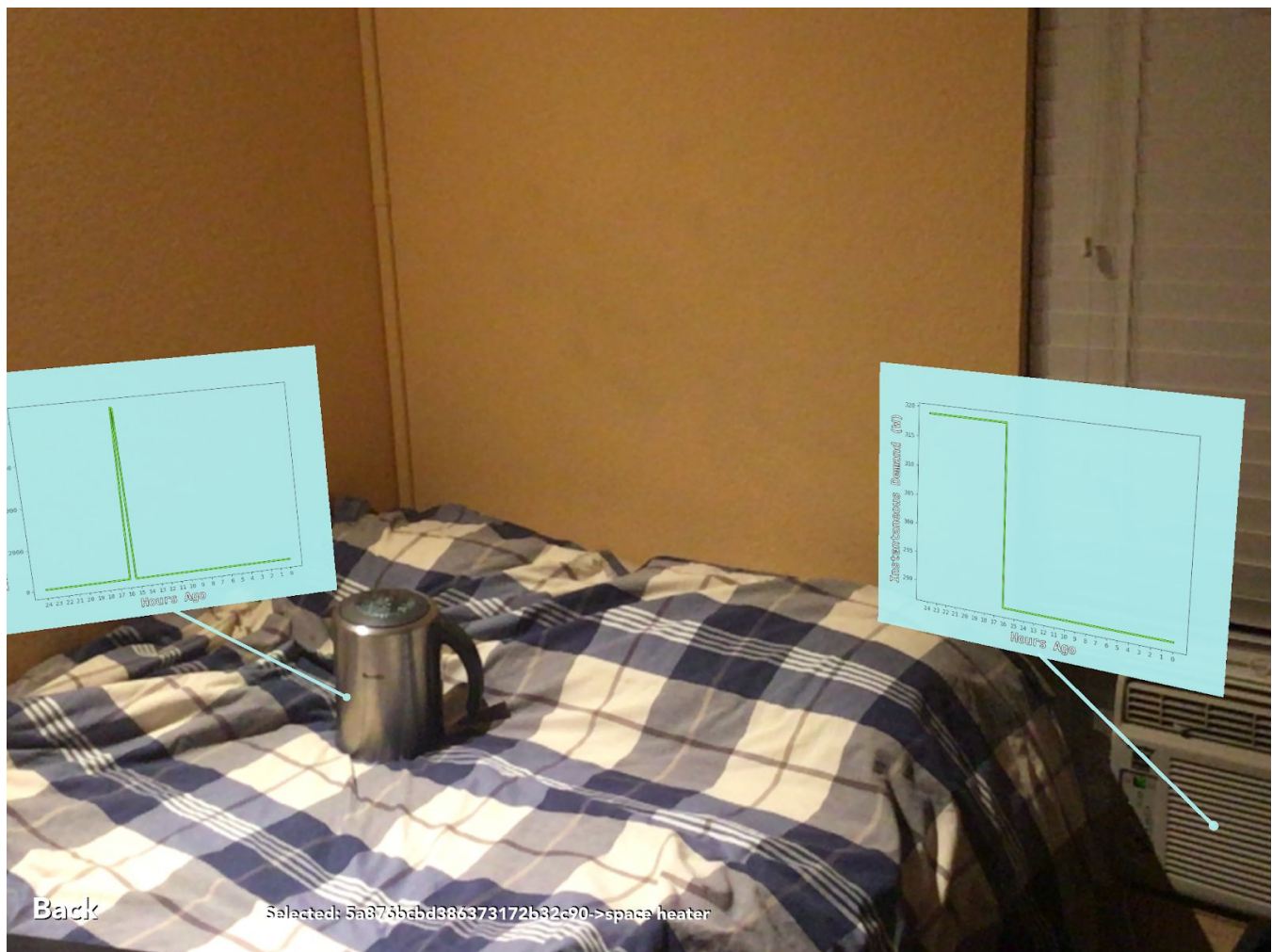
The AR will now display an energy visualization plot for the associated device.




	PlugLoad Perceptoscope		Page 21
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Step 12 - Add More Devices

The steps can be repeated to add more devices for energy usage visualization.



	PlugLoad Perceptoscope		Page 22
	Technical Authors: Viet Ly, Nikita Tsvetkov, Kim Phan Truong	03/28/18 Rev 1.0	

Possible Extensions

This project is minimalistic, and therefore there are a lot of directions that it can grow in. The following points are just some of the possible areas of extension that are possible.

- A different image recognition model can be used or trained in order to fit more custom use-cases. Our current model works for a consumer home, but would fail in an industrial setting, where a lot of objects look similar or bland with a lack of distinguishing features.
- A different recognition model can be used, or an algorithm can be added to our current model, to support bounding box visual indication of the selected object. As discussed previously, feature recognition models which inherently provide bounding boxes have worse performance. However, in the future a new model might come out that has better performance, and still accomplishes this task. It is also possible to use the current MobileNet classification model, and run the analysis multiple times with different crops of the current captured image, so that a pseudo-feature based analysis is performed. This method will once again be limited by performance constraints, and the amount of image crops per second that can be analyzed may be low.
- The ARKit is still a new framework, and there are features that we were not able to use, simply because they were not out yet. The release of iOS 11.3 on 3/28/18 added new ARKit features which can be leveraged to make our solution better.
- Arbitrary data can be visualized by the system. Our current architecture separates the data gathering script from the internal parts of the AWS backend system. Our structural diagram shows that the data readings come in through the AWS API Gateway, even though the data gathering script exists on infrastructure that we setup. This is because we can have any sensor data being sent to our API Gateway, and only once it passes through there, then the data exists in our innermost layer.
- Plot gestures and controls are one of the easier extensions that one could make. It is always possible to improve the usability of our data visualization system, by improving how the user interacts the with data displays. As suggested by Joy Pixley and Michael Klopfer, a pinch gesture can be added to make the plots bigger or smaller. Also, a double-tap on plot gesture can be added, to make the plot occupy the full space of the screen, similar to zooming in.

Conclusions

Our solution allows users to take control of energy usage in their homes, through AR visualization. Our system is extensible enough to be generalized for any sort of visual data display, such as arbitrary sensors in an industrial environment