International Journal of Service and Knowledge Management
International Institute of Applied Informatics
2021, Vol. 5, No. 1, 36 – 50

A Context-aware Image Recognition System with Self-localization in Augmented Reality

Ryosuke Suzuki *, Tadachika Ozono *, Toramatsu Shintani *

Abstract

The diffusion of augmented-reality (AR) frameworks has facilitated the implementation of support systems for several real-world tasks. This paper introduces a system that supports Mahjong scoring for beginners. Mahjong is a globally popular strategic board game. Playing Mahjong improves cognitive functions and promotes social interactions. However, it is complex for beginners to accumulate a score according to the combinations of Mahjong tiles. We aim to develop an offline system to tally the score by visually recognizing the Mahjong tiles, which have classes and attributes based on their positional context. This system, therefore, requires a context-aware image recognition. The system recognizes their contextual attributes via self-localization and detects each tile using OpenCV and a convolutional neural network to classify them. The accuracy of detecting tiles and recognizing attributes was good enough to provide an acceptable support system. Our experimental results demonstrate that the system is accurate enough to detect tiles and to recognize attributes. We concluded that the system provides adequate support for the novices.

Keywords: Context-aware Image Recognition, Augmented Reality, Self-localization, Classification, Object Detection, Mahjong

1 Introduction

Augmented-reality (AR) technologies have grown and diffused into many new frameworks (e.g., Apple ARKit, Google ARCore), and this has facilitated the implementation of AR systems that can support real-world tasks. For example, the superimposition of virtual objects can be used to support human interactions in the real world. Imposing visible information on visible objects improves learning efficiency and reduces the cognitive recognition burden [1]. For such an application, a strong context-aware image recognition capability is necessary for choosing the right information to display. In this research, context-aware recognition is used for attributed object detection. Object detection is a method of detecting and identifying multiple classes of objects in images and videos. Attributed object detection combines regular object detection with attribute identification. For our purposes, the attributes represent contextual information about Mahjong tiles, because the semantics applied them for scoring depend upon their context, which can refer to locality. Torralba et al.

* Department of Computer Science, Graduate School of Engineering, Nagoya Institute of Technology

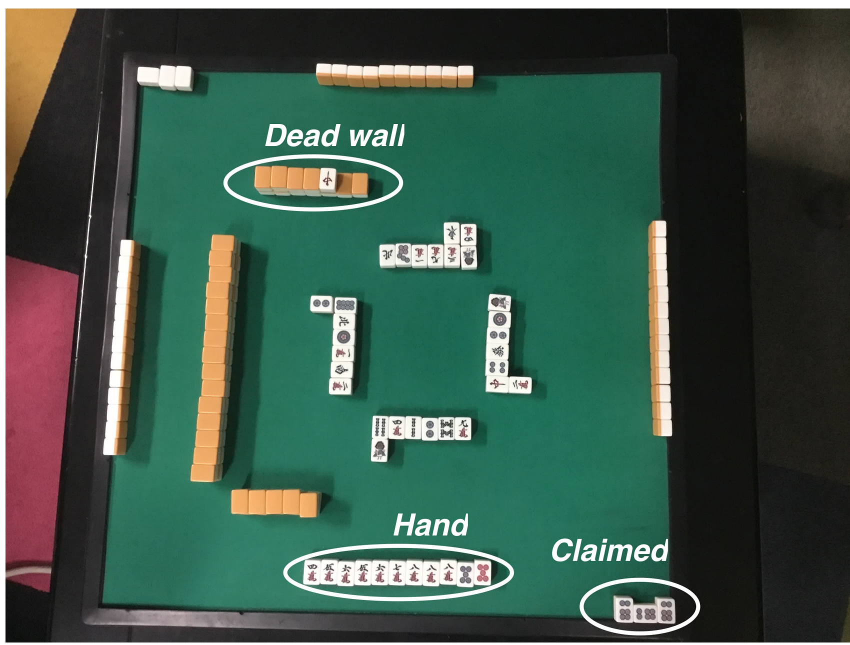

Figure 1: Example of arranged Mahjong tiles

[2] indicated that place recognitions can be useful for object recognitions. With Mahjong tiles, their positions in relation to other objects in the gaming area is pertinent information required for scoring. Thus, we apply AR technology that ascertains tile positions. A key feature of AR technology is its ability to track feature points. The more capably the system tracks feature points, the more accurately it can apply the semantics. In this paper, an AR world denotes a virtual 3D space based on the real world. The AR world has the same scale as the real world and can superimpose information on the real-world display. We develop a context-aware image recognition method based on AR technologies to implement a Mahjong support system. The system automatically calculates a score based on the combination of Mahjong tiles, which is especially useful to beginners. We implement the system on iOS devices, because they provide a powerful AR framework (i.e., ARKit). This framework can track feature points very well.

Author et al. [3] proposed a Mahjong support system and described a process for recognizing Mahjong tiles, comprising categories of ***Hand***, ***Claimed***, and ***Dead wall***. This paper also describes an error-handling process that facilitates tile recognition.

The remainder of this paper is organized as follows. In Section 2, we introduce Mahjong game and how we propose to support Mahjong players. In Section 3, we describe extant works and the self-localization method. We describe the details of the system architecture in Section 4. In Section 5, we evaluate the effectiveness of the system via experiments. We present discussions and future development in Section 6. Finally, we conclude this paper in Section 7.

2 Mahjong Support System

This section provides an overview of Mahjong and how our system supports it. Mahjong is a globally popular strategic board game. Cheng et al. [4] indicated that playing Mahjong

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

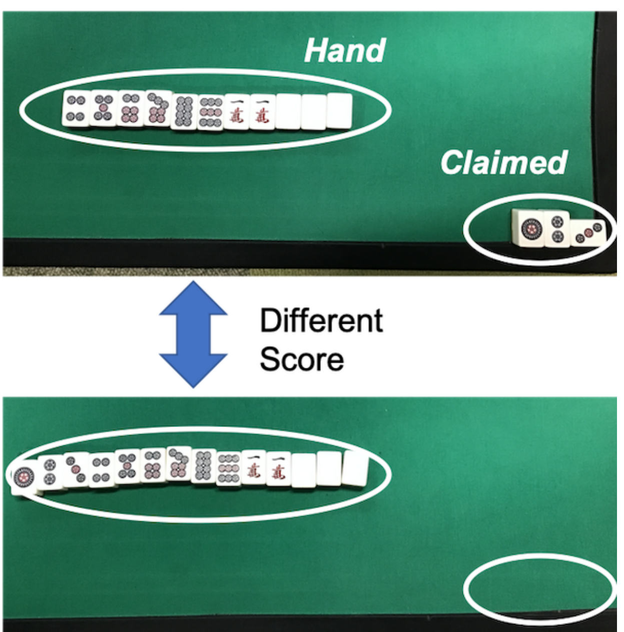

Figure 2: Same tiles, different scores.

improves cognitive functions and promotes social interactions. However, it is complex for beginners. It is a four-player game that uses 136 tiles. The player having the highest score at the end of the game wins. Players accumulate a score according to the combinations of 14 tiles called ***Yaku***. A ***Yaku*** consists of four ***Melds*** and a ***Head***. A ***Meld*** is either a ***Chow***, a ***Pung***, or a ***Kong***. A ***Chow*** consists of three consecutively numbered tiles. Each a ***Head***, a ***Pung***, and a ***Kong*** consists of two, three, and four tiles same of a class. Scoring depends on the attributes of the tiles, and each has a class and an attribute. 136 tiles are arranged on the Mahjong table according to the setup rules. Chinese characters or figures are used to identify class, and they are printed on their faces. Four tiles exist for each class, and the positions of the tiles on the Mahjong table are used to identify attributes, reflecting the roles of the tiles.

In preparation, each player makes a ***Wall*** comprising 34 tiles turned upside down and picks 13 tiles from it. Players decide an order in which to roll the dice. Players pick a tile from the ***Wall*** and add it to their ***Hand*** in the order of the roll. They must discard a tile if their ***Hand*** does not reach a ***Yaku***. When a player reaches a ***Yaku***, he or she is supposed to calculate a score. Players have two ways to pick a tile, and this influences the assignment of attributes. The first type involves self-picking, where a tile is chosen from the ***Wall***. The second type involves claiming a tile discarded from another player. The tiles picked this way are ***Claimed*** and placed on the table separately from the ***Hand***. The other main factor that influences scoring is the ***Dora***. The number of ***Dora*** creates a bonus score. The ***Dead wall*** contains tiles indicating ***Dora***. Figure 1 illustrates a Mahjong table scene during game play. The circled areas denote the main influencers of scoring.

It is important to distinguish ***Hand***, ***Claimed***, and ***Dead wall***. Figure 2 shows two examples of ***Yaku*** containing the same tiles, but their scores are different because of the variance in ***Hand*** and ***Claimed***. Recognizing these differences is essential for obtaining a correct score. Thus, the calculation process has many factors, including the number of ***Dora***, the presence or absence of ***Claimed***, and the kind of ***Yaku***. This process is complex.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

If a beginner can recognize only a ***Hand***, they face troubles. For example, 14 tiles cannot achieve a ***Yaku*** if users confuse ***Hand*** and ***Claimed***. In another case, users cannot consider ***Dora***. Therefore, a Mahjong support system for beginners is needed. Such a system will help the player calculate a score. It is notably difficult to realize such a system for offline Mahjong. The system must recognize Mahjong tiles visually or electronically, requiring context-aware recognition. Joys Co., Ltd. developed a Mahjong table that automatically calculates scores[1]. The table uses sensors that detects tiles using embedded integrated circuits. This product does, in fact, calculate the correct score. However, it is inconvenient for users to carry around. Therefore, we are compelled to implement a tool that relies on a ubiquitous and easily portable system, such as an iOS device.

Next, we describe how the system recognizes tile attributes, including the locations at which the Mahjong tiles are placed. Players place tiles at known positions according to the rules. The system, therefore, knows in advance the tiles' rough positions. The system captures an overall Mahjong table image, as shown in Figure 1. Unfortunately, the picture contains low-resolution tile features. Thus, the system cannot distinguish tile classes. The system, therefore, is required to capture images closer to the tiles. Doing so presents another problem in that it cannot see the entire table. Users could be required to input attributes of each tile to the system. However, this is far from automatic. Plus, we intend to implement the system using AR glass. Ideally, users will calculate a score by simply scanning the table while wearing their AR glass. Then, the system can adequately track feature points via start and captured positions. The system must, therefore, see the entire table while compiling close-up images of the tiles.

3 Related Work

This section describes previous and related works and clarifies the novelty of our research. We implement a system based on the prototype from [5], which can recognize a ***Hand***. However, users are required to input a lot of information by hand. Matsui et al. [6] implemented a Mahjong score calculating system to recognize ***Hand*** and ***Claimed***. However, users are required to rearrange Mahjong tiles for recognizing, and the system does not support for ***Dead wall***. We then explain the difficulties of detecting ***Claimed*** and ***Dead wall***. Notably, the self-localization capability of AR technologies can solve the problem of capturing the entire table while focusing on individual tiles. The system can, therefore, distinguish ***Hand***, ***Claimed***, and ***Dead wall***.

3.1 Self-localization

This section describes methods of self-localization and the role ARKit plays in this activity. The ARKit self-localization technique is called visual inertial odometry (VIO), which uses correct positioning to construct an accurate AR world corresponding to the real one. Visual odometry (VO) [7] is the predecessor to VIO. The difference is found in the physical sensors of acceleration, magnetism, etc. Zhou, et al. [8] identified three tracking techniques: sensor-based, vision-based, and hybrid. Sensor-based tracking uses physical sensors. Vision-based tracks feature points from camera data. Hybrid tracking uses sensor- and vision-based tracking together and is more sensitive than the other techniques. VO uses vision-based tracking, and VIO uses hybrid. We can estimate attributes with higher accuracy when using

[1] http://Mahjong-joys.jp/

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

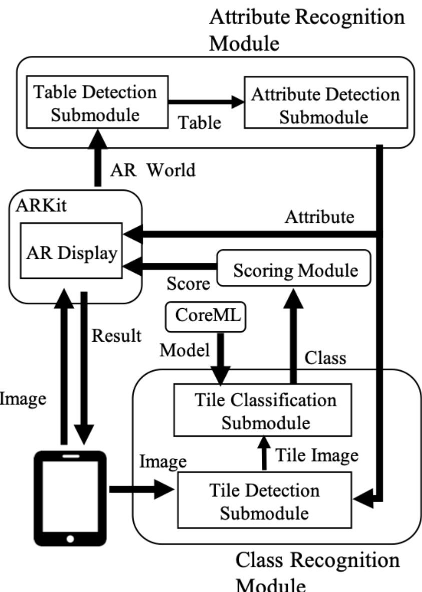

Figure 3: System architecture

VIO. A similar technique to VO is simultaneous localization and mapping (SLAM). Yousif, et al. [9] described the differences between VO and SLAM, wherein VO focuses on local consistency to estimate the path of a camera and to perform local optimization. SLAM, on the other hand, estimates the path of a camera and creates a map. SLAM considers the integrity of the overall path and performs loop-closure using local distortion. VIO does not integrate an overall path. As mentioned VIO was developed from VO. ARKit adopts VIO. When supporting a real Mahjong game, the system does not perceive long distances as does robotics localization. Thus, the system does not consider the integrity of the overall path, as does SLAM.

4 Mahjong Support System based on AR

This section describes the detailed system described in the previous sections. Figure 3 shows the system architecture comprising three modules: attribute recognition, class recognition, and scoring. Regarding system flow, the device first captures an image that includes the tiles. The attribute recognition module estimates their position in relation to the image. The class recognition module estimates the classes of each tile. The scoring module then returns the score of a ***Yaku*** consisting of the classes and attributes of the tiles. Finally, ARKit displays the score onto the Mahjong table. Note that the system cannot obtain all the information for calculating the score from the image. For example, the system cannot recognize the role of each player. One of the four players is the leader (or parent), and the others are non-leaders (children). They change their roles during game play. Players have this information, but it is invisible to the camera. However, the score depends on this information. Therefore, users input their roles to the system with other information about the leader, the ***Wind*** of the round, and the number of ***Bone***. In this paper, we omit those detailed rules.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

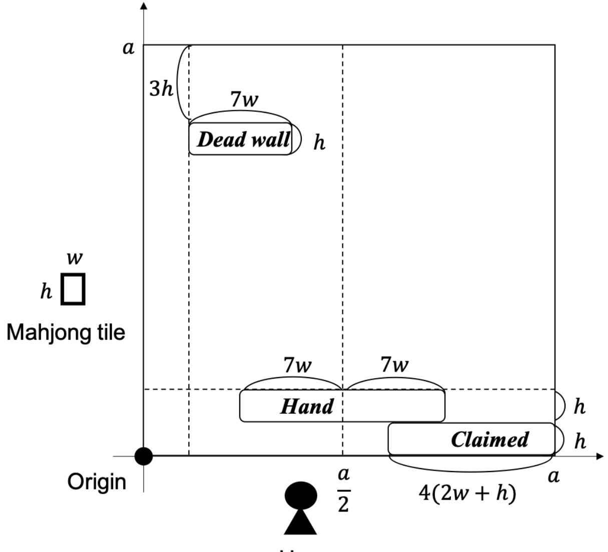

Figure 4: W_{AR}:AR Mahjong world

4.1 Recognize attributes

We describe how the attribute recognition module works to determine the tile attributes. This module consists of two submodules, as shown in Figure 3. The first is the table detection submodule, which is used to obtain axes and an origin of the AR world using the feature points of the table. The second is the attribute detection submodule, which recognizes attributes.

We next define a map with tool lengths for attribute recognition preparation. We define the AR world as W_{AR} and the real world as W_{real}. An AR world uses a 3D origin and axes. The system sets the origin of W_{AR} to the corner of the left-front point of the Mahjong table. The axes are aligned with the edges of the table. In W_{AR}, the lateral direction for users is the x-axis, and the longitudinal direction is the y-axis. Then, we define the length of the tools. The flat surface of the table is a cm^2. The long side of a tile indicates its height, and the short side indicates its base. The height is h[cm], and a base is w[cm]. We define three domains to define the attribute areas based on the lengths. Our heuristics are as follows:

$$\begin{aligned} \textbf{\textit{Hand}} &: \frac{1}{2}a - 7w \leq x \leq \frac{1}{2}a + 7w, h \leq y \leq 2h \\ \textbf{\textit{Claimed}} &: a - 4(2w + h) \leq x \leq a, 0 \leq y \leq h \\ \textbf{\textit{Dead wall}} &: 3h \leq x \leq 3h + 7w, a - 4h \leq y \leq a - 3h \end{aligned} \tag{1}$$

We plot the 2D coordinate space, a player, and a Mahjong tile in Figure 4. This definition is managed at the millimeter level. The areas of attributes must have a great deal of flexibility for individual differences during game play. We, therefore, define the areas requiring such flexibility. The position of the ***Dead wall*** with respect to a player depends on the roll of a dice when starting a game. The remainder of this section describes the two submodules.

The table detection submodule detects the plane of the Mahjong table using ARKit. The Mahjong table becomes the plane, W_{real}, in W_{AR}. The many feature points of the Mahjong table enable the system to detect the plane. Next, users capture an image of the vicinity of the corner and the system detects the edges and the corner of the table using OpenCV. We

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

tried to detect the corner directly using a Harris corner detector, but it detected too many candidates because of noise. It is still difficult to detect the corner with high accuracy. To resolve this, the system first detects the edges of the table. The intersection of those lines is assumed to be the corner. OpenCV binarizes the image and further ratifies the detection. This method is effective, because the color of the edges is different from those on the inside. We employ a canny algorithm for detecting the edges and Hough transforms for constructing the lines. Detected lines become the candidates of the axes, and it is expected that the axes will intersect at a right angle. By narrowing the candidate lines, we can restrict the angle between a pair to improve accuracy. The angle of lines is θ [rad], and we restrict it using the following definition:

$$\frac{4}{9}\pi \leq \theta \leq \frac{1}{2}\pi$$

The system places dot objects on the intersection points of the edge candidates. A user then selects the correct point as the corner by tapping the display. Then, the system quickly obtains the origin and axes, because each origin corresponds to two lines. With this approach, the system obtains an accurate origin with correct axes. Next, the system transforms the AR space based on the detected corner and edges. The system sets the origin to the corner and the axes to the edges. ARKit initializes the origin and axes of the AR space based on the orientation and position of the device. ARKit saves the information as a 4×4 identity matrix. The system then calculates a rotation matrix based on the detected edges and the corner. Then, the system actively maintains the necessary relative coordinates from the detected corner.

We next describe the attribute detection submodule, which is used to obtain attributes. ARKit has a built-in method of projecting a 2D position on the screen to 3D W_{AR} coordinates. The system identifies the camera's position relative to the Mahjong table in W_{AR} for each frame. The system estimates the self-location and superimposes information on the space, as shown in Figure 9. The system obtains an attribute when the self-location is mapped to any of the areas indicated, as shown in Eq. (1).

4.2 Recognize the classes

We next describe how the system recognizes the classes from an image using its attributes. The class recognition module recognizes the classes of the tiles in the image captured by the user, and the module comprises two submodules. The first is the tile detection submodule, which detects contours and passes each cropped tile image to the other submodule. The second submodule is the tile classification submodule, which classifies the cropped image into one of the 34 classes. Algorithm 1 describes the recognizing process. We next describe the detail of Algorithm 1. In line 1, the function $attribute_to_number(X)$ returns the *number* from X. The *number* refers to the expected number of tiles having the attribute. We describe the detail how the system decides the *number* in Section 4.3. In line 2, the function, $make_empty_array(X)$, initializes the *classes* variable. *Classes* then sets class names with lengths of X. In lines 3–20, users repeatedly recognize tiles while *have_empty* is "true." The function, $have_empty(X)$, returns "true" or "false" regarding array X having an empty element. In line 4, $tile_images$ sets Mahjong tile images with length of *number*. In lines 5–11, the tile detection submodule repeatedly detects Mahjong tiles until the value of *number* is detected. The function, $count(X)$, returns the value of X. In line 6, the function, $capture()$, returns a Mahjong-tiles image. In lines 7 and 9, the function, $find_contours(X)$,

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

Algorithm 1 Procedure for recognizing Mahjong tiles

Input: $attribute$

Output: $classes$

1: $number = attribute_to_number(attribute)$
2: $classes = make_empty_array(number)$
3: **while** $have_empty(classes)$ **do**
4: $tile_images = []$
5: **while** $count(tile_images) \neq number$ **do**
6: $image = capture()$
7: $overall_contours = find_contours(image)$
8: $crop_image = crop(image, overall_contours)$
9: $tile_contours = find_contours(crop_image)$
10: $tile_images = crop(crop_image, tile_contours)$
11: **end while**
12: **for** $i = 0 \ldots number - 1$ **do**
13: **if** $empty(classes[i])$ **then**
14: $class = classify(tile_images[i])$
15: **if** $check_result(class)$ **then**
16: $classes[i] = class$
17: **end if**
18: **end if**
19: **end for**
20: **end while**

finds $overall_contours$ from X. In lines 8 and 10, the function, $crop(X,Y)$, crops from the X according to Y and returns $crop_image$. We describe the detail of line 6–10 in the tile detection submodule. In lines 12–19, the tile classification submodule classifies up to $number$ Mahjong tile images into 34 classes. In lines 13–18, the tile classification submodule classifies $tile_images[i]$ if the function, $empty(X)$, returns that X is empty. The function, $classify(X)$, in line 14 is the tile classification submodule. In lines 15–17, the function, $check_result(X)$, returns the result of a user checking that X is true or false and fixes the true result. Lines 5 and 15 provide error handling, as described in Section 4.3. The remainder of this section describes the two submodules and their error handling.

The tile detection submodule is described first. The submodule is built using OpenCV. The tile detection submodule has the function of detecting contours via two processes. The former detects an overall contour of the group of arranged tiles. The latter detects each tile's contour. Figure 5 shows an example of the group of arranged tiles. It is easy to detect them by binarizing the image, as shown in Figure 6, where the system leverages the difference of the colors between a tile and the Mahjong table. Because contours are usually twisted, the system must reform them into a rectangle to obtain accurate tile contours, which the system does next. The simplest way is to divide the overall contour into separate tiles. However, the system cannot get the number of tiles in advance, and the tiles' sizes are not uniform, depending on the angle of the system image's viewpoint. The system should, therefore, detect the actual contours of the tiles. However, simple binarization cannot succeed, because the borders between tiles are unclear, as shown in Figure 6. Therefore, we adopt an adaptive

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

Figure 5: An example of arranged tiles

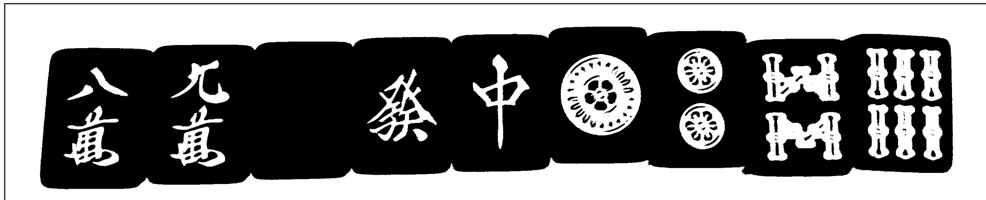

Figure 6: Binarized image

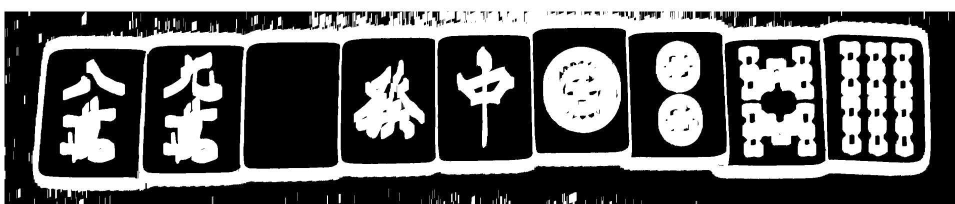

Figure 7: Adaptive binarized image

threshold using neighborhood pixels. The *cv2.adaptiveThreshold* [2] method emphasizes borders and binarizes the image. The system then dilates the value of borders in a longitudinal direction. The system gets an image like the one as shown in Figure 7. Because this method can get borders clearer than simple binarization, the system can effectively obtain images of each tile. To reduce the noise from *cv2.adaptiveThreshold*, the first process of detecting the overall contours using the simple binarization narrows the area. When detecting a ***Dead wall***, the system mis-detects downward tiles whose surfaces are yellow. A ***Dead wall*** contains upward and downward tiles adjacent to one another. The system sets a color threshold for its detections and detects only upward tiles. Thus, the system can classify images in any of the 34 classes. The system detects each tile in order from left-to-right as viewed from the player.

Next, we describe the tile classification submodule. This submodule is built using a convolutional neural network (CNN) [10]. CNN has advantages of decreasing errors compared with normal neural networks. We adopt Tensorflow and Keras to construct this subsystem. Figure 8 shows the architecture of our network, which is very simple. It consists of four convolutional layers and two fully-connected layers. We adopt a rectified linear unit for activating the function and our model architecture. The input is an image of a Mahjong tile that is $70\times50\times3$ pixels with an output comprising a 34 class probability vectors. We captured Mahjong tiles and train the network using the images. We captured 600 tile images using the tile detection submodule. Using the same method to get images for recognizing targets and training classifying CNN, we then increased the data. The 600 tile images con-

[2] https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

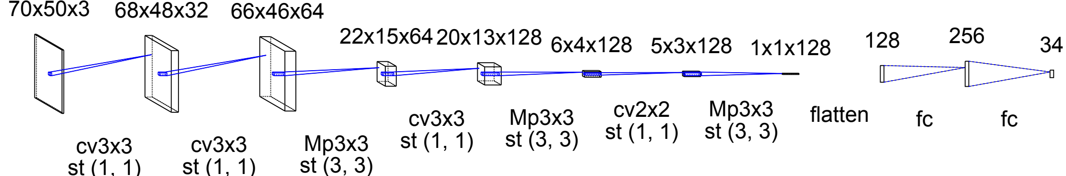

Figure 8: This figure illustrates the model architecture. cv applies to the convolutional layer, Mp is the max-pooling layer, fc is the fully-connected layer, and st is the stride of the filter. The numbers atop of each layer reflect the shapes of input data. The model takes a Mahjong tile image of 70×50×3 pixels and calculates the class probability.

tained about 17 or 18 images of each tile. Data augmentation methods account for blur and adjust brightness using two variations and a projective transform. The datasets comprise 3,000 images. We resized them to 70×50 pixels and trained our network on the datasets. Then, we used the trained model with an iOS device using CoreML, a machine-learning framework. CoreML are optimized to perform calculations at high speeds with low loads on iOS devices. The system does not require client-server data transfer. This approach is cost effective, because there is no communication load and preserves user privacy. Coreml-tools converted a Keras CNN model to a CoreML model, and by using this tool, our system can classify tile images into any of the 34 classes.

4.3 Error handling

The system provides a two-step error detection function to retry the recognition if needed. The first and second steps are automatic and manual error detection, respectively. First, the system applies automatic error detection. Then, it uses the manual error detection. It is possible for the class recognition module to misrecognize the classes. However, the system practically handles errors and reduces the time required to get results. The tile classification submodule takes longer than the tile detection submodule, because of the CNN model. We, therefore, reduce the number of times that the CNN classifies images. Next, we describe two methods of accomplishing this.

First, we describe a method used in the tile detection submodule. This method is automatic. The system judges the success or failure of a detection. The tile classification submodule classifies an image when the system detects success. Almost errors occur in the tile detection. The system finds some errors by considering the Mahjong rules. We introduce five error detection rules as follows:

- Error if $tiles(\textbf{\textit{Hand}})$ is not 2, 5, 8, 11, or 14.
- Error if $tiles(\textbf{\textit{Claimed}})$ is not 3 or greater.
- Error if $tiles(\textbf{\textit{Hand}}) + tiles(\textbf{\textit{Claimed}})$ is not 14 + the number of ***Kongs***
- Error if $tiles(\textbf{\textit{Dead wall}})$ is not 1.

The function, $tiles(X)$, in Algorithm 1 returns the number of tiles in X. ***Claimed*** has a ***Kong***, and ***Hand*** does not have a ***Kong***. The system repeats detections while the number of detected tiles does not match that established by the rules.

Next, we describe a method to repeat recognitions. The method is manual. The system displays recognized tile images in order from left-to-right on the display. The system then

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

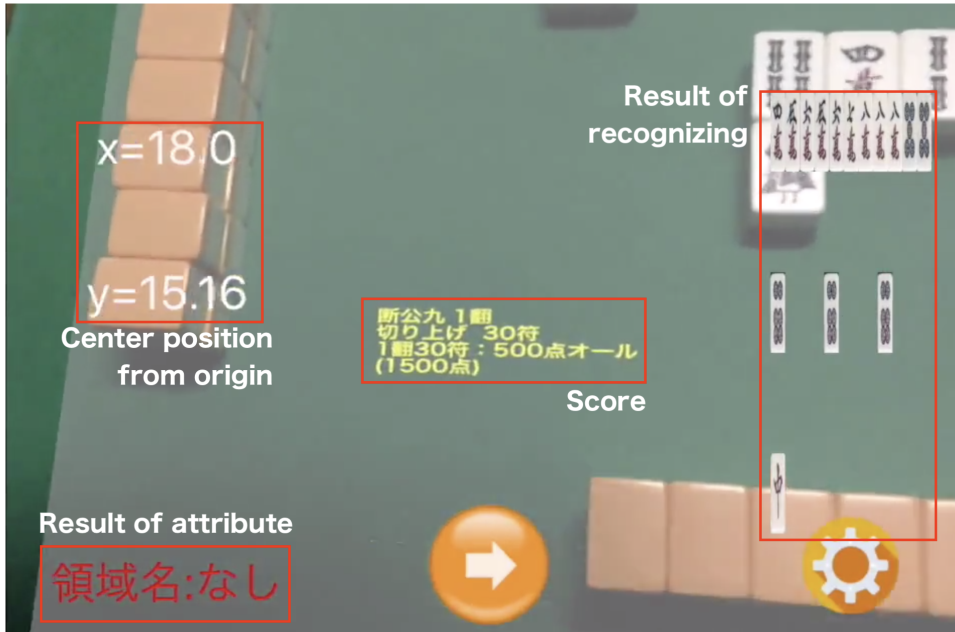

Figure 9: The system displays the score of the user in the AR space. The score is calculated from recognized tiles considering their context-aware attributes.

attempts to recognize the mistaken tiles pointed out by users again, but it keeps the successful results. The system repeats until all tiles are correctly recognized. The number of tiles classified by the CNN decreases as the system repeats recognitions.

4.4 Showing the result

This section describes how the system presents the result to users. We intend to implement this system using AR glass, as mentioned in Section 1. This technology enables users to see the score by observing the board through the phone's camera. The system presents the score using the tabletop, already detected, as the display. The Mahjong table has a simple design, and users can read the text easily. Figure 9 presents an example of execution. The pair value of x and y reflects the coordinates of the center position of the device from the origin. The red characters show the attributes from the captured screenshot. Attributes are ***Hand***, ***Claimed***, ***Dead wall***, and ***None***. The yellow characters reflect the score. The display layout is simple and requires further development. For example, the system should also superimpose labels that indicate Mahjong roles. Roles include ***Chow***, ***Pung***, ***Kong***, and ***Head***. Mahjong beginners can easily understand the rules in this case.

5 Experiments

5.1 Experimental setup

We conducted experiment wherein we evaluated attribute and class recognition on a commercially available Mahjong set in a bright room. We used a 62.5-cm^2 Mahjong table and 2.5-cm-high $\times$ 1.8-cm-wide tiles. We placed the tiles on the table as shown in Figure 1 and Figure 4. We use an iPad (iOS12.0) as the experimental device. First, we measured the

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

Figure 10: Evaluation situation

accuracy of self-localization to evaluate attribute recognition, which depends only on the performance of self-localization. We measured the positions of the four corners of the table and the area of ***Hand***, ***Claimed***, and ***Dead wall***. In this experiment, an error indicates a difference between the real position and one in the AR world. The iPad was moved in order of the origin to ***Hand***, ***Claimed***, and ***Dead wall***. Therefore, the total movements from the origin of ***Dead wall*** was longer than of ***Claimed***, and ***Claimed*** was longer than ***Hand***. We repeated this process 10 times. Regarding accuracy, we defined an allowance of error. That is, the detected areas must not overlap. The closest pair of areas were ***Claimed*** and ***Dead wall***. The distance was about 5 cm, which was twice as long as the height of the tiles. The self-localization error must, therefore, be less than 5 cm. We experimented using a natural player posture.

We also evaluated the performance of the class that comprises accuracy and speed. The recognition process consisted of detecting and classifying. The system recognized ***Hands***, because it was the most numerous, and it was best for evaluating performance. ***Hands*** were of 10 different variations, and we evaluated each 10 times. The system only classified tiles when the system detected 14 of them. Then, we evaluated the accuracy and speed of detecting and classifying. We performed the experiment in a bright room, keeping the angle and the distance of the iPad at 30 cm and 45°, as shown in Figure 10.

5.2 Results

First, we present the result of attributes in Tables 1 and 2. Table 1 shows the errors of the two directions in each area. Table 2 presents a summary of Table 1. The overall average error was 2.77 cm, and the maximum error was 8.00 cm. The variances of B and C were

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

Table 1: Means of two directions errors[cm]

		1	2	3	4	5	6	7	8	9	10
Hand	x	3.76	3.75	4.85	3.65	4.32	3.92	4.17	4.40	4.13	3.91
	y	0.72	0.91	2.94	0.30	0.42	1.04	1.08	2.78	0.63	0.17
Claimed	x	3.94	1.38	1.82	2.91	3.85	3.39	5.73	5.07	4.07	3.59
	y	0.88	0.86	5.11	0.27	0.44	0.90	1.55	1.62	4.11	0.52
Dead wall	x	1.74	0.92	1.71	0.37	2.22	2.09	8.00	5.19	4.78	2.46
	y	4.57	5.49	5.15	2.01	2.36	3.23	0.54	1.79	2.33	5.30

Table 2: Summary of Errors

		min [cm]	max [cm]	mean [cm]	variance	correlation coefficient	accuracy [%]
Hand	x	3.75	4.85	4.09	0.13	0.240	100
	y	0.17	2.94	1.10	0.95		
Claimed	x	1.38	5.73	3.57	1.75	-7.59×10^{-3}	70
	y	0.27	5.11	1.63	2.71		
Dead wall	x	0.37	8.00	2.95	5.46	-1.81×10^{-2}	50
	y	0.54	5.49	3.28	3.02		

higher than that of A. The moving distance of the device was longest in the order of C, B, A. The longer the distance, the lower the stability. We defined the error allowance in the self-localization procedure that provided the overall mean error. Additionally, the value was 4.4%, compared with the length of the Mahjong table side. We could add recognition targets, but we would constrict them in the system. However, the maximum error exceeded the allowance, and the accuracy of B and C did not reach 100%. We therefore need to improve stability, because the system cannot obtain an accurate score without recognizing attributes correctly. For the correlation coefficient, the maximum absolute value of the correlation coefficient was 0.24, and no correlation was obtained from this result.

Moreover, we present the result of recognizing classes. The recognized targets were 100 ***Hands***, consisting of 1,400 tiles, and the system detected 63 ***Hands***. These consisted of 882 tiles, and the system classified 830 of them. Therefore, the accuracy of detection was 63% and that of classifying was 94.1%. The total accuracy was 59.3%. The accuracy of detecting tiles was much lower than that of classifying. Moreover, the time spent detecting and classifying was 0.46 s.

6 Discussion

To consider the result of recognizing attributes, we estimated the causes of errors in self-localization. Vision-based tracking depends on tracking feature points. Therefore, the system cannot track them accurately if a user moves the device camera rapidly. The physical sensors are also affected by this. However, the system must recognize all tiles in A, B, and C areas to obtain an accurate score. Reducing the movement of a device improves accuracy, and we can decrease some movements when the system begins recognizing attributes. Moreover, through the display, the system can influence device movement. For example, if needed, the system will show the text, “please move more slowly.” Another solution would

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

Table 3: Probabilities of recognizing different products

(a)	(b)	(c)	(d)
1.00	0.90	0.00	0.01

be to share the AR world and the tiles of each user with other players. A player would require others' origins and axes. Other players do not have to see them, and each player only recognizes tiles nearest themselves. For example, users need not recognize a ***Dead wall*** in the situation shown in Figure 4. This decreases some movements and improves the accuracy of recognizing attributes. The current version of ARKit supports AR-world sharing across multiple devices. Each system shares feature points in the real world, enabling origins and axes to be shared. Our system can realize this function using ARKit.

Next, we consider the result of detecting and classifying. First, we describe the process of detecting. The accuracy of recognizing classes is low. However, the system can obtain the image any number of times and can modify the results of class recognition. As described in Section 4.3, the system displays recognized tile images, and users check and modify the results. System accuracy increases if the system can repeat detection in a variety of conditions at different distances and angles. Realistically, the accuracy of repeating detection five times can reach 99.3%. The time spent detecting and classifying takes 0.46 s. Thus, the system can obtain classes completely and accurately within 2.3 s. This result is much faster than the one obtained when a new player calculates a score by themselves. Therefore, this system is quite practical for beginners.

Next, we describe the process of classifying and training our model using a commercially available set of Mahjong tiles. However, the printed contents on tiles vary by product manufacturer. Thus, we must consider cases wherein the system must recognize Mahjong tiles of different brands. Table 3 shows several Mahjong tiles. Our CNN model trained (a). (b) resembles (a). The accuracy of (b) is higher than (c) and (d) and lower than (a). These results suggest the growing difficulty of general Mahjong tile recognition using limited training data. Thus, we must then create a new dataset containing a variety of Mahjong tile sets. However, we cannot classify all tiles accurately by training in advance, because tiles printed with varying characters or figures can still change in the future. Users should, therefore, retrain suitable models using the tile detection submodule.

7 Conclusions

We developed a context-aware image recognition method based on AR technologies to help Mahjong beginners keep score. The system successfully recognized the classes and attributes of Mahjong tiles, and, using OpenCV and a CNN, assisted in recognizing tile classes. Moreover, ARKit enabled the system to recognize attributes via self-localization. In addition to [3], this paper described an error-handling process. The system can therefore obtain practical accuracy using our error-handling mechanism. The accuracy of detecting tiles and recognizing attributes was good enough to provide an acceptable support system. The system recognized figures and contexts of tile positions well-enough to determine the

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.

score of Mahjong tiles with good accuracy.

Acknowledgments

This work was supported in part by JSPS KAKENHI Grant Number JP16K00420, 19K12097, 19K12266. The authors would like to thank Enago (www.enago.jp) for the English language review.

References

[1] Y. Fujimoto, G. Yamamoto, T. Taketomi, J. Miyazaki, and H. Kato, "Relation between Displaying Features of Augmented Reality and Userś Memorization," TVRSJ, Vol.18, No.1, pp.81-91, 2013.

[2] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin, "Context-based vision system for place and object recognition," Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV), pp.273-280, 2003.

[3] R. Suzuki, T. Ozono, and T. Shintani, "An Offline Mahjong Support System Based on Augmented Reality with Context-aware Image Recognition," ESKM2019, IEEE, pp.127-132, 2019.

[4] S. Cheng, P. K. Chow, Y. Song, E. C.S. Yu, A. C.M. Chan, T. M.C. Lee, and J. H.M. Lam, "Mental and physical activities delay cognitive decline in older persons with dementia," The American Journal of Geriatric Psychiatry, Vol.22, Issue 1, pp.63-74, 2014.

[5] R. Suzuki, T. Ozono, and T. Shintani, "Implementing a Real-time Mahjong Hand Recognition System in the Real World Using AR Technologies and Deep Learning," Tokai-Section Joint Conference on Electrical, Electronics, Information, and Related Engineering, 1p, 2018.

[6] Y. Matsui, H. Sawano, and S. Mizuno, "A Proposal of a Score Calculation System for Mahjong on a Smartphone," DICOMO 2013, pp.2145-2150, 2013.

[7] D. Nister, Naroditsky, and J. Bergen, "Visual odometry," Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Vol.1, pp.652-659, 2004.

[8] F. Zhou, H. Been-Lirn Duh, and M. Billinghurst, "Trends in augmented reality tracking. interaction and display: A review of ten years of ISMAR," 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, pp.193-202, 2008.

[9] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics," Intelligent Industrial Systems, Vol.1, Issue 4, pp.289-311, 2015.

[10] K. Fukushima, and S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position," Pattern Recognition, Vol.15, No.6, pp.455-469, 1982.

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.