Journal of Sensor and Actuator Networks

MDPI

Article

VisKit: Web-Based Interactive IoT Management with Deep Visual Object Detection

Chae-Eun Hwang, Sung-Hun Lee and Jin-Woo Jeong *

Department of Computer Engineering, Kumoh National Institute of Technology, Gumi 39177, Korea; codms50111@naver.com (C.-E.H.); hun3102@naver.com (S.-H.L.)

* Correspondence: jinw.jeong@kumoh.ac.kr

Received: 30 December 2018; Accepted: 8 February 2019; Published: 12 February 2019

Abstract: Various technologies and standards for the Internet of Things (IoT) have defined the way devices should interact with each other to provide an intelligent IoT service to users in an efficient manner. Although the usability of system interface between the platform and users is one of the key factors for the success of IoT ecosystems, the manner in which IoT platforms should interact with users has not been well studied. Current IoT platforms provide a simple list-based interface to manage devices, which result in the degradation of their usability as the number of devices increases. In this paper, we propose an interactive web-based IoT management system where deep object detection techniques are adopted to facilitate user's device selection and management. The proposed system automatically recognizes the device type from video streams and accordingly generates smart controllers. The users can choose a device by touching an object in the video stream and use a smart controller to control the selected device. Finally, we show the feasibility of the proposed system through the implementation of a prototype which demonstrates a set of user scenarios.

Keywords: device control; Internet of Things; object detection; web interaction

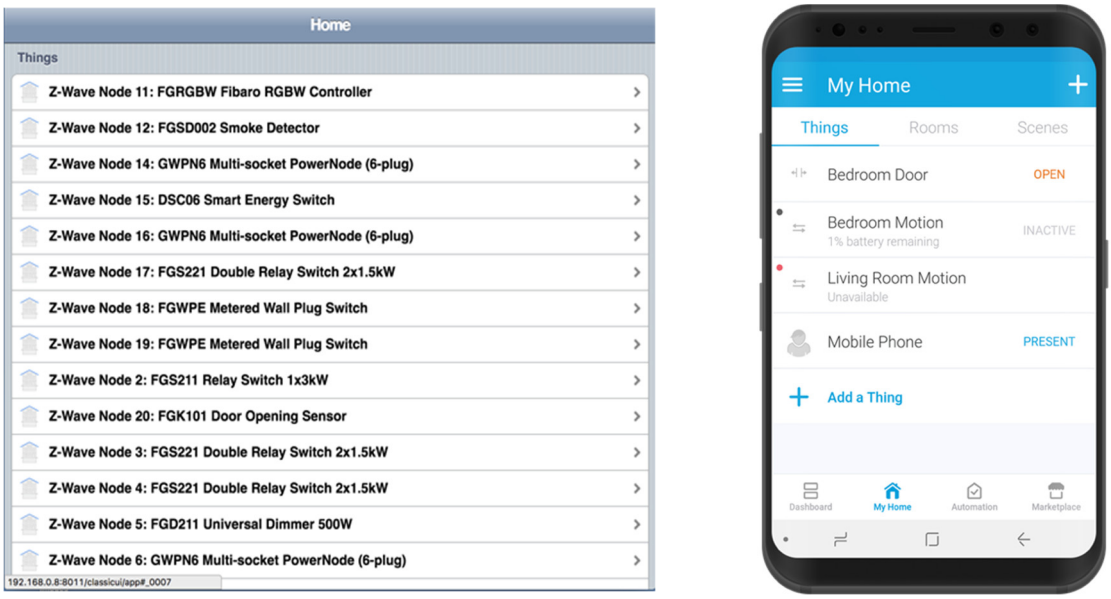
1. Introduction

Recently, with the development of machine to machine communication technologies such as Ethernet, Wi-Fi, Bluetooth, and ZigBee, people can now experience various user scenarios, which require a seamless connection between devices [1]. These technologies have enabled different kinds of devices to interact with each other, thereby supporting remote monitoring and controlling of smart objects or devices [1,2]. In addition, the recent growth of sensors and actuators with a mature network infrastructure has resulted in the formation and foundation of the Internet of Things (IoT). In the era of IoT, devices can exchange data, analyze their environment, make decisions, and perform various operations without user's explicit instructions.

To realize the ultimate goal of IoT, where different types of devices can communicate with each other to generate and provide intelligent services, standardization for interoperability between the devices is considered the main success factors in the IoT ecosystem. IoT standards [3,4] define how IoT devices should interact with each other to discover and manage devices, collect and share information, and perform physical actuations.

The usability of a system—how easily users can interact with a system—is also one of the important factors to expand an IoT ecosystem. However, methods to improve the usability of IoT platforms/systems have not been intensively studied. Recent commercial IoT platforms/systems such as IBM Watson IoT Platform [5], Thing+ [6], Amazon Web Services (AWS) IoT [7], OpenHAB [8], and Samsung SmartThings [9] and ARTIK [10] platform provide an interface to manage, monitor, and control a set of buildings, locations, devices through their mobile or web-based applications. Figure 1a depicts a web-based interface to show a list of installed devices supported by OpenHAB [8]. The list in Figure 1 shows that there are approximately 20 devices installed in "Home", some of which are

sensors and the rest are actuators. However, it is not intuitive for users to recognize the name, type, and installed location of each IoT device owing to the inefficient design of its web application interface. Figure 1b is a mobile interface provided by SmartThings [9], which provides a simple list-based interface to show a set of installed devices. Even with a mobile application, it is obvious that the usability of an IoT platform will drastically decrease as the number of installed devices increases because users cannot easily choose the IoT device they want to observe and manipulate.



(**a**) (**b**)

Figure 1. Conventional interfaces of Internet of Things (IoT) platforms: (**a**) web interface provided by OpenHAB [8]; (**b**) mobile interface provided by SmartThings [9].

Furthermore, the recent advances in IoT ecosystem enables the interoperability between users, processes, data, and devices, which results in the increasing needs of more complex IoT scenarios such as "Turn off all lights when nobody is at home" or "If a room is cold, turn on the air-conditioner to warm up". To handle such a complex query, several commercial IoT platforms and previous studies have tried to develop an efficient rule-engine or data structure for IoT environment [11–14] and to provide rule-based application services through their web or mobile interfaces [7,9]. However, the previous works focused on introducing the functionality but failed to consider the usability when designing an interface for composing IoT rules between smart objects, such as sensors, actuators, and applications. With a list or grid-type interfaces, it is still difficult to find and select an appropriate object to be used for the condition and action components of IoT rules if the number of installed devices is large. Figure 2 shows a web interface for the rule composition of ARTIK Cloud which supports a simple list-based view. In this design, users need to scroll up and down to locate a target device to use, which is inconvenient and time-consuming.

To address the limitations above, we introduce an interactive IoT management system with deep object detection techniques. The proposed system is equipped with a set of smart cameras to monitor the in-door status of a room in real-time. With the proposed system, the users not only access video streams to observe the in-door status but also directly control the smart devices found in the video streams. Specifically, the proposed system automatically analyzes the video streams to detect and recognize the category of a smart device. If users touch a detected object in the video stream, a context menu (i.e., smart controller) for the object opens to provide various IoT-related services such as monitoring and control, and composing rules. Through the proposed system, the users can more intuitively and quickly locate the target IoT device to use.

The rest of this paper is organized as follows. Section 2 briefly reviews the previous studies. Section 3 provides the details of the proposed interactive IoT management system with deep object

detection methods. In Section 4, prototype implementation and the results on user studies are discussed. Finally, we provide the conclusion and future research directions in Section 5.

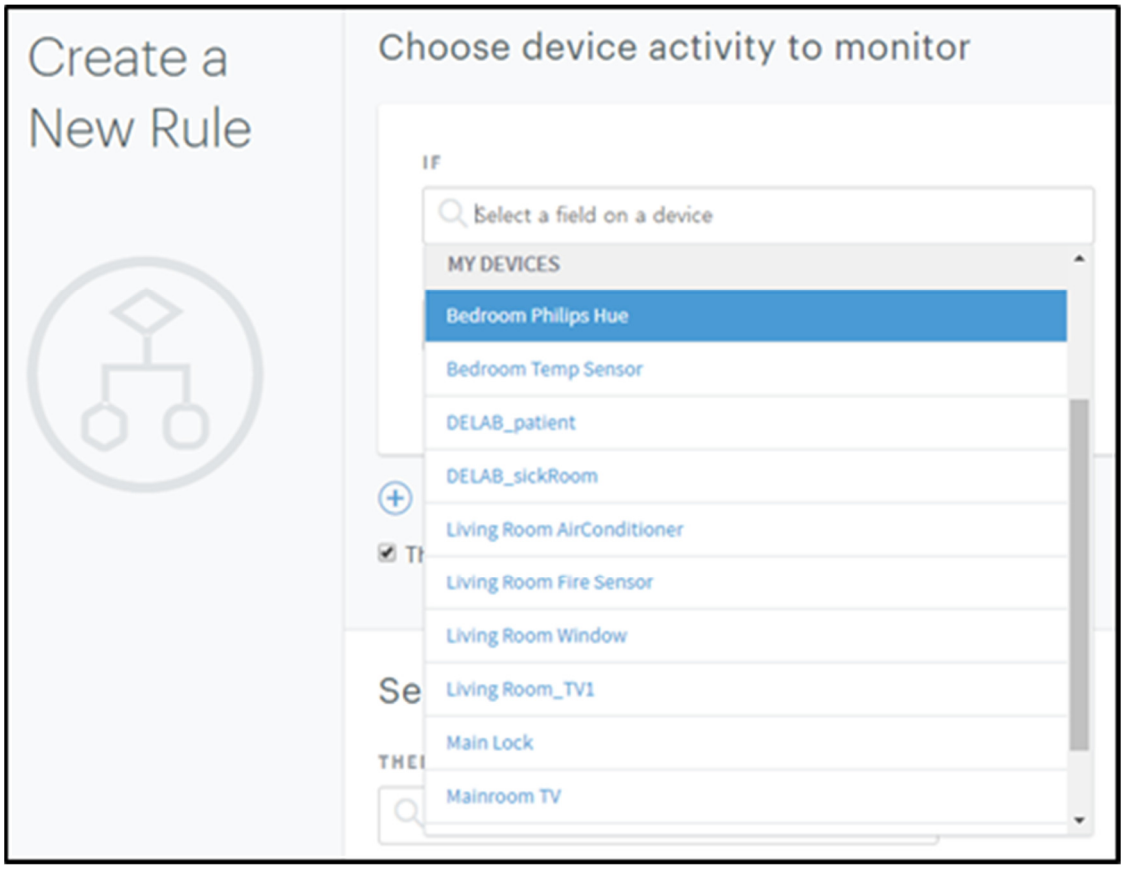


Figure 2. Web interface for rule composition provided by ARTIK Cloud [10].

2. Related Work

There have been various products and studies to provide IoT applications and services with cloud support. In this section, we briefly review the previous works and compare them with the proposed work from various perspectives.

Recently, global companies such as Amazon, IBM, and Samsung have converged on the strategy of offering IoT services, platforms, and hardware modules [15]. Amazon IoT (AWS IoT) [7] provides interaction between the devices and AWS services through a cloud-based architecture. The connected devices can communicate with each other by traditional data protocols such as Message Queue Telemetry Transport (MQTT), HTTP, or WebSockets. In addition, the AWS IoT provides an interface for a rule engine to support building applications that act on data generated by IoT devices. IBM Watson IoT [5] also provides a cloud-based IoT management system with various kinds of analytics services (e.g., data visualization in real-time). The messaging protocols like MQTT and HTTP are supported for communication between users, IoT devices, applications, and data storages. The IBM Watson IoT also provides a rule composition system to set conditions and actions for IoT devices or applications. Samsung ARTIK platform [10] provides ARTIK IoT hardware modules as well as a cloud service which provides interoperability between devices, applications, and services. For easy development of IoT applications and services, ARTIK supports various connectivity protocols (e.g., Ethernet, Wi-Fi, ZigBee, Thread, etc.) and data protocols such as REST/HTTP, MQTT, and WebSockets. Similar to AWS IoT and IBM Watson IoT, Samsung ARTIK Cloud provides an IoT rule engine interface to add and edit the rules or to invoke actions when conditions are met. In particular, the ARTIK cloud provides a device simulator feature so that the development and testing of new IoT services can be easy and straightforward.

The recent IoT platforms including ARTIK, AWS IoT, and IBM Watson IoT tried to provide a GUI-based web application for device and rule management as presented in Figure 3. As stated in Introduction, however, the current web-based GUI applications of commercial IoT platforms did not consider the usability; hence, it is not intuitive and efficient for users to manage the IoT devices and rules. For example, ARTIK cloud web application's device management interface shown in Figure 3a just provides a long list of installed devices, each of which displays a name, type, and some buttons. If the number of installed devices becomes large, the users are required to spend much more time to locate their target devices. Figure 3b also presents a web interface for rule management of the ARTIK cloud. Similar to the device management interface, the rule management interface also provides a list of generated rules, each of which displays a name, descriptions of rule components based on the ARTIK cloud rule syntax. Therefore, it is obvious that the users have to spend much more time as the number of devices increases and the type varies.

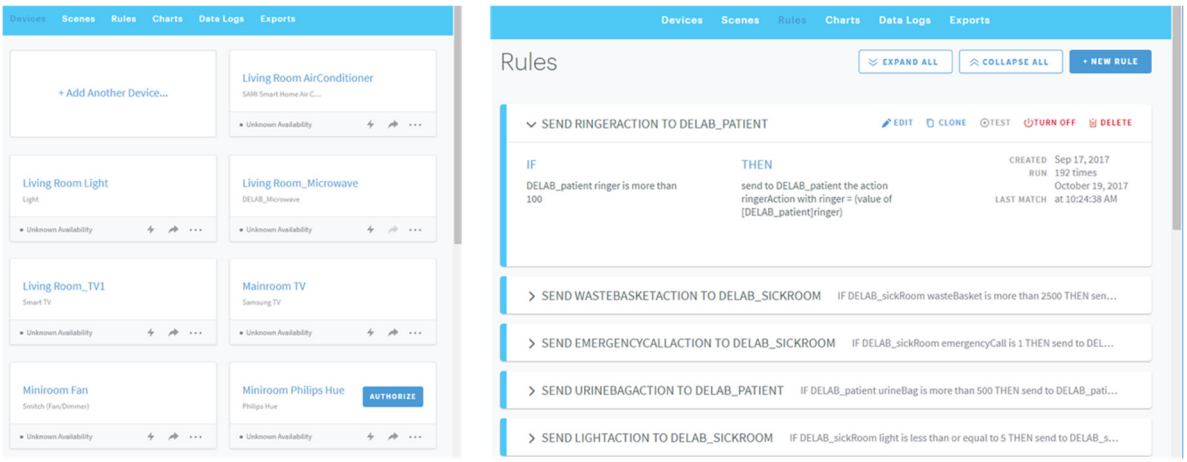


Figure 3. A web-based interface of ARTIK Cloud: (**a**) device management, (**b**) rule management.

Recent academic studies proposed various models, methods, and architectures to define and integrate IoT components for the success of IoT ecosystems. Garcia et al. [16] defined a theoretical frame about smart objects with a different dimension of intelligence. Each IoT device was classified based on its level of intelligence, location of intelligence, and integration level of intelligence. The authors of [16] claimed that a smart object with the intelligence through the network from the location of the intelligence dimension is the most important device type since these kinds of objects can have ever-increasing intelligence through the network; therefore, be more intelligent. Molano et al. [17] analyzed and reviewed the relationships between IoT, social networking services, cloud computing, and Industry 4.0. The authors of [17] presented a 5-layer architecture for an IoT platform which is comprised of sensing, database, network, data response, and the user layer. A meta-model of system integration which defines the responsibility and relationship between sensors, actuators, rules, and social networks was also introduced. The proposed system in this paper consists of the core components mentioned by [17] (i.e., sensors, actuators, rules, etc.) and gives an appropriate role to each component according to the meta-model presented by [17]. Ref. [18,19] presented an efficient and interactive device and service management model for cloud-based IoT systems. Dinh et al. proposed an efficient interactive model for the sensor-cloud to efficiently provide on-demand sensing services for multiple applications with different kinds of requirements [18]. The sensor cloud model was designed to optimize the resource consumption of physical sensors as well as the bandwidth consumption of sensing traffic. Cirani et al. [19] introduced the virtual IoT Hub replica, a cloud-based entity replicating all the features of an actual IoT Hub, which external clients can query to access resources. Their experimental results demonstrated the scalability and feasibility of the architecture for managing resources in IoT scenarios.

The aforementioned studies [16–19], however, still focused on the efficient management and interaction of smart objects (i.e., sensors and actuators) rather than interaction between users and IoT services. The platforms and systems presented by [20–24] suggest various approaches to provide IoT services by taking the interaction between users and devices into account. Ref. [20,21] proposed an IoT-based smart home automation and monitoring system. Various sensor and actuator nodes are configured with microcomputers like Raspberry Pi and Arduino and then installed in a home. The collected data are transmitted to a cloud server and subsequently processed according to the user's requests. However, these works also suffer from the limitations that a simple list-style interface can have. Memedi et al. [22] presented an IoT-based system for better management of Parkinson disease by giving patients insights into symptom and medication information through mobile interfaces. In particular, the mobile interface was designed according to the usability heuristics for the elderly, such as text size, color and contrast, navigation and location, mouse use, and page organization. Similarly, Wu et al. [23] and Hong et al. [24] proposed a GUI-based application to monitor the status of sensor nodes in a safety and health domain. However, ref. [23] does not support a manual or rule-based actuation features; therefore, its usefulness is limited. In particular, since [24] depends on the ARTIK cloud as its cloud backend, the aforementioned problems and limitations of ARTIK cloud interface still exist. On the other hand, Zhao et al. [25] and Heikkinen et al. Ref. [26] proposed methodologies and architectural frameworks to build and support time-critical applications in a cloud environment.

Even though various studies have been proposed for the integration of IoT services, applications, and devices, how to improve the usability of a system interface focusing on the interaction between users and systems still remains challenging. The remainder of this paper describes how the proposed work addresses this issue by (1) integration between the IoT platform server and deep learning-based object detection approaches and (2) a web-based interface with smart controllers.

3. The Proposed System

3.1. Main Scenario

To show the main contribution of the proposed system and its differences compared to the previous approaches, we briefly describe the main use case of the proposed system.

Assume we have a set of rooms and installed devices connected to an arbitrary IoT platform. Figure 4 presents a web-based IoT management interface which is similar to current commercial IoT platforms. As mentioned above, the use of a simple grid (or list)-based interface results in the degradation of the usability of a system as the number of installed devices increases. Figure 5 illustrates a typical example to control a device installed in a certain room. In this example, the user has to find a target room (e.g., Mini Room) first and subsequently locate a target device (e.g., TV) from the list. Therefore, users need to spend more time if the list is long even though the list items are represented as a figure or icon to improve readability.

Figure 6 shows how the proposed system works. To observe the indoor status of a room, users can interact with a visual interface in which video streams captured from a smart camera are displayed. In the visual interface, users can immediately locate an IoT device recognized by the system and subsequently use a smart controller (a white rectangle with buttons shown in Figure 6) to check the details of the device or to perform operations that are supported by the device (e.g., turn on/off, volume up/down, etc.). The smart controller for each device appears when a user clicks or touches the identified device on the video stream view, which means a user does not need to scroll up and down to locate a target device. In addition, users can add a new IoT rule for the device or view a list of IoT rules regarding the selected device by touching the rule-related buttons attached at the bottom of a smart controller. Therefore, the goal of the proposed system design is to improve the usability and efficiency of the existing IoT platform interfaces.

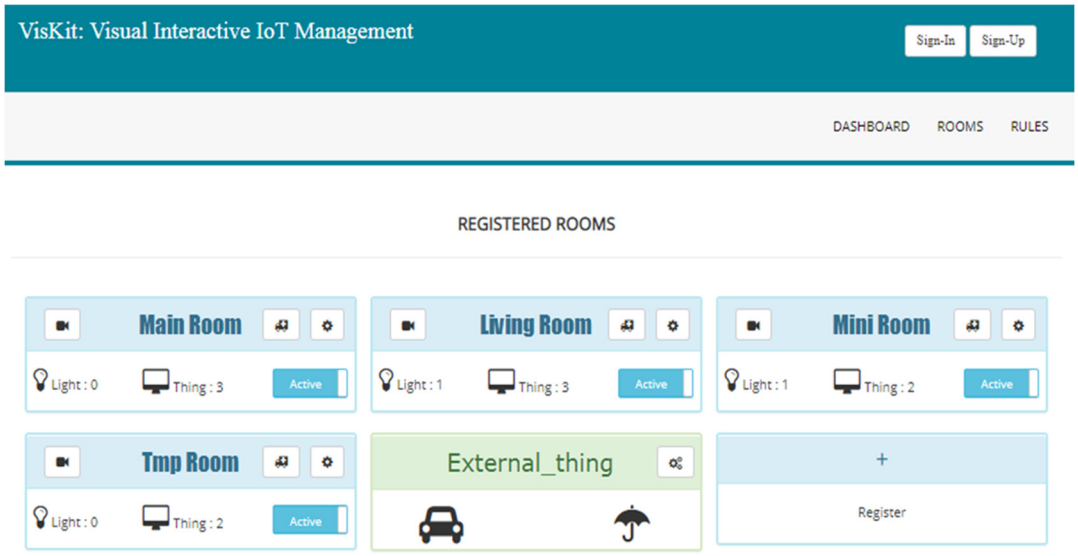


Figure 4. A general web-based interface for IoT management.

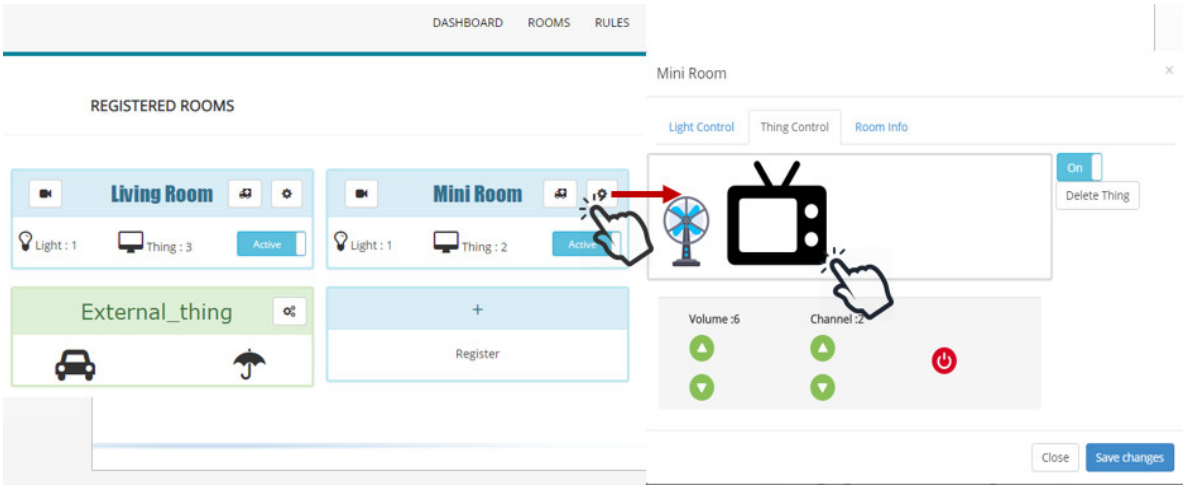


Figure 5. A typical way of using a web-based interface to control IoT device.

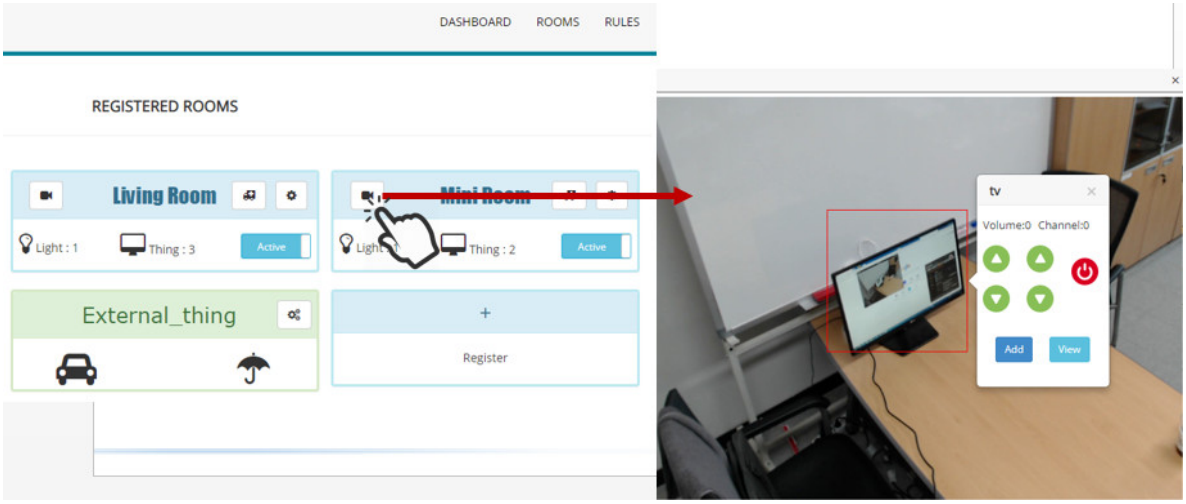


Figure 6. The proposed web-based interface for IoT management.

3.2. System Architecture

Figure 7 illustrates the overall system architecture of the proposed approach. The system consists of the following three main components: (1) Deep learning-based vision server, (2) IoT platform server, and (3) Web-based visual interface.

First, a set of smart cameras (or light bulb modules embedded with smart cameras) need to be installed in the rooms of interest. The smart camera captures the in-door status and the recorded streams are transmitted to the deep learning vision server. The goal of the vision server is to detect Internet of Everything (IoE) devices from the video stream and subsequently classify their device types for further operations. For this, the proposed work utilizes a real-time object detector based on deep neural networks called YOLO [27]. The deep object detection and recognition network is fine-tuned for an IoE domain, which contains a fan, TV, and refrigerator using pre-trained weights from the ImageNet collection. Afterwards, for each frame from a smart camera YOLO model detects and recognizes a set of IoE devices. The detected information is then passed to an IoE server to configure a web-based interface.

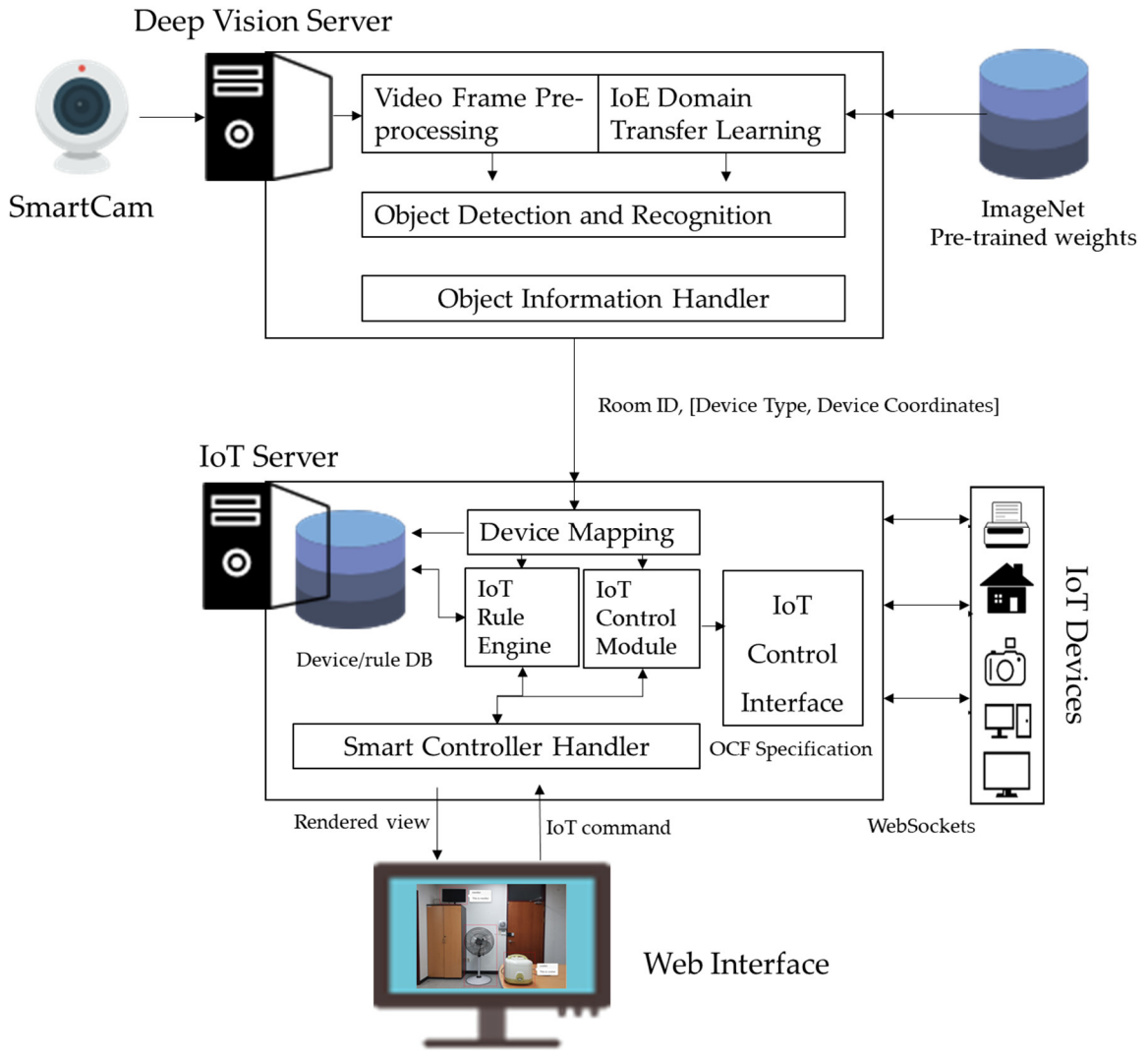


Figure 7. System architecture.

The IoT platform server takes a video stream with its classification data from the vision server and prepares a list of device information to generate context menus that will be used in the web interface. The information passed from the deep vision server is a list of items containing the type and position of the detected device. The device mapping procedure in Figure 7 is used to associate a device item in this list with its corresponding device ID that can be used for device and rule

management. The device IDs can be retrieved from the IoT device database using a room ID and device type information. The smart controller handler, shown in Figure 7, then generates a template HTML code to compose a smart controller for each device type. The design of a smart controller is determined according to the type and specification of the detected device. In this work, the device and resource specifications are derived from Open Connectivity Foundation (OCF) specification [3] which is one of the most popular IoT standards. For example, if the type of a device is "TV", then the smart controller can have buttons for "channel up/down" and "volume up/down" operations according to the specification. The generated smart controller UI codes are delivered to the web interface rendering module.

The web interface provides a video-stream based controller User Interface (UI) for monitoring the status of the room and manipulating the installed IoT devices. A context popup menu (i.e., smart controller), where users can find buttons related to device operations and rule composition, is open when a user clicks (or touches) an IoT device in the video stream. If a user clicks any device operation button, its corresponding IoT command is generated and sent to the IoT control module. The IoT control module is responsible for translating the received IoT control command into a request message complied with an IoT protocol and updating the device database. The IoT request messages are transmitted over Websocket connections between the platform server and installed IoT devices. Finally, the IoT rule engine in the IoT platform server takes charge of CRUD operations for IoT rules.

The IoT platform server mentioned in this study is from the authors' previous work which supports fundamental IoT platform features (e.g., IP-based identification and communication, device registration and management) and consists of its own database, device management module, and IoT rule engine [28]. Therefore, we do not describe the very details of these modules as it is out of the scope of this work. Rather, we focus on how to provide an interactive video stream-based interface using deep object detection techniques.

3.3. IoT Device Recognition

To recognize an IoT device from the video streams captured by smart cameras and provide context menus for each device, object detection and classification should be successfully performed. Traditional approaches considered an object detection task as a combination of region proposal and classification of bounding boxes [29–31]. These double-stage deep learning-based detectors have shown an excellent performance in terms of accuracy, but the processing speed has been a bottleneck for real-time use. On the other hand, the regression-based single-stage object detector called Yolo [32] proposed a method to directly predict boxes in a single feed-forward pass without reusing any component of the neural network or generating proposals of any kind, thereby speeding up the detector [33]. Moreover, because Yolo provides a comparable accuracy with improved speed, we adopt the Yolo object detector as part of the deep learning vision server for IoT device detection and classification.

The Yolo framework divides each input image into $S \times S$ grids and each grid predicts B bounding boxes with their confidence scores indicating whether the bounding box contains an object and class probability scores. The combination of bounding box information and class probabilities produces the final object detection and classification output. As the Yolo algorithm sees the entire image rather than the proposed regions, it is known to encode contextual information, thus learn generalizable representations of objects [33]. The original version of Yolo, however, has some limitations such as the inability to detect small objects and relatively low accuracy of localization. Redmon et al. [27,34] improved the performance of the original Yolo detector by adding some changes such as batch normalization, removal of a fully connected layer (i.e., make the network fully convolutional), multi-scale training, and customization of backbone network (DarkNet-53).

For our work, the Yolo network is configured with convolutional weights from DarkNet-53 pre-trained on ImageNet and trained with a subset of MSCOCO dataset [35]. The full dataset consists of 80 categories, some of which are not necessary for the IoT domain; hence, we chose categories suitable only in the IoT domain, such as TV/monitor, microwave, and refrigerator as shown in Figure 8.

Figure 8. Sample images used for IoT domain (refrigerator, microwave, and TV/monitor)

The vision server finally produces a Javascript Object Notation (JSON)-formatted string which contains a list of data on the detected IoT devices from the video stream. Figure 9 is an example of a generated JSON-formatted string containing three detected IoT devices. The symbols x and y indicate the coordinate of the top-left corner, and w and h represent the width and height, respectively, of the bounding box of each detected device. The generated JSON-formatted string is used by the IoT management and handler module in the IoT platform server to prepare context menus for each device.

```
[
 { // first detected object
  "x": "478", "y": "203",  "w": "144", "h": "155",
  "label": "tvmonitor"
 },
 { // second detected object
  "x": "289", "y": "45","w": "86", "h": "64",
  "label": "microwave"
 },
 { // third detected object
  "x": "2", "y": "132","w": "166", "h": "323",
  "label": "refrigerator"
 }
]
```

Figure 9. JSON-format string about the detected objects.

3.4. Smart Controller Configuration

To provide a visual interface with smart controllers (i.e., context menu) as depicted in Figure 6, how to associate an IoT device detected from the video stream with its corresponding IoT device instance managed by the IoT platform server should be addressed. In this section, we describe a method to map the devices for device management (i.e., monitor and control) and rule composition.

The smart controller (i.e., context menu in the web interface) mainly has two components: (1) buttons for device operations and (2) buttons for rule composition. Therefore, for each detected device the IoT platform server should be aware of its device ID and available operations to configure the smart controller. Figure 10 shows a sequence diagram of a device configuration for smart controllers to be used in the web interface. Once a smart camera starts recording a video stream of the room, the vision server detects a set of IoT devices using the approach mentioned in Section 3.3. On the other hand, the web interface displays the video stream on the web page while querying the controller configuration to the IoT platform server. The controller configuration is used for rendering smart controllers for each detected device; hence, it should be composed of a set of device IDs, device types, device coordinates, and device features and rules. The device configuration step in Figure 10 is to collect the IDs, types, coordinates, and available operations (features) of the detected devices, while the rule configuration step is to collect available rules of the detected devices.

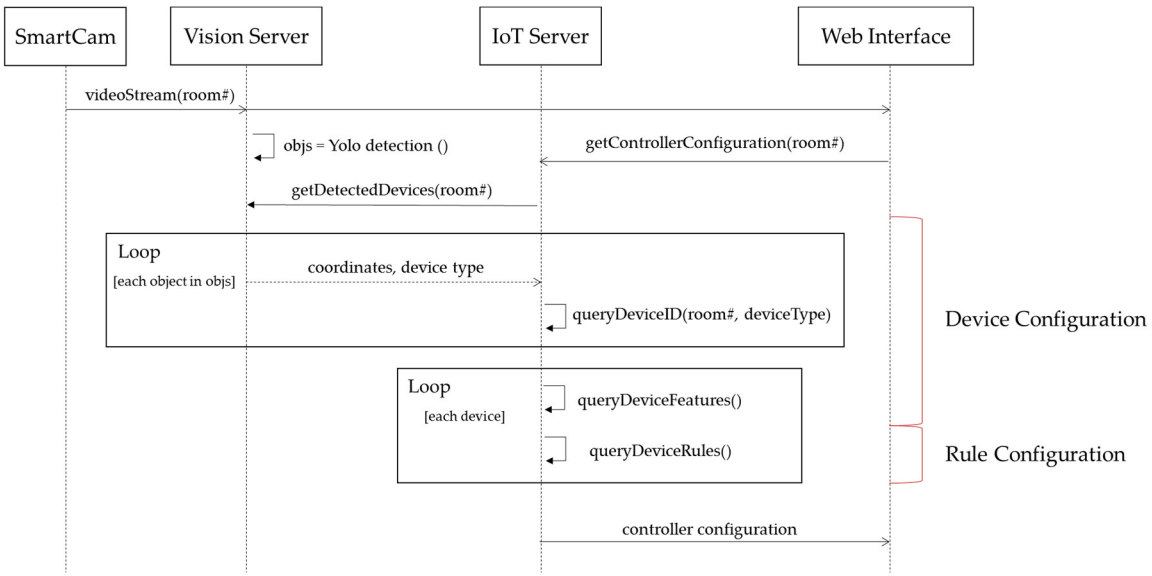


Figure 10. Sequence diagram of IoT device mapping for smart controllers.

3.4.1. Device Configuration

To obtain detailed information about the detected devices, the IoT platform sends a query to the vision server. First, the coordinates and device types of the IoT devices detected by the Yolo framework are iteratively sent to the IoT platform server as depicted in the first loop of Figure 10. The queryDeviceID() and queryDeviceFeatures() operations are used to search for a device ID and operations based on the given device type and room number. These procedures are conducted inside the IoT platform server by firing search queries to the IoT device and rule databases. The IoT platform server consists of the following four tables: thing table, spec table, room table, and rule table. The thing table stores a set of IoT devices installed in the platform and the spec table defines a list of device types and available device operations. Therefore, a natural join operation to these tables on the device type property with a room ID filter can successfully retrieve information for the device configuration (i.e., device ID and available operations) for each detected device. Figure 11 and Algorithm 1 briefly illustrates the working procedure of the device configuration step. The device ID and available operations for a given device type are retrieved in Line 1 of Algorithm 1. In Line 2, the matched rows are collected and parsed into the array of JSON-formatted string for further processing. The device ID and available operations for each detected device are appended to the list in Line 3–7. Figure 12 shows an example of the device operations supported in the IoT platform server. As mentioned above, the specifications are derived from OCF specification [3] and slightly revised to add vendor-specific customizations. The buttons of smart controllers are rendered according to the operation specifications (e.g., binarySwitch or AudioControls).

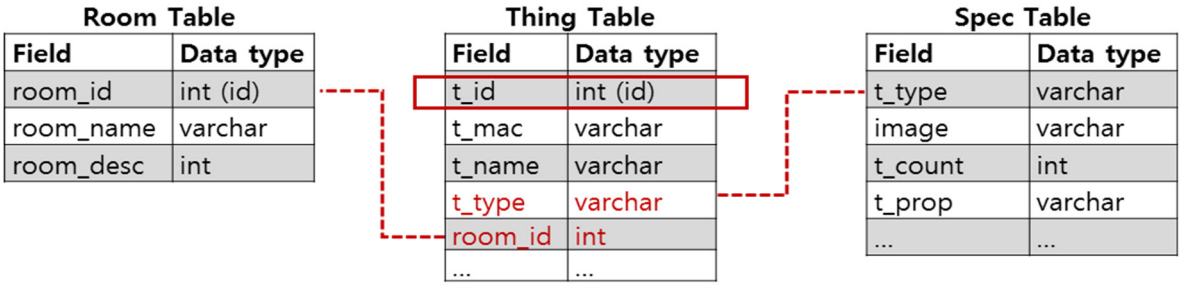


Figure 11. Procedure of device ID mapping.

Algorithm 1: Device ID and Operations Mapping

```
Input: ROOM_ID, Array (DEVICE_TYPES, COORDINATES)
Output: Array (DEVICE_ID, OPERATIONS,…)
  devices = Array (device_type, coordinates)//received from vision server
  query = "select t_id, t_type, t_prop from thing_table natural join spec_table where room_id
= ROOM_ID"
  results ← Get database rows for query and perform JSON parsing
  For result in results
  device = devices.find (item => {return item.type = result.t_type})
  device.id ← result.id
  device.props ← result.t_prop
  End
  Return devices
```

```
"definitions": { // BinarySwitch
      "oic.r.switch.binary": {
            "type": "object",
            "properties": {
                  "value": {
                        "type": "integer",
                        "description": "Status of the switch (0=off, 1 = on)"
                  }
            }
      }
}

"definitions": { //Audio Controls
      "oic.r.audio": {
            "type": "object",
            "properties": {
                  "volume": {
                        "type": "integer",
                        "description": "Volume setting of an audio rendering device.",
                        "minimum": 0,
                        "maximum": 100
                  },
                  …
            }
      }
}
```

Figure 12. Example of device operations based on OCF specification.

3.4.2. Rule Configuration

The second part of the smart controller is a set of buttons for rule composition. Using these buttons, users can add a new IoT rule about the selected device or view a list of IoT rules in which the selected device is enrolled. In contrast to the previous approaches, users do not need to spend time to locate a target device from the list of IoT devices as presented in Figure 2.

To prepare rule-related data for the selected IoT device, the platform server sends a query to the rule table which is composed of rule details and interacts with an IoT rule engine. In this work, the authors' previous work [28] is used as an IoT rule engine for the IoT platform server. The rule engine is based on the open-source RETE-based rule engine framework called Nools [36] where "condition" component and "actions" component form a single rule item. For persistence, the rules composed in the IoT rule engine are converted to a JSON-formatted string and stored in the rule database of the IoT platform server. Figure 13 shows an example of the IoT rule stored in the rule table. As can be

seen, a rule consists of "if" clause and "then" clause which define a target room, device, operator, properties and values.

Adding a new IoT rule regarding the selected device through a smart controller is straightforward. Once a user clicks the "Add rule" button on the controller, the platform server just uses the device ID for rendering interfaces for adding new rules for a specific device. To view a list of rules regarding the selected device, the platform server needs to have a set of rule IDs containing the ID of the selected device. For this, the platform server investigates the rule_details field of each rule item in the rule table to check if the ID of the selected device is included or not. Figure 14 shows an example of getting the list of rules regarding a certain device ID. The example database query is designed to check if each IoT rule contains the target device ID in its details. For example, if the target device ID is set to "t1", then rule ID 65 and 68 are returned as a result. Finally, a set of rule IDs can be passed to the web interface for rendering a list of rules regarding the selected devices. The other components of the rule_details field are related to Nools rule engine syntax. Details about the Nools rule engine or the authors' previous rule engine can be found in [28].

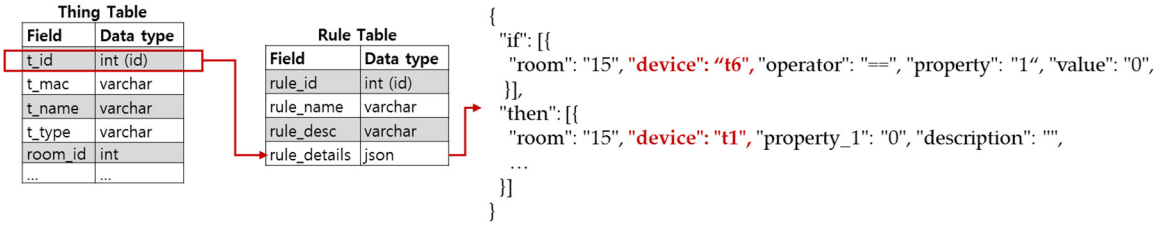


Figure 13. Rule table and examples of rule details.

query = select rule_id from rule_table where rule_details like '%"DEVICE_ID"%'

Rule_id	Rule_details
64	{"if": [{"type": "Delay", "detail": "1", "repeat": "None"}], "then": [{"room": "15", "count": "1,"device": "t3", "property_1": "0"}]
65	{"if": [{"room": "15", "value": "0", "device": "1196", "operator": "==", "property": "1"}], "then": [{"room": "15", "count": "3", **"device": "t1"**, "property_1": "1", "property_2": "2", "property_3": "1"}]
68	{"if": [{"room": "13", "value": "8966261817", "device": "weather", "operator": "==", "property": "weather_today"}], "then": [{"room": "15", "count": "3", **"device": "t1"**, "property_1": "0", "property_2": "2", "property_3": "1"}]

Figure 14. RuleID lookup example

3.5. Interactive Web Interface

The proposed web interface mainly consists of an indoor video stream and smart controllers for each detected device. Once the camera view for a room is requested, the web interface communicates with the vision server to get a video stream of the room as stated in Section 3.2. The streamers such as mjpg-streamer [37] or polling-based streamers can be used for video streaming. In this study, we have adopted a polling-based video streaming approach for the sake of simplicity.

Figure 15 presents a template of the proposed web interface. The dashed boxes indicate bounding boxes for the detected IoT devices. According to the coordinates data passed from the vision server, the web interface adds each of the bounding boxes on the video stream view. The bounding boxes only have a click event handler to open a smart controller and they are not visualized on the video view when initialized. If a user clicks the area of a bounding box, its corresponding smart controller appears on the view. The solid line box of Figure 15 represents the template for a

smart controller and Figure 16 depicts the rendered smart controllers for refrigerator and TV device types. The buttons for device properties and operations are rendered according to the specifications retrieved during the device configuration step. Finally, the web interface for rule composition (i.e., adding a new rule or viewing a list) is also rendered according to the device ID delivered from the IoT platform server.

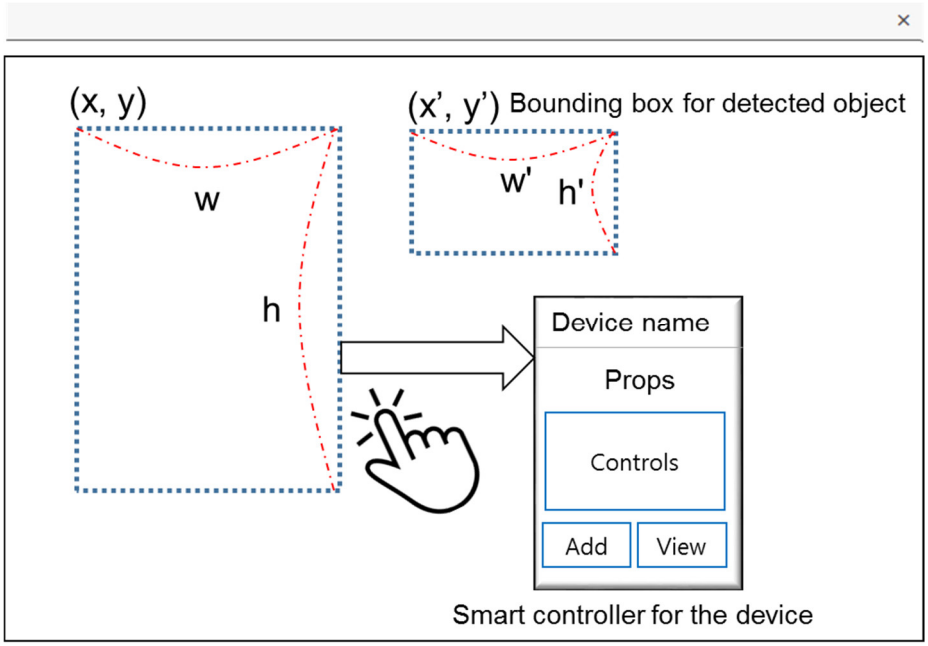


Figure 15. Web interface template.

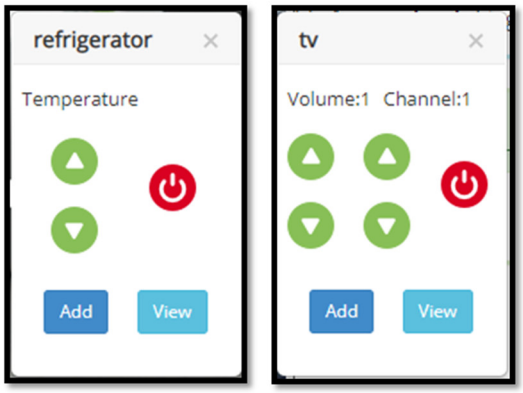


Figure 16. Smart controller UI.

4. Implementation

4.1. Prototype Implementation

The core modules of the IoT platform server were implemented based on Node.JS backend. The web framework of the IoT platform server was implemented using Express engine. The database and rule engine were constructed using MySQL and Nools framework, respectively. The vision server was developed using Yolo-v3 [27] python wrapper and python web framework. The Yolo detector was trained with the MSCOCO dataset [35] with the default hyper-parameters. The frontend (i.e., web interface) was designed using HTML5, CSS3, and jQuery library.

4.2. Use Cases

4.2.1. Room and Device Monitoring

Figure 17 depicts a web interface where the video stream of a room is open and the smart controllers for a microwave and TV are activated. It is shown that the UI of a smart controller is rendered according to the type of the selected device. For example, a microwave only has a binarySwitch operation (i.e., turn on/off), thus only a power button is located in the smart controller for the microwave. On the other hand, a TV has three properties: binarySwitch, audioControl, and channelControl. Therefore, the smart controller for a TV consists of three different kinds of button groups. Figure 17 also shows that the proposed system can successfully track and recognize an IoT device even though its location is changed. For example, the smart controller for the microwave is successfully activated even though its location is changed.

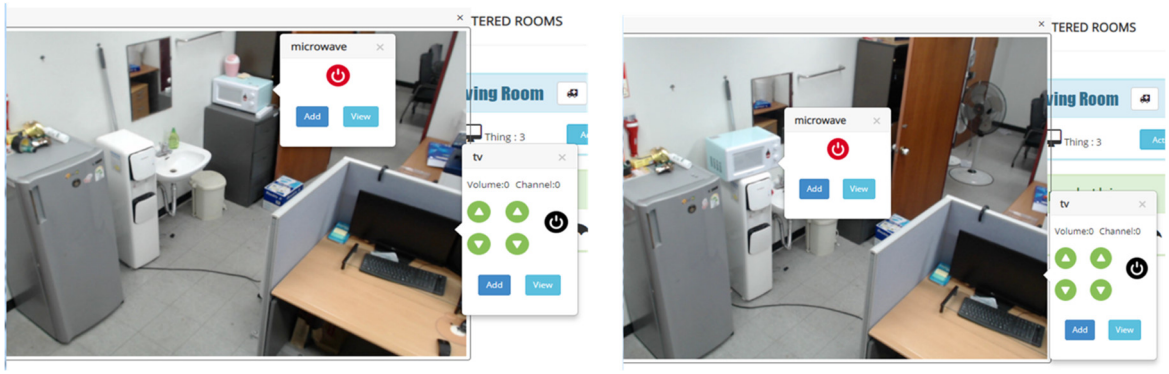


Figure 17. Monitoring room and device status: (**a**) before moving the microwave; (**b**) after moving the microwave.

4.2.2. Device Control

With a smart controller, a user can directly manipulate the status of the selected device by touching the buttons. For example, if a user touches the power button in the smart controller, the IoT command for the selected device is generated according to the power status of the device and subsequently sent to the target device by the IoT handler in the platform server. After receiving an acknowledgment message from the target device, the platform server updates the thing table to store the status of the device. Figure 18 shows how the thing table is updated after the power button is pressed. The value of the binarySwitch field has been changed from 0 (off) to 1 (on).

```
+------+--------+-----------+---------+----------------+---------------+
| t_id | t_name | t_type    | room_id | property_value | property_type |
+------+--------+-----------+---------+----------------+---------------+
|    1 | tv     | tvmonitor |      15 |              0 | binaryswitch  |
+------+--------+-----------+---------+----------------+---------------+
1 row in set (0.00 sec)
```

(a)

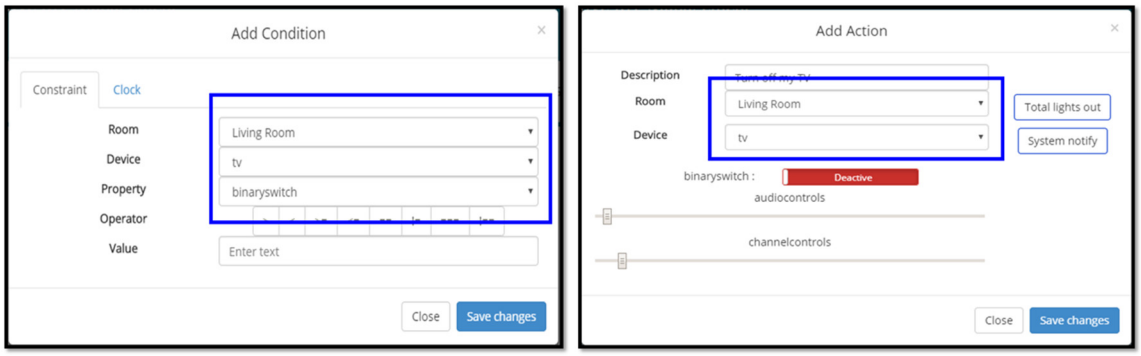
```
+------+--------+-----------+---------+----------------+---------------+
| t_id | t_name | t_type    | room_id | property_value | property_type |
+------+--------+-----------+---------+----------------+---------------+
|    1 | tv     | tvmonitor |      15 |              1 | binaryswitch  |
+------+--------+-----------+---------+----------------+---------------+
1 row in set (0.00 sec)
```

(b)

Figure 18. Manipulating device status: (**a**) initial status of a TV; (**b**) changed status of the TV.

4.2.3. Adding and Viewing Rules

The "add" and "view" buttons in the smart controller can be used for composing new IoT rules or viewing a list of IoT rules regarding the selected device. Compared to the previous approaches for IoT rule composition, the proposed system helps users intuitively locate a target device to be used for rule composition. When composing a condition or an action component of a rule, the ID of the selected device and its default field are automatically set for quick manipulation. Figure 19 shows an example of the interface for setting conditions to compose a new rule with the proposed system. As depicted in Figure 19a, the input fields for a room, device, and its attributes for the condition part are automatically set according to the device ID that a user selects. Similarly, the input fields for a room, device, and its available operations and properties for the action part are accordingly set and rendered. On the other hand, users can easily search for a list of IoT rules related to the selected device using the smart controller. For example, Figure 20 presents a list of IoT rules related to the selected TV device. Because the search operation for IoT rules is performed based on the device ID, only the IoT rules for a specific device can be returned to the users.



(**a**) (**b**)

Figure 19. Viewing a list of IoT rules for the selected IoT device: (**a**) composing the condition part of the IoT rules with the controller; (**b**) composing the action part of the IoT rules with the controller [9].

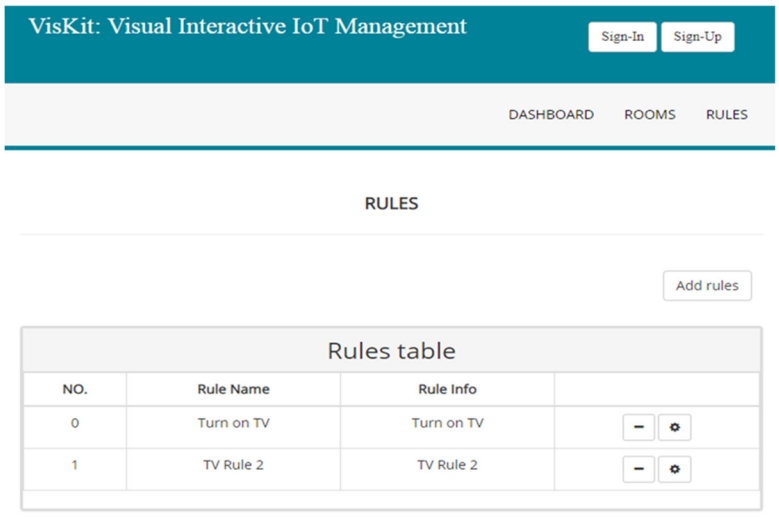


Figure 20. Viewing a list of IoT rules for the selected IoT device.

4.3. User Studies

To evaluate the usability of the proposed system, we conducted a user study on using IoT management systems. For the user study, we recruited 10 university students (7 female and 3 male) aged 22–26 years. The participants were asked to use (1) our system with smart controllers, (2) our system without smart controllers, and (3) ARTIK cloud [10] to complete the following IoT tasks:

- Task 1: Find a set of rules about a specific device.

- Task 2: Find a specific device and add a new rule for it.
- Task 3: Find a specific device and turn it off.

Before conducting the tasks, all the participants were given the user guide about the systems and asked to explore the menus and features of each system. During the user study, we have measured the user performance (i.e., elapsed time and the number of clicks) to complete the tasks.

Table 1 presents the result of the user study of Task 1. For this task, we measured the elapsed time to discover all the matching IoT rules. The result shows that it takes much less time to complete Task 1 when using the proposed system with a smart controller, i.e., four times faster than the others. However, we could not find any significant difference between the proposed system without a smart controller and ARTIK cloud platform. Without the smart controller, the proposed system provides a list-based interface to users, which results in the degradation of user performance.

Table 1. Experimental results of Task 1 (elapsed time, unit: seconds).

System	Min	Max	Average	Std.dev
Proposed with smart controller	5.5	12.1	7.88	2.92
Proposed without smart controller	20.85	45.0	32.33	8.75
ARTIK	21.21	47.23	34.53	9.55

Tables 2 and 3 summarize the results of Task 2. For this task, we measured the elapsed time and number of user clicks to add a specific IoT rule. The participants were asked to generate a new IoT rule according to the experimenter's instruction. On an average, it took 27.25 s for the proposed system with smart controllers to complete Task 2, which is 17.5% and 15% faster than the proposed system without smart controllers and ARTIK platform, respectively. Moreover, the proposed system with smart controllers required less number of clicks to complete the task.

Table 2. Experimental results of Task 2 (elapsed time, unit: seconds).

System	Min	Max	Average	Std.dev
Proposed with smart controller	22.64	31.30	27.25	3.01
Proposed without smart controller	27.32	36.03	33.02	4.83
ARTIK	26.86	38.29	32.03	4.23

Table 3. Experimental results of Task 2 (number of clicks).

System	Min	Max	Average	Std.dev
Proposed with smart controller	7	9	7.86	0.69
Proposed without smart controller	10	12	10.71	0.75
ARTIK	11	13	11.85	0.69

Tables 4 and 5 present the results of Task 3. Similar to Task 2, we measured the elapsed time and number of user clicks to turn off a specific device. As can be seen from Table 4, the proposed system with smart controllers achieved the best performance in terms of elapsed time. It is worth noting that the number of clicks between the proposed system with smart controllers and the ARTIK platform does not show a difference, but the smart controller UX resulted in a 25% performance gain with respect to elapsed time.

Table 4. Experimental results of Task 3 (elapsed time, unit: seconds).

System	Min	Max	Average	Std.dev
Proposed with smart controller	4.23	7.45	5.4	1.03
Proposed without smart controller	6.35	8.21	7.17	0.59
ARTIK	6.75	8.45	7.44	0.46

Table 5. Experimental results of Task 3 (number of clicks).

System	Min	Max	Average	Std.dev
Proposed with smart controller	3	4	3.2	0.42
Proposed without smart controller	5	6	5.6	0.51
ARTIK	3	4	3.4	0.51

Finally, we conducted a survey on user satisfaction about the usability of the system with the Likert-5 scale (i.e., 1: Very unsatisfactory, 4: Very satisfactory). As shown in Figure 21, the proposed system with the smart controller achieved the best scores among the systems considered. However, the proposed system without smart controllers achieved a lesser score (i.e., 3.1) than the ARTIK platform (i.e., 3.6). This result came from the fact that the ARTIK platform has a more fancy and professional interface than the proposed system. It was found that the users preferred a system with a better interface if there were no significant differences in performance between the systems.

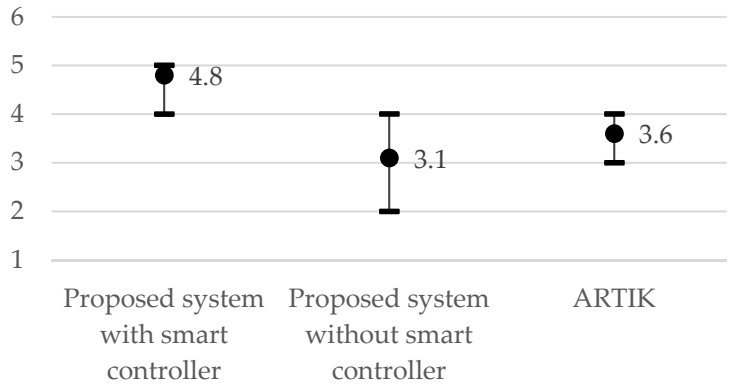


Figure 21. User satisfaction.

In summary, the experimental results show that most of the participants performed the tasks more efficiently when using the proposed system. As mentioned above, the proposed system can help users intuitively locate a target device with a video stream and smart controllers, which result in the improvement of the system usability.

4.4. Comparison with Previous Work

Table 6 summarizes the comparison results with previous IoT systems from various perspectives. As mentioned in Introduction, most recent IoT systems provide a simple grid or list-based interface which is not efficient for users to manage the devices and rules. Moreover, some systems [2,20–23] provide an interface for device management only. Although some studies [12–14,22] provide graphical components for presentation, their functionalities are limited to manage only sensor data or support only a web-based interface or device management feature. Compared to the previous studies, the proposed system supports both device and rule management through a

web/mobile interface with a video view. In particular, the smart controller feature based on the deep learning object detection approach has drastically improved the usability of the system as demonstrated by experimental results.

Table 6. Comparison with previous work.

Method	Device	Accessibility	Interface	Usability	Domain	Features [1]
[2]	Sensor and actuator	Web and mobile	List/Grid	Low	Smart Home	DM
[5–10]	Sensor and actuator	Web and mobile	List/Grid	Medium	General	DM + RM
[11]	Sensor and actuator	Web and mobile	List/Grid (ARTIK)	Medium	Smart Home	DM + RM
[12]	Sensor only	Web and mobile	Icons	Low	Smart Factory	DM + RM
[13]	Sensor only	Web only	Icons	Medium	General	DM + RM
[14]	Sensor and actuator	Web only	Icons	Medium	Smart Home	DM + RM
[17]	Sensor and actuator	Platform specific	Platform specific	Platform specific	Smart Factory	DM + RM
[20]	Sensor and actuator	Mobile only	List/Grid	Low	Smart Home	DM
[21]	Sensor and actuator	Web and mobile	List/Grid	Low	Smart Home	DM
[22]	Sensor only	Web and mobile	Icons	Medium	Smart Health	DM
[23]	Sensor only	Web and mobile	List/Grid	Medium	Smart Factory	DM
[24]	Sensor only	Web and mobile	List/Grid (ARTIK)	Medium	Smart Health	DM+RM
Ours	Sensor and actuator	Web and mobile with video view	Icons	High	Smart Home	DM+RM

DM: Device Management, RM: Rule Management

5. Discussion and Conclusions

In this paper, we proposed an interactive visual interface for an intuitive and efficient IoT device management. The proposed system captures the indoor status of a room and recognizes IoT devices to provide smart controllers on the video stream. Through the web interface, users can not only monitor and control the IoT devices installed in the room but also manage the IoT rules of the selected devices.

The main strength of this work can be summarized as follows: First, the proposed system supports both monitoring the status of sensor and actuator devices installed in a room and handling actuations with various conditions. In addition to the physical devices, we also plan to expand the scope of the system to support external services and applications such as social network services, 3rd party IoT cloud solutions, and IFTTT applets. Second, the main functionalities of the proposed system are provided through a graphical user interface for improved usability. As stated in the Introduction, the GUI-based presentation can overcome the limitations of a traditional text-based grid or list style interfaces. Third, through the integration between the IoT platform server and deep learning-based object detection approach, the smart controllers for each device type are accordingly configured. It was shown that the prototype implementation tracks an IoT device with the deep object detection method so that the smart controllers are correctly activated even if the location of the device is changed. Finally, as demonstrated by the experimental results, the video view with smart controller interfaces successfully improved the performance of the system in terms of efficiency and user satisfaction. The quantitative experimental results showed that the proposed system with smart

controllers achieved an average 40% performance gain in terms of efficiency. The results of the user survey also confirmed that the proposed system is more usable than competitive solutions.

However, the proposed system still suffers from the following limitations. First, the proposed architecture requires the installation of a smart camera to provide video streaming and smart controllers, which is not possible in certain environments. A low-light environment where general computer vision algorithms cannot work or an environment with limited network bandwidth where real-time video streaming is impossible are typical examples. To address these issues, we plan to study the extension of previous deep learning-based object detection approaches for a low-light environment and the utilization of compressed video streaming strategies. Second, as shown in Figures 6,17, the current video view can cover only a limited angle. Therefore, multiple cameras can be required to fully cover the entire space of a single room in the worst case. As future work, we plan to design and implement a smart light module embedded with a 360-degree smart camera to provide better functionalities. Also, the improved version of deep object detection and classification method will be studied for handling a 360-degree video stream environment. Third, due to the lack of a large dataset suitable for an IoT domain, the current proposed system supports only a limited number of device types such as TV/monitor, microwave, and refrigerator. To support more device types, the Yolo framework needs to be trained and fine-tuned with more image samples of IoT devices. Finally, if the multiple devices of the same type are installed in a single room, it is impossible to correctly detect the ID of each device, which results in the failure of IoT interaction. To handle such a complex environment, we plan to record and track the location of each device.

Author Contributions: All three authors have significantly contributed to the work presented in this paper. Conceptualization and methodology by J.-W.J.; Software development, validation, experiment by C.-E.H., S.-H.L., and J.-W.J.; Writing, reviewing, editing, and supervision by J.-W.J.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2016R1D1A1B03931672).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Husain, S.; Prasad, A.; Kunz, A.; Papageorgiou, A.; Song, J. Recent Trends in Standards Related to the Internet of Things and Machine-to-Machine Communications. *J. Inf. Commun. Converg. Eng.* **2014**, *12*, 228–236.
2. Santoso, L.W.; Lim, R.; Trisnajaya, K. Smart Home System using Internet of Things. *J. Inf. Commun. Converg.* **2018**, *16*, 60–65.
3. OCF SPECIFICATION 2.0. Available online: http://openconnectivity.org (accessed on 29 December 2018).
4. Standards for M2M and the Internet of Things. Available online: http://onem2m.org (accessed on 29 December 2018)
5. IBM Watson IoT Platform. Available online: http://internetofthings.ibmcloud.com (accessed on 29 December 2018).
6. Thing+. Available online: http://thingplus.net/ (accessed on 29 December 2018).
7. AWS IoT. Available online: https://aws.amazon.com/iot/ (accessed on 29 December 2018).
8. OpenHAB. Available online: https://www.openhab.org/ (accessed on 29 December 2018).
9. Samsung SmartThings. Available online: http://www.smartthings.com (accessed on 29 December 2018).
10. Samsung ARTIK Cloud Available online: http://artik.cloud (accessed on 29 December 2018).
11. Heon-Je, L.; Sung-Soon, P.; Kyung-Hoon, K. Expandable IoT integrated Management and Rule based automation System. In Proceedings of the KIISE Winter Conference, PyeongChang, Korea, 17–19 December 2015; pp. 1627–1629.
12. Jang, K.H.; Park, J.I. Study on IoT System Design for Factory Site Management Service by Event—Condition—Action Rules. In Proceedings of the Korean Operations Research And Management Society Conference, Jeju, Korea, 8–11 April 2015; pp. 1459–1465.
13. Kim, K.; Lee, H.; Cho, D.-S. Real-time Event Processing Role Management System for IFTTT Service. *J. Korea Multimed. Soc.* **2017**, *8*, 1379–1386.

14. Byun, J.M.; Park, S.S.; Kim, K.H. Designing and Implementation of Rule-based Automation control and management Framework considering the IoT devices scalability. In Proceedings of the KIISE Winter Conference, PyeongChang, Korea, 17–19 December 2015; pp. 398–400.
15. Díaz López, D.; Blanco Uribe, M.; Santiago Cely, C.; Tarquino Murgueitio, D.; Garcia Garcia, E.; Nespoli, P.; Gómez Mármol, F.; Díaz López, D.; Blanco Uribe, M.; Santiago Cely, C.; et al. Developing Secure IoT Services: A Security-Oriented Review of IoT Platforms. *Symmetry* **2018**, *10*, 669.
16. Garcia, C.G.; Meana-Llorián, D.; Bustelo, C.P.G.; Lovelle, J.M.C. A review about Smart Objects, Sensors, and Actuators. *Int. J. Interact. Multimed. Artif. Intell.* **2017**, *4*, 7–10.
17. Molano, J.I.R.; Lovelle, J.M.C.; Montenegro, C.E.; Granados, J.J.R.; Crespo, R.G. Metamodel for integration of Internet of Things, Social Networks, the Cloud and Industry 4.0. *J. Ambient Intell. Humaniz. Comput.* **2018**, *9*, 709–723.
18. Dinh, T.; Kim, Y. An Efficient Interactive Model for On-Demand Sensing-As-A-Servicesof Sensor-Cloud. *Sensors* **2016**, *16*.
19. Cirani, S.; Ferrari, G.; Mancin, M.; Picone, M. Virtual Replication of IoT Hubs in the Cloud: A Flexible Approach to Smart Object Management. *J. Sens. Actuator Netw.* **2018**, *7*, 16.
20. Al-Kuwari, M.; Ramadan, A.; Ismael, Y.; Al-Sughair, L.; Gastli, A.; Benammar, M. Smart-home automation using IoT-based sensing and monitoring platform. In Proceedings of the IEEE 12th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG 2018), Doha, Qatar, 10–12 April 2018; pp. 1–6.
21. Govindraj, V.; Sathiyanarayanan, M.; Abubakar, B. Customary homes to smart homes using Internet of Things (IoT) and mobile application. In Proceedings of the International Conference On Smart Technologies For Smart Nation (SmartTechCon), Bangalore, India, 17–19 August 2017; pp. 1059–1063.
22. Memedi, M.; Tshering, G.; Fogelberg, M.; Jusufi, I.; Kolkowska, E.; Klein, G. An Interface for IoT: Feeding Back Health-Related Data to Parkinson' s Disease Patients. *J. Sens. Actuator Netw.* **2018**, *7*, 14.
23. Wu, F.; Wu, T.; Yuce, M.R. An Internet-of-Things (IoT) Network System for Connected Safety and Health Monitoring Applications. *Sensors* **2018**, *19*, 21.
24. Hong, J.; Morris, P.; Seo, J. Interconnected Personal Health Record Ecosystem Using IoT Cloud Platform and HL7 FHIR. In Proceedings of the IEEE International Conference on Healthcare Informatics (ICHI), Park City, UT, USA, 23–26 August 2017; pp. 362–367.
25. Zhao, Z.; Martin, P.; Jones, A.; Taylor, I.; Stankovski, V.; Salado, G.F.; Suciu, G.; Ulisses, A.; de Laat, C. Developing, Provisioning and Controlling Time Critical Applications in Cloud. In Proceedings of the Advances in Service-Oriented and Cloud Computing, Oslo, Norway, 27–29 September 2018; pp. 169–174.
26. Heikkinen, A.; Pääkkönen, P.; Viitanen, M.; Vanne, J.; Riikonen, T.; Bakanoglu, K. Fast and Easy Live Video Service Setup Using Lightweight Virtualization. In Proceedings of the 9th ACM Multimedia Systems Conference, Amsterdam, Netherlands, 12–15 June 2018; pp. 487–489.
27. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arxiv: 1804.02767.
28. Lee, S.-H.; Hwang, C.-E.; Jeong, J.-W. Design and Implementation of Nools-based Rule Engine for Smart IoE Platform. *J. Korea Inst. Inf. Electron. Commun. Technol.* **2018**, *11*, 379–387.
29. Girshick, R. Fast R-CNN. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 13–16 December 2015; pp. 1440–1448.
30. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Proceedings of the Advances in Neural Information Processing Systems 28, Montréal, QC, Canada, 7–12 December 2015; pp. 91–99.
31. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 24–27 June 2014; pp. 580–587.
32. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 779–788.
33. Agarwal, S.; Terrail, J.O.D.; Jurie, F. Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks. *arXiv* **2018**, arXiv: 1809.03193.
34. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 6517–6525.

35. Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In Proceedings of the Computer Vision, Zurich, Swiss, 6–12 September 2014; pp. 740–755.
36. Nools: RETE-based Rules Engine. Available online: http://noolsjs.com (accessed on 29 December 2018).
37. Mjpg-streamer. Available online: https://github.com/jacksonliam/mjpg-streamer (accessed on 29 December 2018).