Software to Support Layout and Data Collection for Machine-learning-based Real-world Sensors

Ayane Saito[1], Wataru Kawai[2] and Yuta Sugiura[1,3]

[1] Keio University, Japan
[2] The University of Tokyo, Japan
[3] JST PRESTO, Japan

Abstract. There have been many studies of gesture recognition and posture estimation by combining real-world sensor and machine learning. In such situations, it is important to consider the sensor layout because the measurement result varies depending on the layout and the number of sensors as well as the motion to be measured. However, it takes time and effort to prototype devices multiple times in order to find a sensor layout that has high identification accuracy. Also, although it is necessary to acquire learning data for recognizing gestures, it takes time to get the data when the user changes the sensor layout. In this study, we developed software that can arrange real-world sensors. In this time, the software can handle distance-measuring sensors as real-world sensors. The user places these sensors freely in the software. The software measures the distance between the sensors and a mesh created from measurements of real-world deformation recorded by a Kinect. The classifier is generated using the time-series of distance data recorded by the software. In addition, we created a physical device that had the same sensor layout as the one designed with the software. We experimentally confirmed that the software could recognize the gestures on the physical device by using the generated classifier.

Keywords: Sensor Layout, Machine Learning, Distance-measuring Sensor.

1 Introduction

Physical changes in the real world can be measured by deploying the real-world sensors in daily life. For such measurements, the number and layout of the sensors have to be determined, taking into account the real-world changes we want to capture, the associated costs, device sizes, and so on. However, it is difficult to design an optimal sensor layout in consideration of these various limitations. In addition, gesture recognition and posture estimation can be performed by combining real-world sensors and machine learning. For example, Touché [1] identifies touch gestures by machine learning of the frequency response when the sensor is touched. In this way, complicated actions can be identified by learning in advance, but it is necessary to generate a classifier using unique sensor data according to the state to be identified. However, the number, position, and angle of sensors that can acquire such unique sensor data are often determined by trial and error. Moreover, it takes time and effort to accumulate learning data in the real world every time the number and layout of the sensors are changed.

In this study, we developed software that helps to arrange real-world sensors and support learning data collections. At present, the software handles distance-measuring sensors as real-world sensors. The user places models of the sensors freely in the software, and the software measures the distance between the placed sensors and real world objects recorded by an RGB-D camera (a Kinect). The obtained distance data is recorded as time series data to identify real-world changes such as gestures. Furthermore, in this study, to confirm that the software can accumulate learning data, we used a classifier generated from gesture data acquired by the software to identify gesture data acquired by sensors placed in the real world.

2 Related Works

2.1 Simulation of Real-World System on a Computer

There have been many studies to simulate by restoring the shapes and conditions of the real world on a computer. Ino et al. produced a CG model of a human hand on the computer by using motion capture data and software called Dhaibaworks which can generate digital models of the human body [2]. They estimated a grasping hand pose by performing learning with a convolutional neural network on a photographed image of a CG model and three-dimensional position data such as hand joints. Kanaya et al. developed a human-mobility sensing system simulator, called Humans, that uses map data, human agent information, sensor information, and network information [3]. They simulated the walking behavior of people in a city while simulating changes in the kind or arrangement of sensors on the map. Yuan et al. learned virtual egocentric video and posture of a humanoid model walking in a virtual world [4]. They estimated walking postures of people in the real world by combining data in images of the real world shot by a camera mounted on the head of a pedestrian and learning data in the virtual world. Since these studies are computer simulations, there is an advantage that it is not necessary to accumulate learning data in the real world and it does not take much money and time.

In this study, the target of simulation was a real-world sensor. We developed software that helps user to avoid the trial and error involved in setting up physical sensors and that also accumulates learning data.

2.2 Measuring Motion with Photo Sensors

We developed software that can handle distance-measuring sensors, which are a kind of photo sensor. There have been many studies on measuring and identifying human body movements by using photo sensors. For instance, AffectiveWear is a device for recognizing facial expressions that measures the distance between the skin and the frame of a pair of glasses by using an array of photo-reflective sensors on the frame [5]. EarTouch recognizes gesture using the ear as an input surface by using photo-reflective sensors attached to earphones to measure skin deformation inside the ear [6]. iRing measures the distance between the skin of the finger and photo-reflective sensors attached to a ring-shaped device to recognize bending of a finger [7]. Miyata et al. estimated the grasping posture of a hand by attaching a band-type device with embedded distance-measuring sensors to the grasped object [8].

In this study, we detected and identified hand gestures made in front of distance-measuring sensors.

3 System Implementation

3.1 Overview

We developed software for arranging the placement of distance-measuring sensors which are often used as real-world sensors. The flow of our system incorporating this software, shown in Fig. 1, is divided into a phase of accumulating learning data by the software and a phase of gesture recognition using data acquired by distance-measuring sensors in the real world. In the learning phase, the user makes gestures in front of a Kinect and acquires time-series distance data with sensors placed on the software. The acquired data is converted into image data from which the histograms of oriented gradients (HOG) feature is then extracted. The classifier is generated by learning with a support vector machine (SVM) that takes the obtained HOG feature as input. In the gesture recognition phase, the same kind of gesture as one performed in front of Kinect is performed in front of distance-measuring sensors arranged in the real world in the same layout as in the software, and sensor data is acquired. The acquired sensor data is converted in order of distance value and image, and the HOG feature is extracted. The

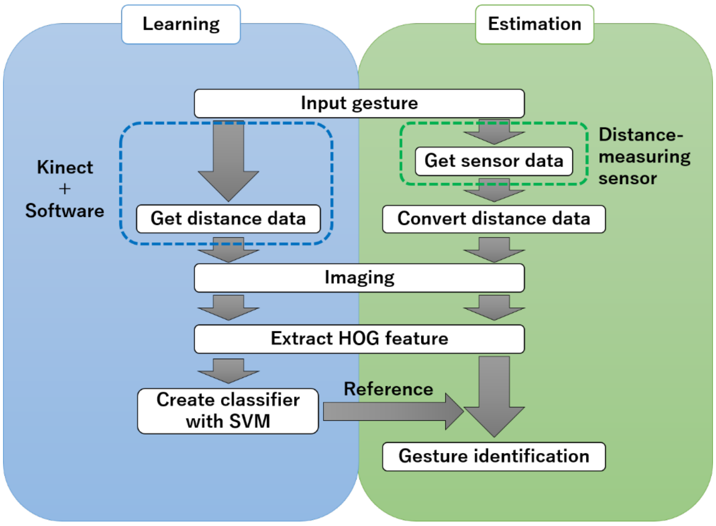

Fig. 1. Flow of our system.

obtained HOG feature performs gesture recognition by referring to the classifier generated from the data acquired in the software.

3.2 Configuration of Software

We used Unity to create the software. The real-world changing data was acquired as 3D depth information with a Kinect v2, and the shape of the real world was restored in the virtual world with Unity. First, we converted the Kinect depth coordinate system into a real-world space coordinate system (Unity's coordinate system). The 3D depth information was converted into a 3D mesh on Unity. In order to construct a high-resolution mesh on Unity, we used depth information from the center of a 1/4 ×1/4 range of the image taken with Kinect. The simulated distance-measuring sensors were placed in the virtual world holding the mesh and the distances between the sensors and the mesh were measured, as shown by the black cylindrical object in Fig. 2.

The distance value was calculated by using a binary search algorithm using a Raycast function that can determine collision with the collider (we set up a 3D mesh is a collider). The binary search was continued down to the resolution of Kinect's depth (1 mm), and when there was no 3D mesh in the measurable range of the sensor, we set the distance value to the maximum measurable distance of the sensor. The acquired distance values were displayed as graphs on Unity's game screen (Fig. 2) and saved as a CSV file.

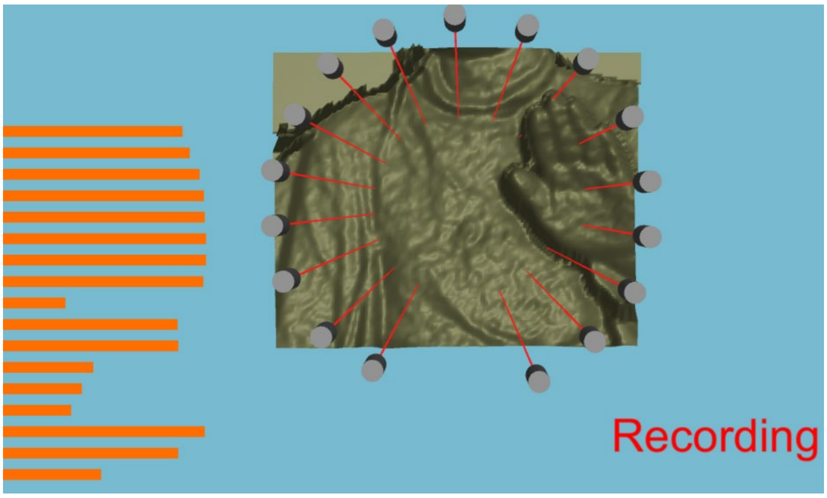

Fig. 2. Overview of developed software.

3.3 Real-World Sensor Data Acquisition

The distance-measuring sensors in the real world were GP2Y0E02A developed by Sharp Corporation. Each sensor consisted of an infrared LED and a position detector

(PSD) and measured the distance to the target object based on an incident position of the reflective infrared light. They could detect distances from 4 cm to 50 cm. Since the relationship between the sensor value and the distance value is linear, the distance value was calculated from the sensor value (1) and recorded as time-series data.

$$Distance = 4.0 + (2.2 - SensorValue * 5 / 1023) / 0.036 \tag{1}$$

The sensors were connected to a microcontroller (Arduino Pro Mini, 3.3 V). The sensor data were sent to a PC. Since the time intervals for acquiring the distance data differed between the software and the distance-measuring sensor, the data of the distance-measuring sensor was extracted according to the time interval of the software, which was the longer time interval, and saved as a CSV file.

3.4 Obtaining Learning Data and Making an Identification

We converted the CSV files acquired by the software and distance-measuring sensors into 2D-grayscale image data.

The start timing of imaging was taken to be when the distance data of any two sensors became smaller than 30 cm. We applied a low-pass filter (RC filter) to the distance data, as follows:

$$y[i]=0.4x[i] + 0.6y[i - 1] \tag{2}$$

(x: present data, y: time series data frame)

After obtaining the filtered distance data, we converted it into 2D-grayscale image data by means of normalization. In the converted image, each distance value was allocated to the vertical axis, and the time-series data were allocated to the horizontal axis.

The SVM classifier was generated by learning using the HOG features of the 2D-grayscale image generated from the software data. A gesture performed in front of the distance-measuring sensors in the real world was identified using the HOG features of a 2D-grayscale image generated from the data of the distance-measuring sensors and the classifier generated with the data of the software.

4 Evaluation

4.1 Overview

We conducted experiments to determine whether the learning data accumulated by our software can be used to identify data measured with distance-measuring sensors in the real world. Seventeen distance-measuring sensors were attached around the perimeter of a Hue Go light developed by Philips. The distance between adjacent sensors was about 2.5 cm (Fig. 3 (left)). The sensors in the software were arranged in the same layout as those in the real world (Fig. 3 (right)). Next, we performed several gestures in front of the Kinect and acquired distance data. After that, the same gestures were

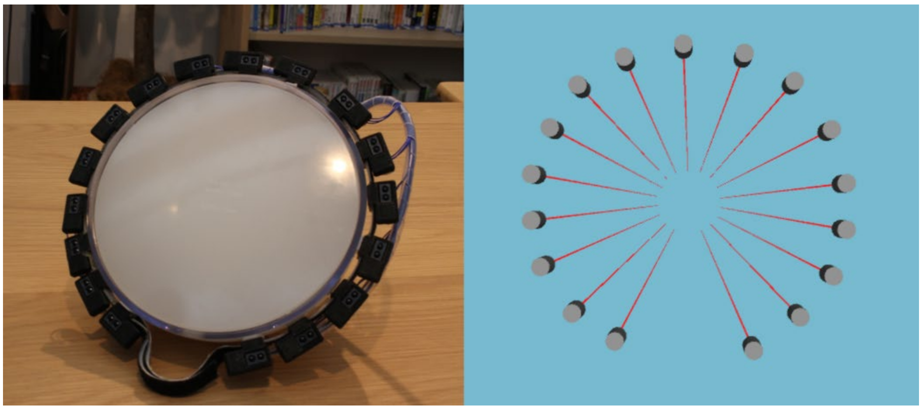

Fig. 3. Sensor layout (Left: in the real world, Right: in the software).

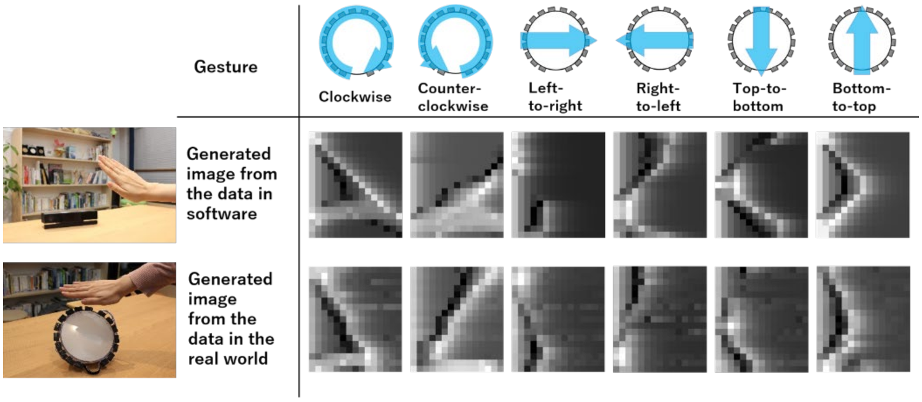
Gesture
Clockwise
Counter-clockwise
Left-to-right
Right-to-left
Top-to-bottom
Bottom-to-top
Generated image from the data in software
Generated image from the data in the real world

Fig. 4. Types of identified gesture and examples of generated images.

performed in front of distance-measuring sensors placed in the real world, and the distance data were acquired from the sensor data. Fig. 4 shows the types of identified gesture and examples of generated images. Each gesture was performed ten times in front of the Kinect and ten times in front of the distance-measuring sensors; 15 frames were acquired per gesture and imaged. Leave-one-out cross-validation was performed on the dataset acquired with the software to calculate the identification rate, and a classifier was generated. Then, we identified each gesture acquired by the distance-measuring sensors using the classifier generated with the data acquired on the software.

4.2 Results and Discussion

Fig. 5 (left) shows the identification results when learning and identifying were performed using data acquired with the software; the average accuracy of the identification

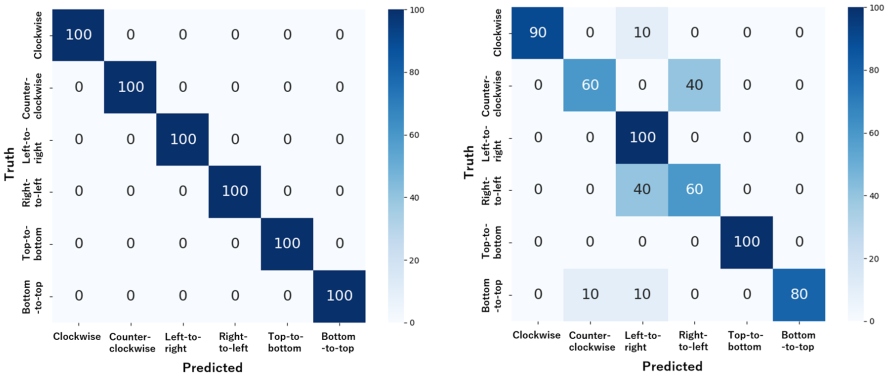

Fig. 5. Identification results (Left: using data acquired with the software, Right: using data acquired by the distance-measuring sensors).

was 100.0%. Fig. 5 (right) shows the identification results for each gesture data acquired with the distance-measuring sensors using the classifier generated with the software data; the average accuracy of the identification was 81.7%. This identification rate was high enough to show that it is possible to identify the data measured with the sensors in the real world with reference to the learning data from the software, instead of learning data from the real world. The misidentification of the right-to-left gesture as the left-to-right gesture in Fig. 5 (right) is considered to have been caused by the fact that the gesture has been completed before all frames had elapsed (Fig. 4). Therefore, the number of frames that can be acquired for one gesture can be increased by increasing the processing speed of the software, and the identification rate will also increase.

5 Limitations and Future Work

The simulated sensors placed in the software are not affected by ambient light, but the actual distance-measuring sensors are affected by it; the ambient light adds noise to the sensor data. The identification was performed on relatively slow gestures because the processing speed of the software was slow in the experiment and the distance data were acquired at intervals of about 0.30 seconds. Since the distance-measuring sensor can obtain data at a higher speed, it is considered possible to identify faster gestures by increasing the processing speed of the software. In future work, we will develop a system that presents a sensor layout that increases the identification rate when the user specifies the number of sensors.

6 Conclusion

We developed software for designing the layout of real-world sensors and verified the identifiability of the data acquired with the sensors placed in the real world by using a classifier generated from the data acquired on the software. The software acquires distance data by measuring the distance in the virtual world based on depth information from a Kinect and sensors placed in the virtual world. Data acquired with sensors in the real world can be identified using a classifier generated from data acquired on the software. We found that the average accuracy of the identification was 81.7%. In future work, we will increase the processing speed of the software and develop a system that presents a sensor layout having a high identification rate.

Acknowledgments

This work was supported by JST AIP-PRISM JPMJCR18Y2 and JST PRESTO JPMJPR17J4.

References

1. Sato, M., Poupyrev, I., Harrison, C.: Touché: Enhancing touch interaction on humans, screens, liquids, and everyday objects. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12), pp. 483-492. ACM, New York, NY, USA (2012).
2. Ino, K., Ienaga, N., Sugiura, Y., Saito, H., Miyata, N., Tada, M.: Grasping hand pose estimation from rgb images using digital human model by convolutional neural network. In: 9th International Conference and Exhibition on 3D Body Scanning and Processing Technologies, pp. 154-160. HOMETRICA CONSULTING, Lugano, Switzerland (2018).
3. Kanaya, T., Hiromori, A., Yamaguchi, H., Higashino, T.: Humans: A human mobility sensing simulator. In: 2012 5th International Conference on New Technologies, Mobility and Security (NTMS), pp. 1-4. IEEE, Istanbul, Turkey (2012).
4. Yuan, Y., Kitani, K.: 3d ego-pose estimation via imitation learning. In: Technical Paper at European Conference on Computer Vision (ECCV), pp. 763-778. LNCS, Munich, Germany (2018).
5. Masai, K., Sugiura, Y., Ogata, M., Kunze, K., Inami, M., Sugimoto, M.: Facial Expression Recognition in Daily Life by Embedded Photo Reflective Sensors on Smart Eyewear. In: Proceedings of the 21st International Conference on Intelligent User Interfaces (IUI '16), pp. 317-326. ACM, New York, NY, USA (2016).
6. Kikuchi, T., Sugiura, Y., Masai, K., Sugimoto, M., Thomas, B.: Eartouch: Turning the ear into an input surface. In: Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '17), pp. 27:1-27:6. ACM, New York, NY, USA (2017).
7. Ogata, M., Sugiura, Y., Osawa, H., Imai, M.: iring: Intelligent ring using infrared reflection. In: Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12), pp. 131-136. ACM, New York, NY, USA (2012).
8. Miyata, N., Honoki, T., Maeda, Y., Endo, Y., Tada, M., Sugiura, Y.: Wrap sense: Grasp capture by a band sensor. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST ’16), pp. 87-89. ACM, New York, NY, USA (2016).