A Low-cost Raspberry Pi-based Control System for Upper Limb Prosthesis

Watanabe Masaki,[a)] Mark Ikechukwu Ogbodo,[b)] and Abderazek Ben Abdallah[c)]

Adaptive Systems Laboratory, School of Computer Science and Engineering, The University of Aizu, Japan.

[a)]*Corresponding author: s1260084@u-aizu.ac.jp*
[b)]*Electronic mail: d8211104@u-aizu.ac.jp*
[c)]*Electronic mail: benab@u-aizu.ac.jp*

Abstract. In recent years, robots have been introduced in most factories. However, manual work still continues to be done in some places where giant robots cannot be installed. In particular, traditional Japanese crafts are done by hand, and people that engage in such crafts are called craftsmen. Generally, such artisans need years of training and cannot become experts right away. One of the problems these artisans face is the lack of successors. To address this challenge, this paper proposes a raspberry pi hardware based control method for a prosthetic hand using hand gestures from camera sensor, which will allow a prosthetic hand to learn the hand movements of the craftsmen and perform the crafts. The advantage of this is that there is no need for training, which usually takes years. To control the prosthetic hand, hand gestures are captured from a camera sensor, converted to HSV and binarized, and then classified into one of five gestures using a CNN implemented on the raspberry pi hardware. The recognized gesture is then relayed to the prosthetic hand to mimic the classified gesture. A dataset containing 2000 captured images of each gesture was created to evaluate the performance, and these gestures clearly define the closing and opening of the fingers. Using a 32x32 hand gesture image dataset captured from camera, we validated the trained CNN first in software for hand recognition without using Raspberry Pi, and achieved an accuracy of 99.63%, and then implemented on the raspberry pi, and performed real-time evaluation by recognizing five hand gestures captured from the camera sensor in real-time. Out of the hand gestures, four were correctly recognized. We presented the design of a low-cost prosthetic hand based on raspberry pi hardware, and evaluated its real-time hand gesture recognition. The evaluation result show that the proposed system is able to correctly recognize four hand gestures.

Keywords: Deep Neural Network, Prosthesis, Hardware, Frame-based, Real-time

INTRODUCTION

The use of machines in factories have grown over the years to become the norm [1]. However, there are still many types of work that must be done manually. Specifically in Japan [2], various fields such as painting, pottery, and metal processing are handcrafted. Individuals who engage in such crafts are called craftsmen. It takes decades to train them, and the number of people in this profession are not many due to lack of successors [3]. To address this challenge, we propose a prosthetic hand which will perform the same hand movement as a craftsman by tracking and memorizing their skills, and this paper presents a method for recognizing hand gestures, and tracking its movements in order to make a prosthetic hand move in a similar manner. The system recognizes images of five different hand gesture captured from a camera sensor, and based on that, the prosthetic hand move in the same way. There are various methods related to hand gesture recognition. One is to wear a special glove and use sensors to collect data on the shape of the hand [4]. This method requires the person to wear gloves, which limits the space available and makes it difficult to keep the gloves on at all times in daily life. Another method is the use of cameras, where hand gestures are captured and fed to a convolutional neural network (CNN) [5] for recognition. This approach requires substantial amount of training data. Electromyography (EMG) is also another method of hand gesture recognition [6], where an EMG sensor is attached to the arm to acquire neural signals from the muscles of the arm during movement [7]. Like the camera, this also requires a huge amount of training data for each movement. In this paper, we propose a method for hand gesture recognition based on image data from a camera.

The reason we chose to use raspberry pi is because of its low cost. Mechanizing a factory is quite expensive. Therefore, we chose the Raspberry Pi because it is not too large and can be easily installed, assuming there is no factory to do it manually.

The rest of this paper is organized as follows; Section gives an overview of the proposed raspberry pi control system. Next, Section describes the creation of the dataset for hand recognition. Then in Section , we described and trained the convolutional neural network (CNN) model and validated its accuracy. Finally, in Section , we proposed a method to run the trained model created in Section on a Raspberry Pi to recognize hand gestures and move the prosthetic hand in real-time.

HAND GESTURE RECOGNITION

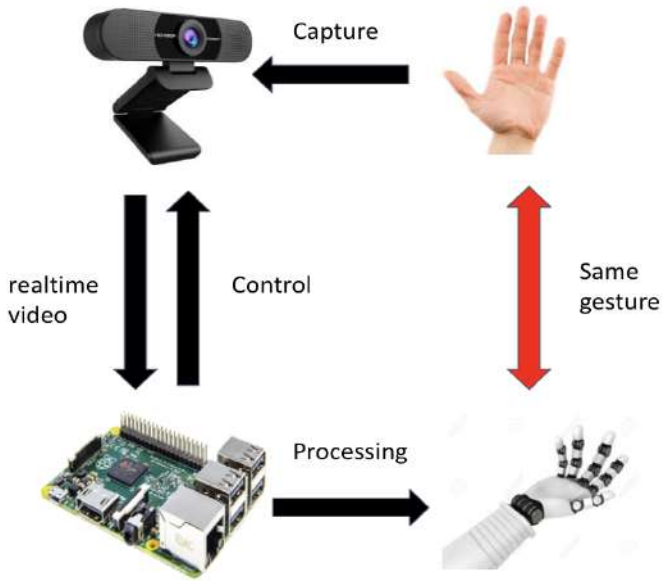


FIGURE 1: Proposed system development flow.

System Architecture

As shown in Figure 1, the system proposed in this paper is made up of an RGB camera which is used as a sensor for capturing hand gestures, a raspberry pi board where the CNN used to recognize the hand gestures captured from camera is implemented, and finally the prosthetic hand that mimics the hand gestures recognized by the CNN in real-time [8].

Data Set

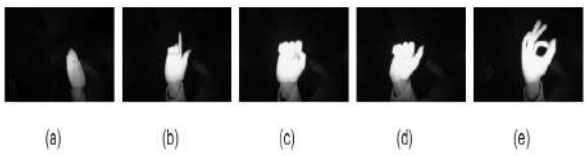


FIGURE 2: Hand gesture image dataset with five classes.

The data set used in training the hand gesture recognition CNN model was taken from Kaggle [9]. A total of five hand gestures; 01_palm, 02_elle, 03_fist, 05_thumb, and 07_ok described in Figure 2 was used. We also prepared some images with just black background, so that the CNN can also learn to detect when there is no gesture because using the dataset alone can make the CNN recognize some classes even when the camera does not show any hand gestures [8]. There are 2000 data sets for each gesture. A total of 12,000 images were prepared. First we separated the data for training and testing then, we used Sklearn's train_test_split method [10] to split the data. We divided it into 8,400 pieces for training and 3,600 pieces for testing.

The kaggle images were not taken directly from a camera, but from a Leap motion sensor. These images however, are displayed in the same way whether they were displayed in RGB or grayscale, so they were processed in grayscale to normalize the image. The original hand gesture images from kaggle are in RGB, so the model processed as RGB is also provided for comparison.

Data Processing

To process the data from Kaggle, we used opencv's imread method [11] to load the images in BGR. Since the training data is like grayscale, we, performed the same binarization process on them. We tried two patterns when doing the binarization process, one with BGR (Black, Green, Red) as HSV (Hue, Saturation, Value) and the other with GRAY. For the comparison, we tried training and verification with RGB without binarization.
We determined the upper and lower limits of to binarize the training data in HSV. In this case, the lower limit is set to (0,0,0) and the upper limit is set to (0,0,20), which is binarized by using OpenCV's threshold method. The data set after the binarization process is shown in Figure 3.
The threshold method for grayscale was also used. The dataset after binarization in grayscale is shown in Figure 4. We examined two patterns and various image sizes to increase the accuracy. The image dataset for training was resized to 16×16, 323 × 332, and 64×64 as a preprocessing step.

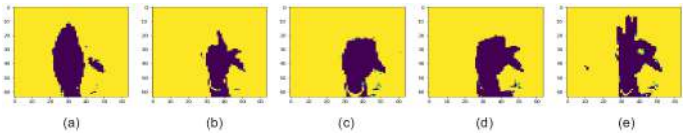


FIGURE 3: Hand gesture images processed with threshold HSV.

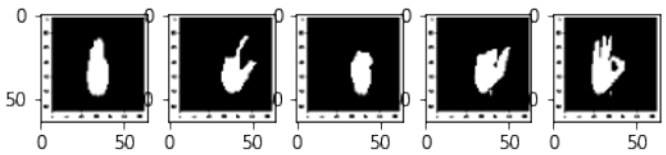


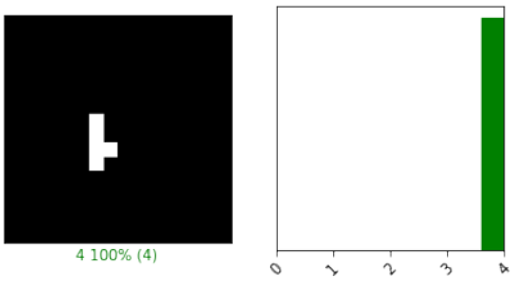
FIGURE 4: Hand gesture images processed with hreshold GRAY.

Size	Processing	Accuracy(%)
16x16	GRAY	97.40
16x16	HSV	98.47
16x16	RGB	98.70
32x32	GRAY	99.40
32x32	HSV	99.63
32x32	RGB	99.19
64x64	GRAY	99.71
64x64	HSV	99.99
64x64	RGB	98.97

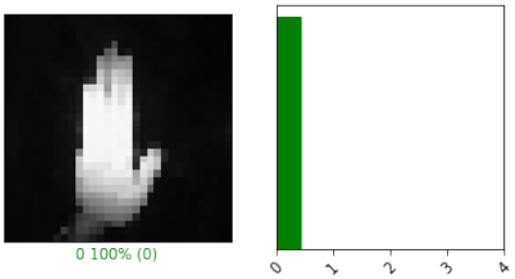
TABLE I: Summary of evaluation result using different image sizes and pre-processing methods.

Architecture and training of CNN model

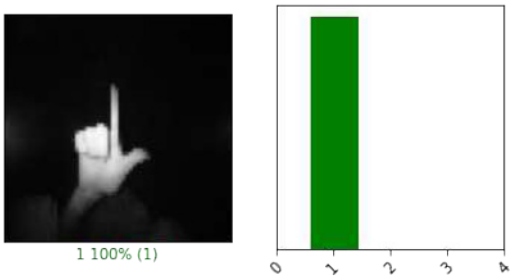
The architecture of the CNN model created for the recognition of hand gesture is described in Figure 6. In the convolutional layer, we used 32 3x3 filters to create the activation map, and the rectified linear unit (ReLU) activation



(a) 16x16 hand gesture image.



(b) 32x32 hand gesture image.



(c) 64x64 hand gesture image.

FIGURE 5: Prediction result of recognized hand gestures.

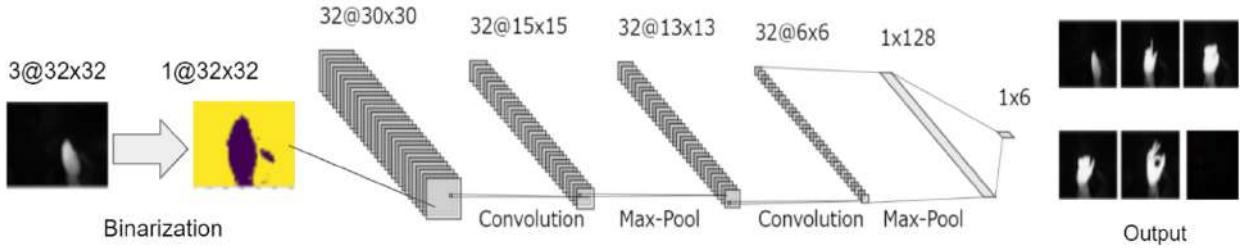


FIGURE 6: Architecture of CNN for hand gesture recognition.

function was used to map the output. Next, the Pooling layer was reduced the output of the convolution layer using a pool size of 2x2, which was repeated twice. In the next step, the 3D data was converted to 1D in the Flatten layer before sending to the fully connected layer. The fully connected layer uses the ReLU function to make the output 128. Finally, the softmax activation function was used to map the output results to different classes.

Training and Evaluation

The data collected from kaggle was split for training and testing. The test results after the training are shown in Table I. A training accuracy of over 98% was achieved for all the utilized image sizes, With the 16x16 Gray images being an exception. The results of the model's predictions are shown in Figure 5. The model used in this case was the one without the binarization process to make predictions. However, even with the model after binarization, we observed that the 16x16 image was squashed and could not be recognized correctly.

(a) (b) (c) (d) (e)

FIGURE 7: Hand gestures captured from camera for testing.

Offline Test

To perform offline testing, we used actual images of hand gestures shown in Figure 7 which were taken using three patterns with sizes of 32x32 and 64x64 (six models in total).

In RGB, neither 32x32 nor 64x64 could produce the accuracy that we had expected. This was due to the fact that the training data used black and white images, and we thought that the correct gestures were not recognized. The next step was the grayscale and binarization process. For the recognition, all the hand gestures were recognized with more than 94% accuracy except for gesture1 and gesture3. We noted that the accuracy of the recognition method using the grayscale binarization process could be improved by setting better parameters for improved binarization result. For the grayscale binarization method, We used a threshold value in the range of 128 to 140, which was chosen with respect to the images. Next, we used HSV for binarization, and were able to get 4 out of 5 hand gestures correct at 32x32 with 100% accuracy, and 3 out of 5 hand gestures correct at 64x64. The threshold values used for the HSV binarization method were fixed at 0 for the lower limit of Saturation and Value of 255 for the upper limit. Only Hue was changed to specify the range. For offline testing, we set the lower limit to 14 and the upper limit to 28. Figure 8 shows a histogram of the Hue elements of the gesture1 image used in the offline test. From this histogram, we focused on the areas where the elements were clustered together, assuming that the area representing the hand occupied most of the image. We chose the threshold range with the highest accuracy from that clustered area, and couldn't get the expected accuracy because proper extraction by hand was done by binarization, but we tried to binarize and extract the outline. It happened that the contours were aligned with the background, taking the form of unintended hand gestures. Hence, it was necessary to remove the background firmly to improve the accuracy.

Hand Gesture Recognition Latency Measurement

Since our goal is to run in real time, we measured the time between the input of an image and the output of a class for six different models in order to find the best model. The results are shown in Table II. The configuration of the environment used for the experiment is Windows 10, 16GB of memory, and a GTX1060Ti GPU. Python3 is used for model creation and testing. In this environment, the difference was only about 0.01 seconds depending on the resolution and processing. On raspberry pi, we obtained the fastest speed on image resolution of 32x32. In addition, when comparing the accuracy, HSV was a little more accurate than grayscale, so we adopted the 32x32 HSV model and implemented it on the Raspberry Pi which is described in the next section.

Size	Processing	speed(s)
32x32	GRAY	0.05
32x32	HSV	0.05
32x32	RGB	0.06
64x64	GRAY	0.06
64x64	HSV	0.06
64x64	RGB	0.07

TABLE II: Summary of operating latency for various image sizes.

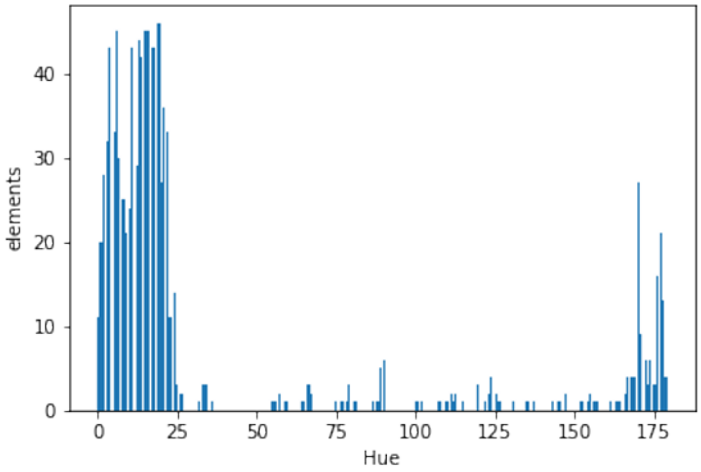


FIGURE 8: Histogram of hue element for gesture 1.

RASPBERRY PI IMPLEMENTATION

For this implementation we used raspberry pi 4 model B which memory is 4GB. The Raspberry Pi can be installed and used at a low cost. However, its performance is less than that of a standard computer. In the case of hand gesture recognition, given that the performance of the Raspberry Pi is lower than that of a standard computer, we considered

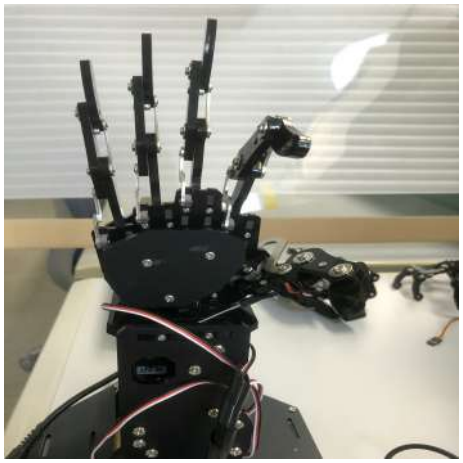


FIGURE 9: Overall view of a right prosthetic hand for system validation.

it is necessary to simplify the image input process to the model by reducing the amount of computation. We also decided to use the HSV 32x32 model. By using a 32x32 image size instead of 64x64, the number of pixels can be significantly reduced and the processing time per image can be reduced as well.

Size	Processing	Accuracy(%)	Speed(s)
32x32	GRAY	14.00	0.18
32x32	HSV	54.00	0.18
32x32	RGB	10.00	0.19
64x64	GRAY	12.00	0.20
64x64	HSV	50.00	0.20
64x64	RGB	0.00	0.20

TABLE III: Summary of operating latency and Evaluation for Raspberry Pi.

In controlling the prosthetic hand shown in Figure 9, We created a program in advance to adjust its servo motors for the five patterns of hand gestures to be performed. In this case, the number of servo motors required was five because the robot arm uses one servo motor for each of the five fingers. With a servo motor for each joint of the robot arm, more patterns can be created.

FIGURE 10: Accurate hand gesture performed by the prosthetic hand.

FIGURE 11: Incorrect hand gesture performed by the prosthetic hand.

Figure 10 and Figure 11 show some of the demonstrations. In Figure 10, the hand gestures were correctly recognized and the robot arm was able in the right way to mimic the gesture. However, in Figure 11, the hand gesture was not recognized correctly and the robot arm moved in the wrong way.

Real-time Hand Gesture Recognition

In order to focus more on real time use, we thought of speeding up the time from image input by the camera to recognition. When capturing an image of a hand gesture from the camera, we created a fixed region of interest in advance so that the hand would appear only in that region, in order to avoid including unnecessary objects other than

the hand in the image. By using the bounding box, the processing speed will be improved because there will be no need to perform time-consuming processes such as object detection. The latency from capturing the hand gesture with the camera to receiving the recognition result was about 0.18 seconds. Compared to the speed test in the environment where the model was created, which took approximately three times longer. In this case, since the Raspberry Pi will be processing the data, we decided that 5 FPS was appropriate. Even taking into account the time for the servo motor to move, it can be utilized in real time without challenge in terms of processing time.

CONCLUSION AND FUTURE WORK

In this paper, we proposed the design of a low-cost prosthetic hand based on raspberry pi hardware, to recognize hand gestures and move the prosthetic hand to mimic the recognized gesture, we made a CNN model which was trained with dataset from Kaggle. To evaluate its performance, we tested it first in a standard PC environment and achieved an accuracy of 99.3%, and the we implemented on raspberry pi where we performed real-time validation, and four out of the five recognized gestures were accurately performed by the prosthetic hand. The operating latency of the system in a standard PC environment and on the raspberry pi hardware was also measured and compared, and the raspberry pi environment was found to achieve three times better latency than the standard PC environment. As future work, the CNN model will be optimized to enable it properly recognize more gestures.

REFERENCES

1. G. Q. Zhang, X. Li, R. Boca, J. Newkirk, B. Zhang, T. A. Fuhlbrigge, H. K. Feng, and N. J. Hunt, "Use of industrial robots in additive manufacturing - a survey and feasibility study," in *ISR/Robotik 2014; 41st International Symposium on Robotics* (2014) pp. 1–6.
2. N. handicraft picture book company, "Nippon handicraft picture book," `https://nippon-teshigoto.jp/` (2022), (Accessed on 02-17-2022).
3. Shikinobi, "Decrease in craftsmen and problems with traditional crafts | the beauty of the four seasons," `https://shikinobi.com/kougei-gensyou` (2020), (Accessed on 17-02-2022).
4. M.-A. A. Faisal, F. F. Abir, and M. U. Ahmed, "Sensor dataglove for real-time static and dynamic hand gesture recognition," in *2021 Joint 10th International Conference on Informatics, Electronics Vision (ICIEV) and 2021 5th International Conference on Imaging, Vision Pattern Recognition (icIVPR)*, edited by G. T. Rado and H. Suhl (IEEE, 2021) pp. 1–7.
5. Y. Rikiya, N. Mizuho, K. G. D. Richard, and T. Kaori, "Convolutional neural networks: an overview and application in radiology," Insights into Imaging **9**, 611–629 (2018).
6. C. Enea, F. Charlotte, B. S. Sumit, T. Gemma, K. Lyes, P. Melika, and D. Elisa, "Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing," Frontiers in Neuroscience **14** (2020), 10.3389/fnins.2020.00637.
7. A. Cisnal, J. Pérez-Turiel, J.-C. Fraile, D. Sierra, and E. de-la Fuente, "Robhand: A hand exoskeleton with real-time emg-driven embedded control. quantifying hand gesture recognition delays for bilateral rehabilitation," IEEE Access **9**, 137809–137823 (2021).
8. W. Masaki, *A Low-cost Raspberry Pi-based Control System for Prosthetic Hand*, Bachelor thesis, University of Aizu (2022).
9. Kaggle, "Hand gesture recognition database," `https://www.kaggle.com/gti-upm/leapgestrecog)` (2018), accessed: 08-10-2021.
10. scikit-learn developers, "sklearn.model_selection.train_test_split — scikit-learn 1.0.2 documentation," `https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html` (2007), (Accessed on 03-03-2022).
11. G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools (2000).