IoTeach:Learning Support System for IoT Programming by Integrating Real Devices and Sequential Contents

TOMOHIRO KAWATANI, Future University Hakodate, Japan
KOJI TSUKADA, Future University Hakodate, Japan
KAZUTAKA KURIHARA, Tsuda University, Japan

In recent years, as the IoT has become popular, there are more requirements for sharing operation/mechanisms of IoT devices, such as Arduino and M5Stack. In addition, because of the coronavirus pandemic, many educational institutions adopted online lectures, such as on-demand class and online class using video conference systems. For IoT programming education, these methods have problems such as lack of linkage with real-world devices and source codes. In this study, we propose a system called "IoTeach", which supports learning of IoT programming by attaching a script language on sequential contents such as videos and slides shared on the Web. The IoTeach can link videos and slides with real-world IoT devices and source code. In this paper, we describe the concept and implementation of the system.

CCS Concepts: • **Human-centered computing** → **User interface programming**; • **Hardware**; • **Software and its engineering** → *Software prototyping*;

Additional Key Words and Phrases: IoT, Education support systems, video, Interactive systems

ACM Reference Format:
Tomohiro Kawatani, Koji Tsukada, and Kazutaka Kurihara. 2018. IoTeach:Learning Support System for IoT Programming by Integrating Real Devices and Sequential Contents. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

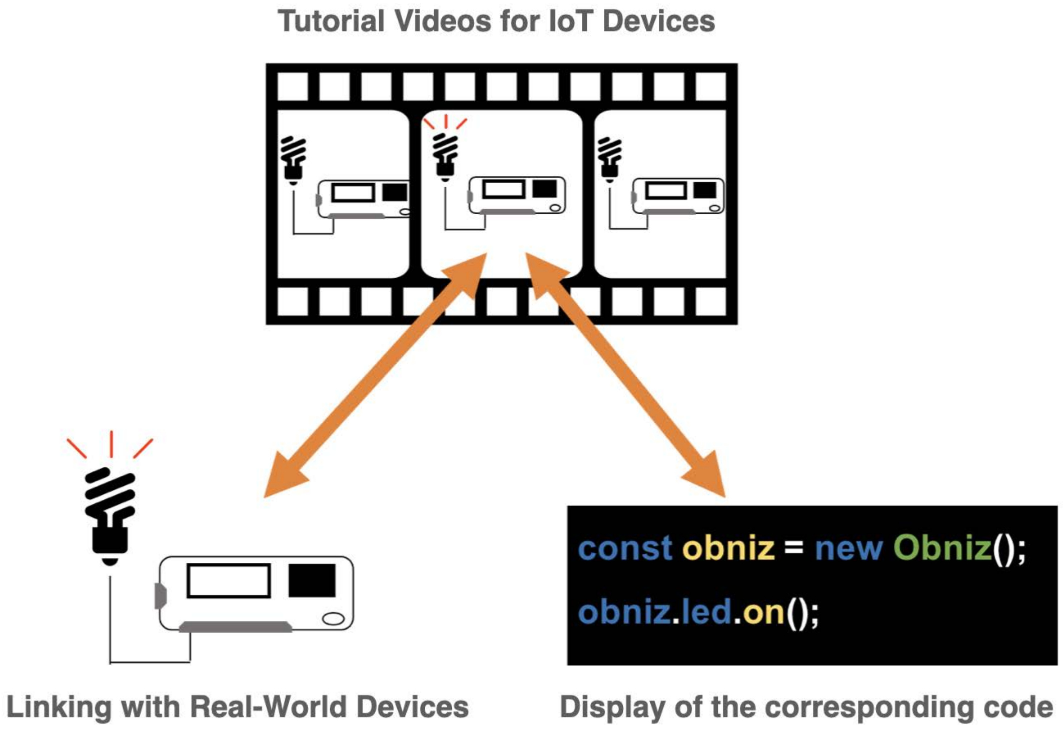
1 INTRODUCTION

As typified by the term IoT, systems that link everyday objects with the Internet are becoming more common. In addition, underlying technologies such as Arduino, Raspberry pi, and M5Stack, which support prototyping of IoT devices, have been enhanced, and the demand for educational support for these technologies are increasing day by day. However, it is difficult to share the behavior and mechanisms of prototyped IoT devices with others who are not there in space and time. Currently, tutorial videos are generally used. For example, M5Stack has published many tutorial videos of its IoT devices[1].

On the other hand, compared to conventional documentation format, tutorial videos lack of linkage to the source code and other elements that actually operate the device. Although users can understand the behavior of a device by watching a video, it is not easy to reproduce the behavior on an actual device at hand. Since it is difficult to express which part of the source code is causing the behavior of a device in a video, it is necessary to refer to the existing documentation to find the source code and transfer it to the device at hand from a dedicated development environment.

[1]https://www.youtube.com/c/M5Stack/

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX


Tutorial Videos for IoT Devices
Linking with Real-World Devices
Display of the corresponding code

```
const obniz = new Obniz();
obniz.led.on();
```

Fig. 1. Overview of IoTeach: it performs IoT devices operating and source code presentation by synchronizing scripts corresponding to the progress of videos and slides on the Web.

Furthermore, with the coronavirus pandemic, educational institutions are rapidly moving programming class online, increasing the needs for online programming education support. Especially in online classes using video conference system, instructors face problems such as difficulty in grasping and controlling the status of computers and IoT devices in the hands of students.

In this study, we propose IoTeach, a system that sync sequential contents such as videos and slides shared on the Web with a scripting language executed in correspondence to the progress of the content. This allows the IoT device in the viewer's hand to be linked according to the contents, and the corresponding source code can also be presented (e.g., Figure 1).

By using IoTeach, we expect to increase the viewer's understanding of devices and improve their motivation.

2 RELATED WORK

We present several examples of synchronization of scripts to sequential contents. "Sikuli"[6]is a system that uses images in programs to enable automated aids and searches, and "Text Alive"[4] is a web service that creates animated videos based on analyzed music information.

Another example of linking sequential contents with real-world devices is "Songle Sync"[3]. This platform allows users to control animations created based on music with various devices in the real world, creating an experience where music and physical space are fused together.

"Udemy"[2] and "Arduino Editor - Web Editor"[1] are examples of learning sites for programming and micro-controllers. These sites allow users to learn programming for microcontrollers online, but they do not allow users to run devices on the web service, and they require special plug-ins and software to be installed on the PC in order to run the devices. This makes it time-consuming and difficult for beginners to try it out.

Although mechanisms for synchronization of scripts to sequential contents(especially music) and for linking sequential contents with real-world devices have existed, but few attempts have been made to integrate the two to support learning. Our research is unique in that it supports IoT programming learning by sync scripts with sequential content such as videos and slides, and linking the behavior of devices in the content with real-world devices and source code.

3 PROPOSAL

3.1 SYSTEM CONCEPTS

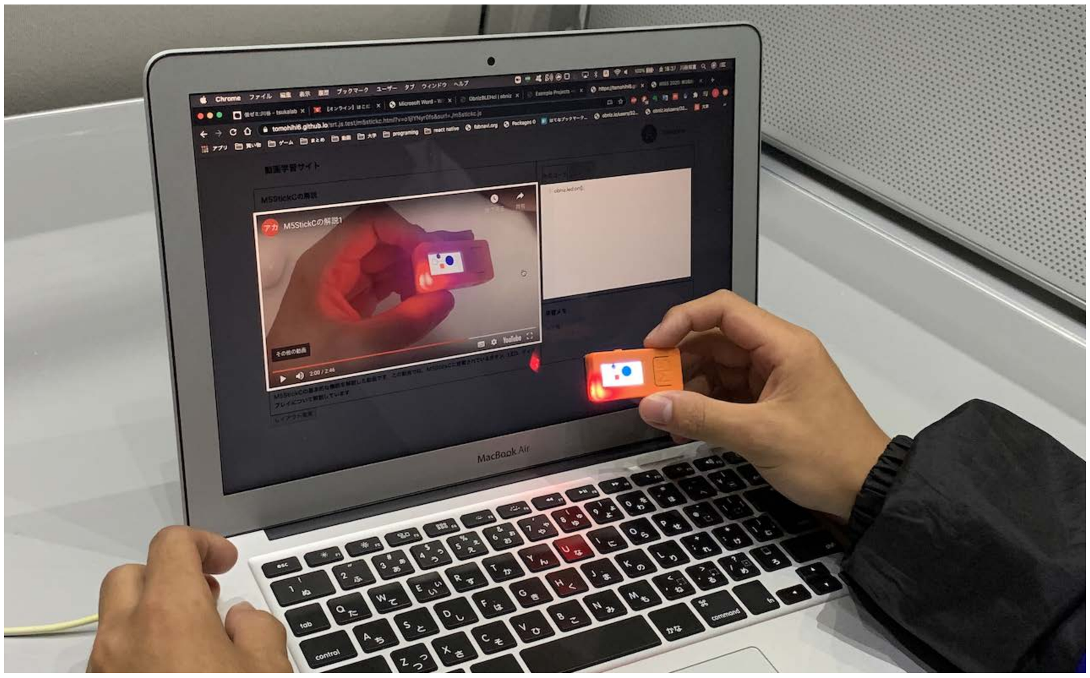


Fig. 2. An example of the use of IoTeach

There are three major elements in IoTeach: sequential contents (videos/slides) and source code on the Web, and IoT devices (microcontrollers) in the real world. Users can see the source code corresponding to the behavior of the devices in the video, making it easier to understand the details of the control. In addition, we believe that the experience of seeing a device in action simply by watching a video will motivate users, especially novice users, to learn more. An example of the use of the system is shown in Figure 2. Here, a user is watching a tutorial video of an IoT microcontrollers on a PC. When the LEDs of the microcontroller in the video light up, the LEDs of the user's microcontroller also light up at same time. In addition, the corresponding source code is shown on the editor on the right side of the screen.

3.2 SYSTEM REQUIREMENTS

In this research, we aim to develop a system that enables smooth linking between contents and devices, and between contents and source code when viewing sequential contents (video/slides). The following three system requirements were set.

Device operates at the user's hands
Not only seeing the behavior of the device on the video/slide, but also seeing the device actually in your hand behave in the same way. Deepen your understanding by actually having the device in your hand behave in the same way or move based on user input. For example, you can understand the sensitivity of sensors and the driving range of actuators through experience. The experience that the device in hand moves just by watching a video is thought to improve the willingness to learn, especially for beginners. We believe that this will lead to an improvement in the willingness to learn, especially for beginners.

Display of source code according to device behavior
As mentioned in Chapter 1, it is difficult to check which part of the source code realizes the behavior of the device on the video/slide. Therefore, the corresponding source code is displayed in the text area of the Web application at the timing when the device moves on the video, allowing editing operations such as copying, etc., so that the video and source code can be smoothly linked.

Guidance of points of interest
In IoT programming learning, it is necessary to be aware of multiple elements such as contents (videos/slides), source code, and the device at hand, which can easily lead to confusion about where to look. To solve this problem, we have developed a function that guides the user's attention by highlighting the elements that require attention in the system.

4 IMPLEMENTATION

In this study, we developed two types of systems, a video version system and a slide version system for on-demand/live classes. Both systems were implemented as Web applications. The front end is build using HTML, CSS, JavaScript, etc., and Firebase[2] is used for hosting and database management. In this section, after introducing IoT devices used in the system and the synchronization scripts, the video/slide version of the system is described.

4.1 IoT DEVICES

The three requirements for IoT devices to be used in the system are as follows.

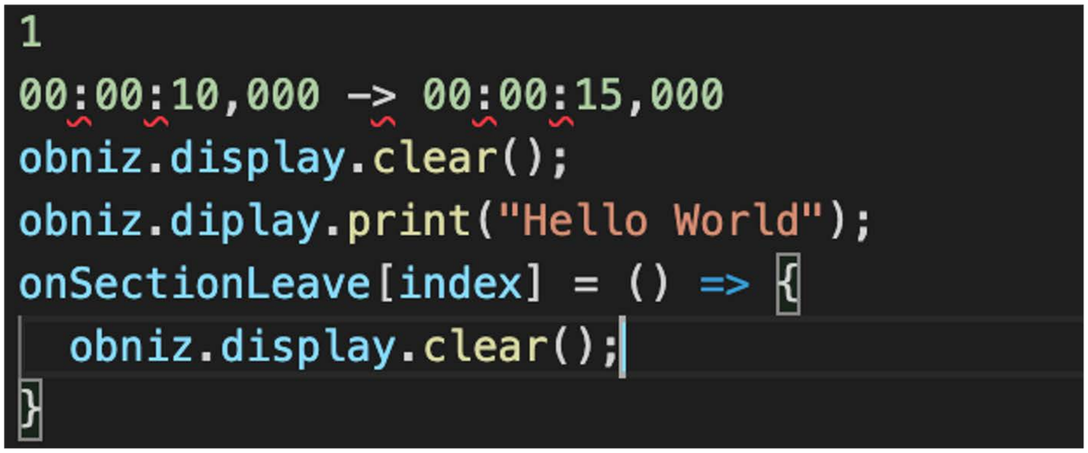
(1) Expandability to connect external sensors and actuators.
(2) Internet access is available via Wi-Fi, etc.
(3) Can run/rewrite programs via the Internet.

These requirements can be achieved with an ESP32 microcontroller or Arduino+Wi-Fi shield. However, the third requirement requires time and effort for firmware development.

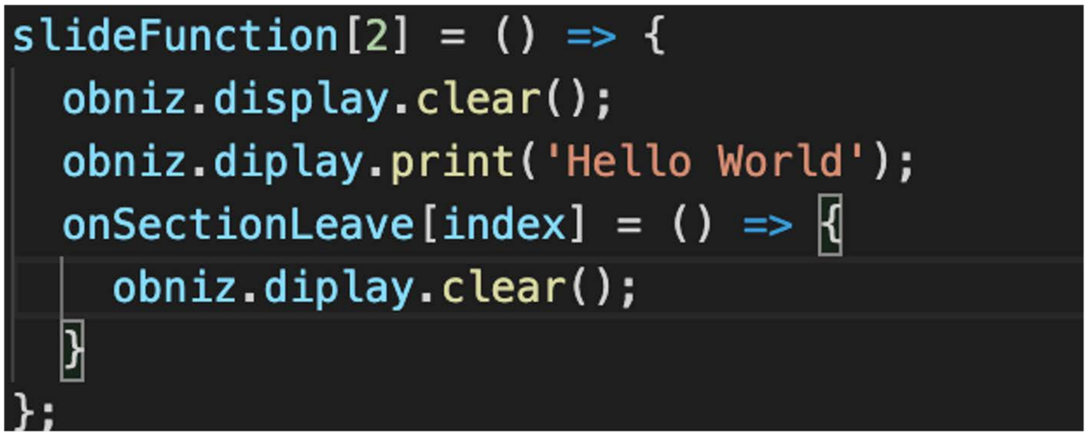
Therefore, we decided to use the microcontroller in which obnizOS[3] is installed in this study. obnizOS is a firmware that runs on IoT microcontrollers. It has a unique ID for each device. By specifying the ID from the cloud API, it is possible to control the built-in sensors/actuators and input/output ports of a specific device.

IoTeach can be used without any complicated preparation as long as the learner has the corresponding obniz-based microcontroller.

[2]https://firebase.google.com/
[3]https://obniz.com/ja/products/obnizos



```
1
00:00:10,000 -> 00:00:15,000
obniz.display.clear();
obniz.diplay.print("Hello World");
onSectionLeave[index] = () => {
  obniz.display.clear();
}
```



```
slideFunction[2] = () => {
  obniz.display.clear();
  obniz.diplay.print('Hello World');
  onSectionLeave[index] = () => {
    obniz.diplay.clear();
  }
};
```

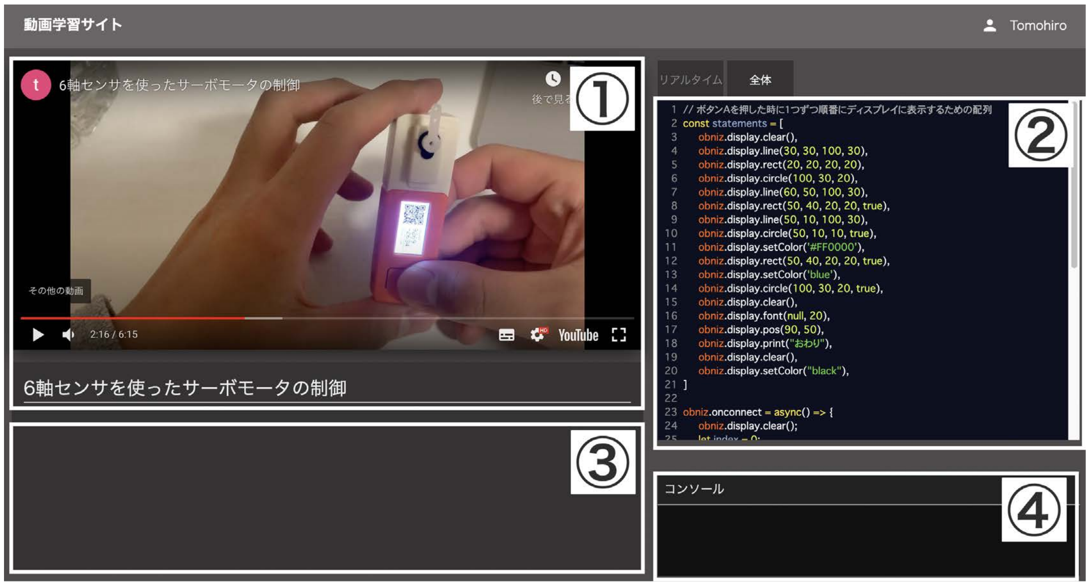
Fig. 3. Example of script description (top: video version, bottom: slide version)

4.2 SCRIPT TO SYNC WITH CONTENT

This section describes scripts that operate in sync with the video/slide version system. Both scripts basically use standard JavaScript and microcontroller libraries, but the video version has a description method for "elapsed time on video" and the slide version has a description method for "page transitions on slides".

The video version uses a JavaScript framework called "srt.js" [5]. This is an extension of the srt format file, which is a format for describing subtitle information for videos, and can execute specific code in response to the elapsed time on the video. For example, in the video version, the elapsed time on the video (e.g., 10 seconds to 15 seconds) is described as shown in the upper part of the Figure 3, and then the code to be executed at that time is described. Here, from the serial number immediately before the elapsed time up to the blank line is recognized as one section.

In the slide version, an array of functions is prepared as shown in the lower part of Figure 3, the page number of the slide is specified as the index, and the code is written in the function. The functions are executed at the timing of the transition to each page (displayed on the screen).



```
// ボタンAを押した時に1つずつ順番にディスプレイに表示するための配列
const statements = [
    obniz.display.clear(),
    obniz.display.line(30, 30, 100, 30),
    obniz.display.rect(20, 20, 20, 20),
    obniz.display.circle(100, 30, 20),
    obniz.display.line(60, 50, 100, 30),
    obniz.display.rect(50, 40, 20, 20, true),
    obniz.display.line(50, 10, 100, 30),
    obniz.display.circle(50, 10, 10, true),
    obniz.display.setColor('#FF0000'),
    obniz.display.rect(50, 40, 20, 20, true),
    obniz.display.setColor('blue'),
    obniz.display.circle(100, 30, 20, true),
    obniz.display.clear(),
    obniz.display.font(null, 20),
    obniz.display.pos(90, 50),
    obniz.display.print("おわり"),
    obniz.display.clear(),
    obniz.display.setColor("black"),
]

obniz.onconnect = async() => {
    obniz.display.clear();
    let index = 0;
```

Fig. 4. A screenshot of the video version viewing application

When the video is played by seeking and skipping sections or skipping slides, there were problems with unexpected behavior (e.g., the process of outputting sensor logs did not stop even in unrelated situations). Therefore, we made it possible to describe the process when the user leaves each section for both the video and slide versions. Specifically, we implemented an event handler function (onSectionLeave) that monitors the current elapsed time/page number during script execution and is called when the user leaves the section. The above problems were solved by writing the stop processing in the function, and stable seeking of movies and slides became possible. These scripts are associated with the contents and stored in a database on Firebase.

4.3 VIDEO VERSION SYSTEMS

The video version system consists of a web application, a video sharing service (e.g., YouTube), and a microcontroller. Also, we developed two web applications: a viewing application and a creation support application. The details are described below.

4.3.1 VIEWING APPLICATION. A screenshot of the viewing application is shown in Figure4. It consists of four major elements: (1) the video display part embedded from YouTube, (2) the source code display part, (3) the subtitle display part, and (4) the output console.

The source code display part was developed using highlight.js[4], a library for highlighting code displayed on a Web page. The code display part has two tabs: "Real-time" and "Entire". Real-time" tab displays only the code that corresponds to the time axis of the video (current device behavior). "Entire" tab displays all the codes that perform a series of processing.

The subtitle display part shows the text manually written in the script(e.g., "Look the device at hand"). We believe that this will reduce the time and effort required for video editing. The creator can write subtitles along

[4]https://highlightjs.org/

with code according to the syntax of srt.js. The output console outputs sensor data, etc. corresponding to the actions of the device at hand (e.g., buttons, accelerometers) in real time.

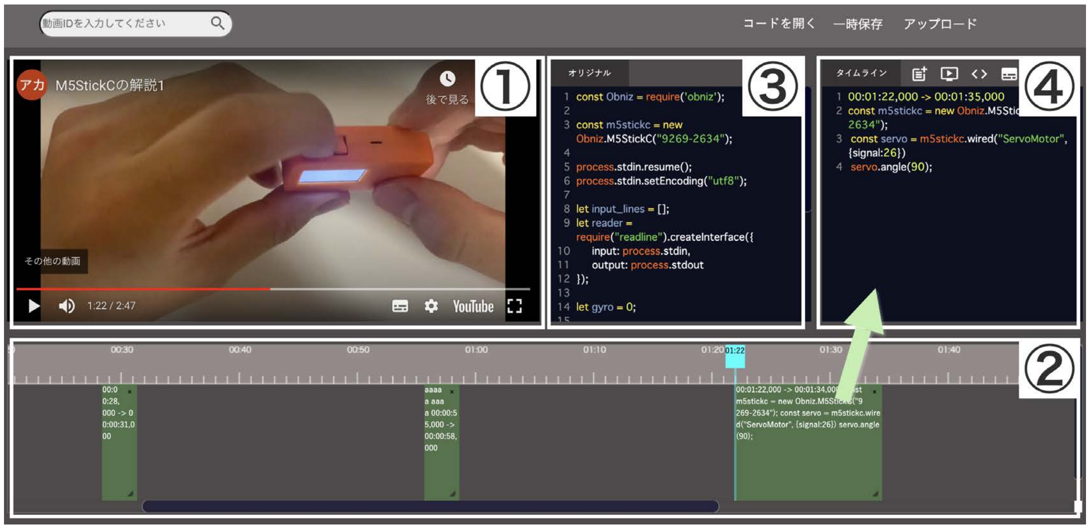
Also, the four elements (video/source code/subtitle/console) can be highlighted to draw attention to them.

4.3.2 CREATION SUPPORT APPLICATION. In the video version system, it is necessary to enter the elapsed time of the video in the script. To do this manually, the user must open the video and the editor side by side, search for the device operation scene on the video, and enter the elapsed time in the editor, which is time-consuming and laborious.

For this reason, we developed a creation support application for the video version system(Fig 5). The system consists of four main elements: (1) video display part, (2) timeline, (3) code editor (original), and (4) code editor (timeline).

The timeline corresponds to the video's time axis, and by double-clicking anywhere on the timeline, a code block can be created that automatically inserts the elapsed time of the video the start time is the current playback time of the video, and the end time is 3 seconds after the current playback time. The length of the created code block can be changed by dragging, and the effective range of the block can be changed later. Furthermore, by double-clicking on a code block, it can be opened and edited in the code editor (timeline) in the upper right corner. The code editor (original) is designed to allow efficient editing of srt.js format code by loading previously created code and using copy and paste, etc.

Finally, by clicking the "Upload" button, the srt.js format file is uploaded to the database on Firebase. After uploading, the URL of the viewing application is provided, allowing the user to immediately check its operation.



```
const Obniz = require('obniz');

const m5stickc = new
Obniz.M5StickC("9269-2634");

process.stdin.resume();
process.stdin.setEncoding("utf8");

let input_lines = [];
let reader =
require("readline").createInterface({
    input: process.stdin,
    output: process.stdout
});

let gyro = 0;
```

```
00:01:22,000 -> 00:01:35,000
const m5stickc = new Obniz.M5Stic
2634");
 const servo = m5stickc.wired("ServoMotor",
{signal:26})
 servo.angle(90);
```

```
00:0
0:28,
000 -> 0
0:00:31,0
00
```

```
aaaa
a aaa
a 00:00:5
5,000 ->
00:00:58,
000
```

```
00:01:22,000 -> 00:01:34,00     st
m5stickc = new Obniz.M5St       ("9
269-2634"); const servo = m5stickc.wire
d("ServoMotor", {signal:26}) servo.angle
(90);
```

Fig. 5. A screenshot of video version creation support application

4.4 SLIDE VERSION SYSTEMS

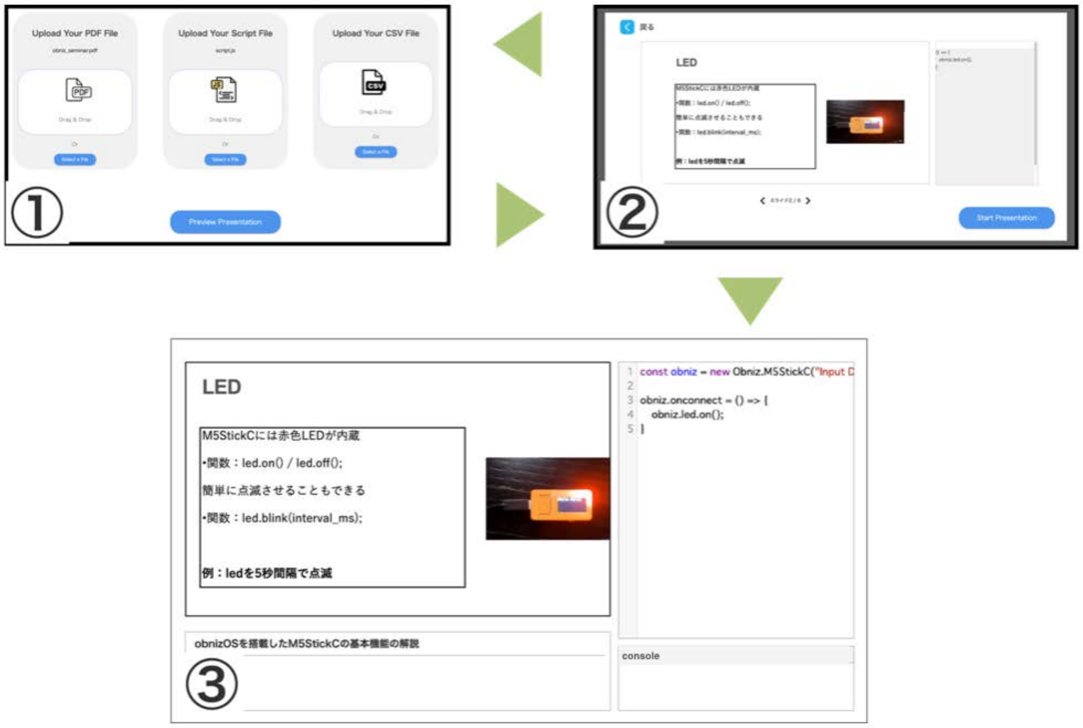
The slide version of the system is structured in much the same way as the video version, and two applications were developed: a viewing application and a creation support application. The creation support application for the slide version(Figure 7) is similar in many aspects to the one for the video version, so we omit the explanation here.

4.4.1 VIWING APPLICATION. Figure 6 shows a screenshot of the system and the flow of use. First, the creator (mainly the instructor) loads the slides (PDF), scripts (JavaScript), and a list of instructor/student device IDs (CSV) into the Web app on the initial screen. The slide version system connects the instructor's PC to all learners' obniz based microcontrollers, so the CSV files can be loaded together.

After the three files have been imported, the user is taken to the preview screen. In the preview screen, you can check the code executed at the transition to each slide. If there is a problem, you can replace only a specific file by clicking the "Back" button.

From the preview screen, click the "Start Presentation" button to display the slide. and start the linkage with the device. When a specific slide is displayed (page transition), the function corresponding to the page number is executed, and all obniz based microcontrollers corresponding to the device ID loaded in advance work together.To reduce the burden on the instructor, the iterative process for multiple microcontrollers is hidden in the code and automatically completed.

Uploaded PDF/JavaScript/CSV files are managed in a database on Firebase and can be recalled later using the PDF file name and time stamp as keys.



LED

M5StickCには赤色LEDが内蔵
•関数：led.on() / led.off();
簡単に点滅させることもできる
•関数：led.blink(interval_ms);
例：ledを5秒間隔で点滅

obnizOSを搭載したM5StickCの基本機能の解説

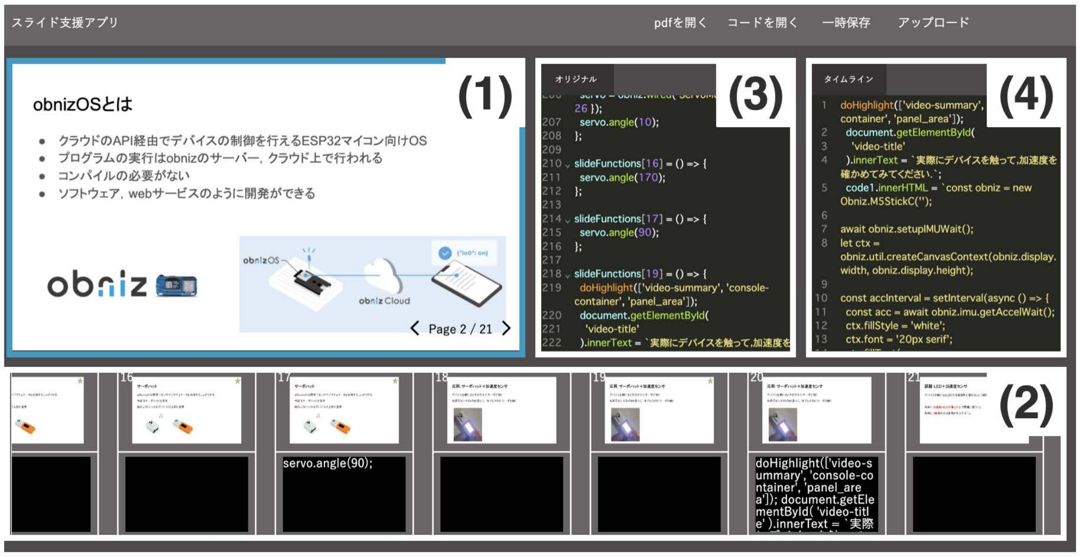
```
const obniz = new Obniz.M5StickC("Input D

obniz.onconnect = () => {
  obniz.led.on();
}
```

Fig. 6. A screenshot of slide version viewing application (1. Initial screen, 2. Preview screen, 3. Presentation screen)

5 SUMMARY

This research proposes and develops IoTeach, a system that supports IoT programming learning by synchronizing sequential content such as videos and slides with a scripting language and linking it to real-world IoT devices and



obnizOSとは

- クラウドのAPI経由でデバイスの制御を行えるESP32マイコン向けOS
- プログラムの実行はobnizのサーバー，クラウド上で行われる
- コンパイルの必要がない
- ソフトウェア，webサービスのように開発ができる

```
  26 });
  servo.angle(10);
};

slideFunctions[16] = () => {
  servo.angle(170);
};

slideFunctions[17] = () => {
  servo.angle(90);
};

slideFunctions[19] = () => {
  doHighlight(['video-summary', 'console-container', 'panel_area']);
  document.getElementById(
    'video-title'
  ).innerText = `実際にデバイスを触って,加速度を
```

```
doHighlight(['video-summary', 'console-container', 'panel_area']);
 document.getElementById(
  'video-title'
 ).innerText = `実際にデバイスを触って,加速度を確かめてみてください.`;
 code1.innerHTML = `const obniz = new Obniz.M5StickC('');

await obniz.setupIMUWait();
let ctx =
obniz.util.createCanvasContext(obniz.display.width, obniz.display.height);

const accInterval = setInterval(async () => {
 const acc = await obniz.imu.getAccelWait();
 ctx.fillStyle = 'white';
 ctx.font = '20px serif';
```

```
servo.angle(90);
```

```
doHighlight(['video-summary', 'console-container', 'panel_area']); document.getElementById( 'video-title' ).innerText = `実際
```

Fig. 7. A screenshot of slide version creation support application

editable source code. We expect that IoTeach will increase the viewer's understanding of devices and motivation. This paper introduced the system concept of IoTeach and the implementation of four applications.

REFERENCES

[1] Arduino. 2023. *Arduino Editor - Web Editor.* Retrieved January 4, 2023 from https://create.arduino.cc/editor
[2] Udemy Inc. 2023. *Udemy.* Retrieved January 4, 2023 from https://www.udemy.com
[3] Kato Jun, Ogata Masa, Inoue Takahiro, and Goto Masataka. 2018. Songle Sync: A Large-Scale Web-based Platform for Controlling Various Devices in Synchronization with Music. In *Proceedings of the 26th ACM International Conference on Multimedia.* 1697–1705.
[4] Kato Jun, Nakano Tomoyasu, and Goto Masataka. 2015. TextAlive: Integrated Design Environment for Kinetic Typography. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.* 3403–3412.
[5] Kazutaka Kurihara and Mika Hashimoto. 2016. srt.js:Framework for embedding IoT-oriented extension programs into video content. In *Proceedings of WISS2016, Japan Society for Software Science and Technology.*
[6] Yeh Tom, Chang Tsung Hsiang, and Miller Robert C. 2009. Sikuli: Using GUI screenshots for search and automation. In *UIST 2009 - Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology.* 183–192.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009