YOLOを用いたPCとEV3-Robot間の物体検出結果のクライアント・サーバー通信

Client-server Communication using YOLO for Object Detection Results between PC and EV3-Robot

Dinda Pramanta, Ninnart Fuengfusin, A.R. Syulistyo, Hakaru Tamukoh

要約

本研究では、畳み込みニューラルネットワーク (CNN) である You Only Look Once (YOLO) をロボットのリアルタイム物体検出に実用化することを探求する。ニューラルネットワークによって信頼性の高い物体検出機能を持つロボットが求められるようになり、処理の高速化・高精度化が進んでいる。しかし、これらの技術に対する需要は依然として高いため、サービスロボットの開発には法外なコストがかかる可能性がある。このような課題を解決するために、LEGO ブロックのカスタマイズ機能を追加した、低価格の教育キット EV3-robot をサーバーとして、ローカルのパーソナルコンピュータ（PC）をクライアントとして、物体検出に使用することを提案する。Ev3dev Python Socket Connection を利用したクライアント - サーバ通信フレームワークを構築し、YOLO の結果をリアルタイムでローカル PC から EV3- ロボットに Bluetooth 経由で送信する。本研究では、YOLO を用いたリアルタイムの物体検出のために、ワイヤレスカメラを搭載した PC 上で事前に訓練された Common Objects in Context (COCO) データセットをテストする。その結果､EV3 ロボットと YOLO 間の通信が成功し､物体検出が実証された。さらに、リアルタイム物体検出時の CPU とグラフィック・プロセッシング・ユニット GPU の処理時間を評価した。

キーワード：CPU, CNN, EV3-robot, GPU, NN, Socket Connection, YOLO.

Abstract

This research explores a practical application of You Only Look Once (YOLO), a Convolutional Neural Network (CNN), for real-time object detection in robots. The demand for robots with reliable object detection capabilities driven by Neural Networks has led to faster and higher accuracy processing. However, the development of service robots may be prohibitively expensive, because the demand for these technologies remains high. To address these challenges, we propose the use of a low-cost educational kit EV3-robot as a server and a local personal computer (PC) as a client for object detection, with the added feature of LEGO Block customization. We establish a Client-Server communication framework utilizing the Ev3dev Python Socket Connection to transmit real-time YOLO results from the local PC to the EV3-robot via Bluetooth. Our experiments involve testing a pre-trained Common Objects in Context (COCO) dataset on the PC, which is equipped with a wireless camera for real-time object detection using YOLO. The results highlight the successful communication between the EV3-robot and YOLO, demonstrating real-time object detection. Additionally, we evaluate the processing times on both the CPU and the Graphic Processing Unit GPU during real-time object detection.

Keywords：CPU, CNN, EV3-robot, GPU, Neural Network, Socket Connection, YOLO.

1 INTRODUCTION

Japan's society has an aging population and a diminishing number of young people[1]. To handle this problem, service robots have been gaining attention[2], for instance, robots are expected to fulfill daily indoor tasks, such as room tidying. To fulfill such tasks, robots must be able to recognize and detect a certain object[3].

In the field of Artificial intelligence (AI), neural networks (NN) have been extensively explored[4,5,6]. NN is equipped with a state-of-the-art perceptual ability, adaptive learning, and sophisticated human interaction. However, processing that information requires fast and high accuracy, especially in-service robots where real-time interaction is almost unavoidable. This affects the increasing need for robots equipped with reliable NN. Therefore YOLO, has gained attention for its fast and high accuracy[7].

To achieve real-time object detection for the service robots, we explore the possibility of the low-cost educational kit of the EV3-robot for being able to integrate with YOLO. We utilize a client-server communication model with the EV3-robot acting as the server and the local PC as the client. By utilizing the feature of fast and high accuracy from YOLO's object detection capabilities, we aim to enable real-time communication. This involves the setup of an Ev3dev Python Socket Connection via Bluetooth.

In our previous hardware-based communication study[8,9], the synchronization of communication processes played a pivotal role. The verification procedure encompassed several phases.

Firstly, we build a client-server architecture for communication between the EV3-robot and the local PC. We conduct an evaluation of latency in this initial phase. The second one is to send pre-trained YOLO weight data from the PC to the EV3-robot. Lastly, we run a real-time connection between the EV3-robot and the YOLO model weights. Within this operational mode, the EV3-robot's motor function was strategically set to stand by and enable the robot to survey and locate the target object until the desired class detection was achieved. Our experimentation also includes testing pre-trained COCO weights with YOLO on the PC in real-time using both CPU and GPU.

2 RELATED WORKS

There are multiple methods for establishing a client-server communication system for robots, which are categorized into 1, 2, and 3-tier architectures. In the context of 2-Tier architectures, this section is structured to discuss approaches where robots perform real-time object detection using YOLOv8. Additionally, we conduct a review of recent robot education and their features.

(1) You Only Look Once version 8 (YOLOv8) as a detection system.

YOLO was first introduced by Redmon et al.[7]. Figure 1 shows the YOLOv8 base model, which is the latest version of this framework which was created in January 2023 by Ultralytics[10]. By dividing an image into a grid of smaller regions and then predicting a bounding box and class probabilities for each object that is present in each region. For object detection, YOLOv8 utilizes the performance parameter by measuring the mean Average Precision (mAP).

$$IoU = \frac{area(C) \cap area(G)}{area(C) \cup area(G)}, \qquad (1)$$

Equation (1) shows the Intersection over Union (IoU) for evaluating the performance of the model for object detection. The C represents the generated candidate bounding box, and G represents the ground truth bounding box containing the object. It calculates the ratio of the overlap and union between the generated candidate bounding box and the ground truth bounding box. The performance of the model improves as the IoU value increases, with higher IoU values indicating less difference between the generated candidate and ground truth bounding boxes.

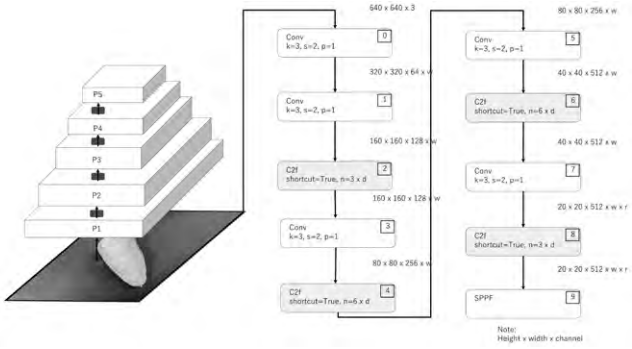


Figure 1. YOLOv8 Backbone System's architecture[10].

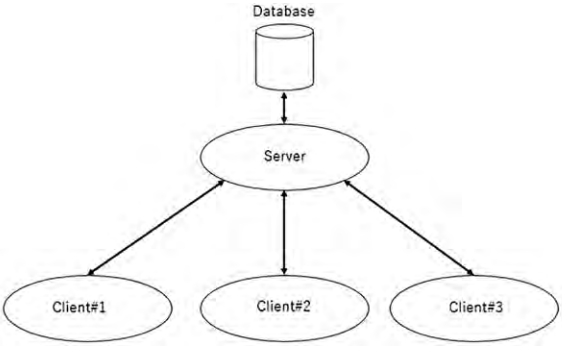


Figure 2. Inter-process communication among client and server[12].

$$Recall = \frac{T_P}{T_P + F_N}, \; Precision = \frac{T_P}{T_P + F_P}, \qquad (2)$$

Equation (2) shows that the Precision-Recall Curve (P-R Curve) is a curve with recall as the x-axis and precision as the y-axis[11]. Each point represents a different threshold value, and all points are connected as a curve. The recall (R) and precision (P) are calculated. True Positive (TP) denotes the prediction result as a positive class and contains a label as to be true; False Positive (FP) denotes the prediction result as a positive class but contains a label as to be false, and False Negative (FN) denotes the prediction result as a negative class but contains a label as to be false.

$$\mathrm{AP} = \int_0^1 p(r)dr \qquad (3)$$

Equation (3) shows the definition for the Average Precision (AP) to find the area under the precision within the range 0 to 1 where p and r are represented as precision and recall. We calculated the mean from the (3) for the multi-class task object detection.

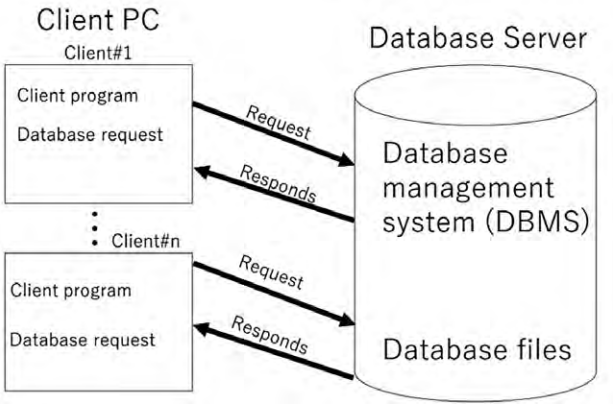


Figure 3. 2-Tier concept of client-server architecture[12].

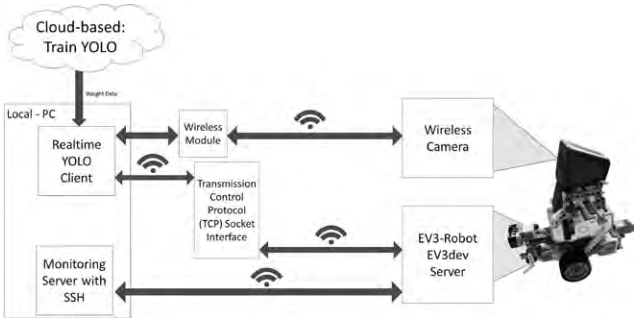


Figure 4. Real-time YOLO with EV3-robot scheme.

(2) Client-Server Systems

Figure 2 shows a client-server system is a software that comprises both clients and server. In this system, clients are responsible for initiating requests, while servers handle the responses to these requests. This architecture facilitates inter-process communication, as it entails the exchange of data between both clients and servers, with each of them carrying out distinct functions[13]. The standardized protocols that clients and servers use to communicate include File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), and Hypertext Transfer Protocol (HTTP).

(3) 2-Tier Client-Server Architecture

Figure 3 shows a 2-tier client-server system architecture is a setup that consists of two primary components: a database server and a client PC. In this arrangement, users operate applications on their local

PCs (the clients) that establish connections to the server through a network[12]. Within this structure, the client application takes on the responsibilities of both coding and executing the programming logic, ultimately presenting the output to the user.

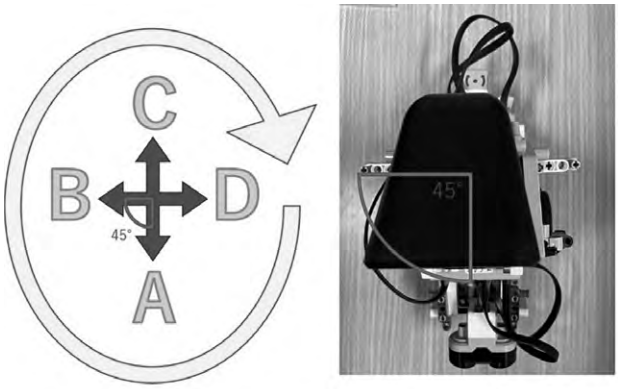
(4) Educational Robot

Proficiency in programming and operating a robot is a crucial skill when it comes to the development of certain technologies, such as service robots. In recent times, the utilization of mobile robots has introduced engaging and motivating learning environments in the field of education[14]. An example of an educational and academic robot kit is the Mindstorms EV3 (EV3-robot) from LEGO. This kit is a versatile platform for constructing and customizing robots.

```
# Example Python code for Server (EV3Dev) starts here:
import socket
server_socket = socket.socket(socket.AF_INET,
 socket.SOCK_STREAM)
server_socket.bind(('EV3Dev_IP_Address', 8000))
server_socket.listen(1)
# Server listening for connections...
client_socket, client_address = server_socket.accept()
print(f"Connection from {client_address}")
while True:
    command = client_socket.recv(1024).decode('utf-8')
    if command == 'forward':
        # Code to move the robot forward
    elif command == 'left':
        # Code to turn the robot left
    elif command == 'right':
        # Code to move the robot right
    elif command == 'backward':
        # Code to move the robot backward
    else:
        # Handle any other commands or conditions here
        pass
client_socket.close()
server_socket.close()
# Example Python code for Server (EV3Dev) ends here.
# Example Python code for Client PC starts here:
import socket
client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
server_address = ('EV3Dev_IP_Address', 8000)
client_socket.connect(server_address)
while True:
    command = input("Enter command (forward/left/
right/backward/exit):")
    client_socket.send(command.encode('utf-8'))
    if command == 'exit':
        break
client_socket.close()
# Example Python code for Client PC ends here.
```

Figure 5. Listing 1.

The EV3-robot[15], is equipped with a variety of sensors, including light, and ultrasonic sensors. It operates using the Linux-based ev3dev operating system, which is stored on a Secure Digital (SD) card. The EV3-robot supports connections with computers or smart devices through Bluetooth operating at a 2.4GHz connection. This robot is powered by an ARM-9 processor and comes with a built-in mini–Liquid Crystal Display (LCD), along with four input and output ports. These features enable the development of control systems that allow them to be executed locally on the robot.



*Clockwise Rotating for scanning “person”
**”A” is an initial direction

Figure 6. 45° EV3-robot rotation.

3 METHODOLOGY

Figure 4 shows a schematic of real-time implementation of YOLO with an EV3 robot for object detection. This section primarily emphasizes the system between YOLO and EV3-robot communication. First, develop a simple remote-control system and then follow the real-time system using the YOLOv8 model for detection. YOLOv8 as a target model, we rely upon Python environment management and setup process for the Ultralytics environment. Additionally, we utilize The Round-Trip Time (RTT) for latency consideration.

(1) Client-Server communication - Ev3dev Python Socket Connection

Client-server communication is a fundamental

concept in computer networking, allowing devices to exchange data over a network. In the context of developing the software part, we utilize the EV3Dev. During the real-time sequence, EV3Dev provides a Debian Linux-based operating system for EV3-robot communication to the PC through Python socket connections[16].

Figure 5 shows a simple remote-control algorithm that sets basic movement as a foundational step using Python socket connections. This communication involves the use of Internet Protocol (IP) addresses to establish connections. The local PC side acts as a client, and sends commands to an EV3-robot, which serves as the server, enabling control over the robot's hardware movements in this case robot's motor.

Figure 5 has five scenarios, for instance forward Command: When the user enters "forward" on the PC, the client sends the "forward" command to the EV3-robot. The server (EV3-robot) then makes the robot move forward. The second command is the Left Command: When "left" is entered on the PC, the client sends the "left" command. The server turns the robot to the left. The third is Right Command: Similarly, "right" triggers the server to turn the robot to the right. The fourth is the Backward Command: "backward" is sent to the server, resulting in the robot moving backward. The last one is Exit Command: If the user enters "exit", the client sends this command to the server, and both the client and server sockets are closed, ending the program.

(2) Run the server from EV3-robot

Figure 5 lines 1-29 demonstrates the foundational structure of a server setup. In the server code, it is essential to implement procedures for handling incoming client requests and generating responses. These actions encompass movement EV3-robot control motor movement. The Python socket server is capable of listening for incoming connections later from the client side. Figure 5 line 5 shows the IP address of the EV3-robot. Deploying and running the developed server code on the EV3-robot.

(3) Getting the weight data from cloud to local PC

The trained COCO weight dataset[17] from Ultralytics is stored in the cloud from ultralytics. Since we are testing the realtime communication during detection, we are only highlighting with specified class from COCO which is "person"(class_id = 0).

Since the weight data model from YOLOv8 has already been downloaded, we add it to the client part and perform real-time processing loop for "person" object detection. The wireless camera is mounted to perform object detection using the YOLOv8 model, focusing on specific classes, and determines if a class_id = 0 (person) is detected and sends a message accordingly via the socket connection.

(4) Latency

Reducing latency is a priority in client-server communications, especially for YOLO applications that require real-time interaction and responsiveness from the EV3-robot. Achieving low-latency communication is crucial for the current system, so we define latency as the round-trip time (ping time) in milliseconds (ms).

$$\text{Latency (ms)} = \frac{\text{Round-Trip Time (RTT)}}{2} \qquad (4)$$

Equation (4) is the mathematical representation. RTT is the total time it takes for a packet of data to travel from the source to the destination and back. Since the RTT measures the time, it takes for a packet to travel from the source to the destination and then back to the source, RTT divider is 2. For instance, if the RTT is 40 ms, the one-way latency (ping time) would be 20 ms.

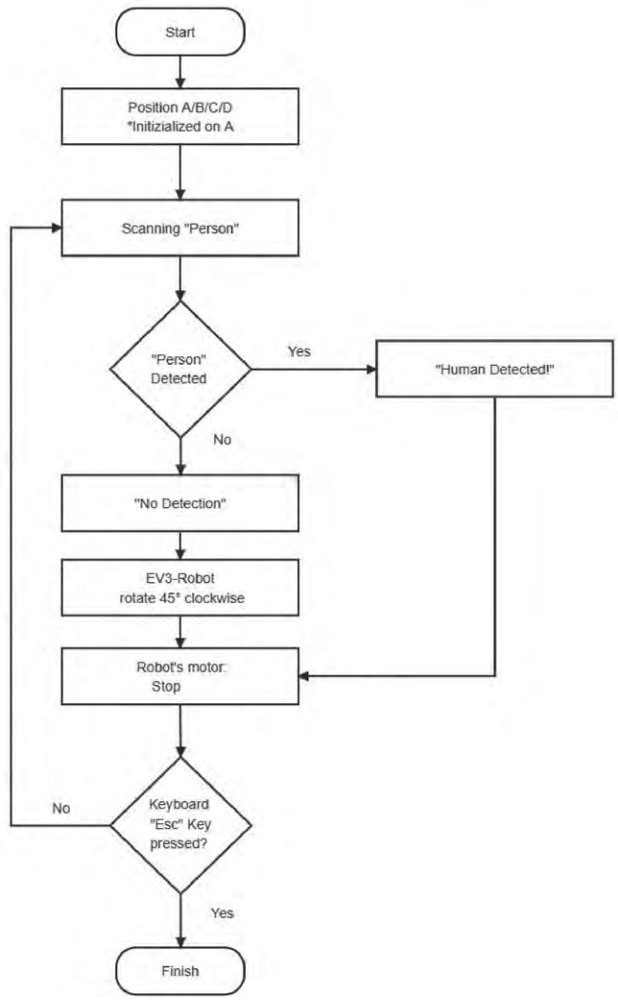


Figure 7. Feedback sequences for Experiment-2 and 3.

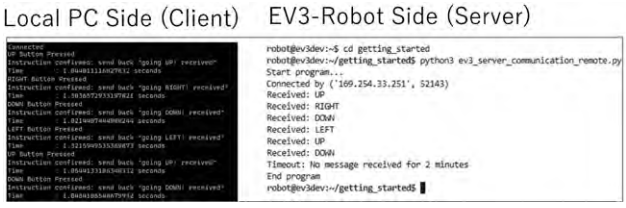


Figure 8. Remote Control Communication System Results.

Keyboard Key	Message	Latency (ms)
UP	"going UP! received"	522
RIGHT	"going RIGHT! received"	652
DOWN	"going DOWN! received"	511
LEFT	"going LEFT! received"	661

Table 1. The latency results of the first four command keys in Experiment-1.

No.	Real-Time (sec)	Detection	AP
1	0	no detection	0
2	0.5	no detection	0
3	1	no detection	0
4	1.5	no detection	0
5	2	1 person	0.85
6	2.5	1 person	0.85
7	3	1 person	0.67
8	3.5	1 person	0.67
9	4	1 person	0.71
10	4.5	1 person	0.71

Table 2. Summary of Experiment-2 results.

No.	Real-Time (sec)	Detection	AP
1	0	no detection	0
2	0.5	no detection	0
3	1	no detection	0
4	1.5	1 person	0
5	2	1 person	0.93
6	2.5	1 person	0.93
7	3	1 person	0.93
8	3.5	1 person	0.93
9	4	1 person	0.93
10	4.5	1 person	0.93

Table 3. Summary of Experiment-3 results.

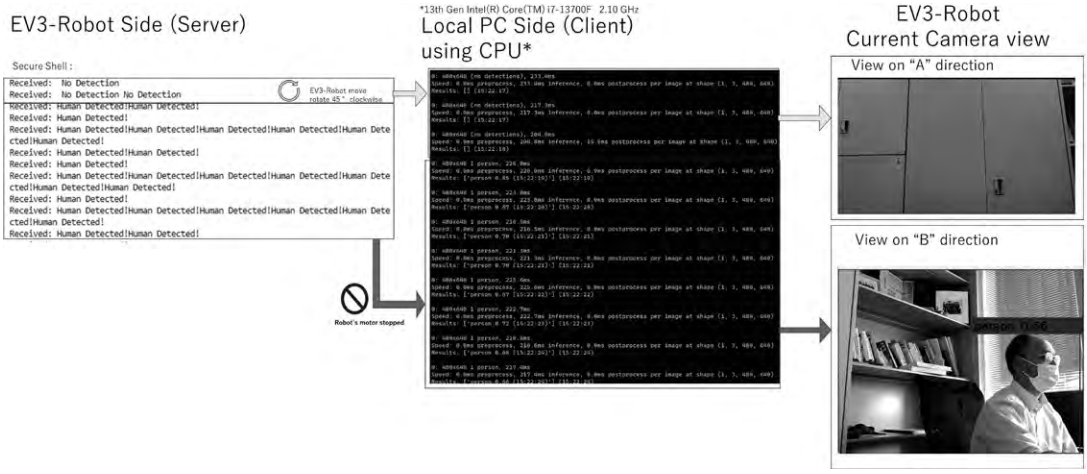


Figure 9. YOLOv8 Real-time Object Detection using CPU-only results.

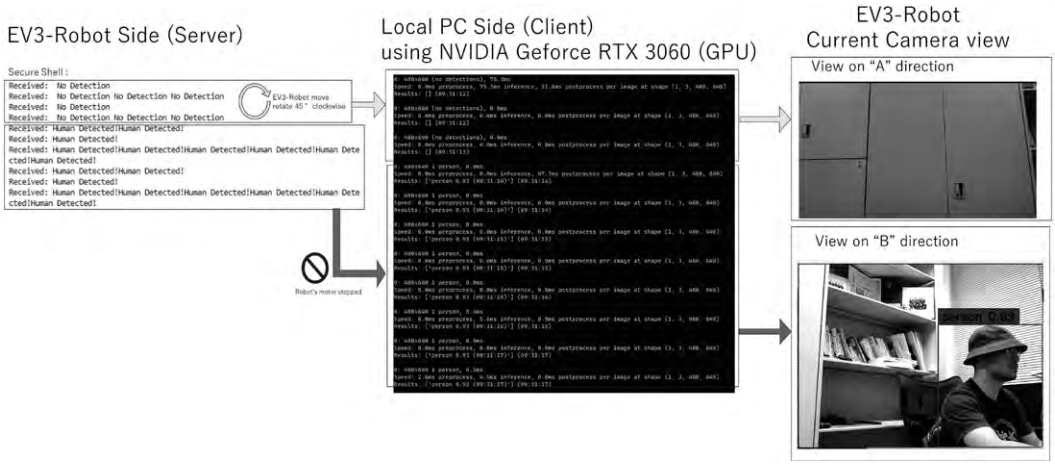


Figure 10. YOLOv8 Real-time Object Detection using CPU+GPU results.

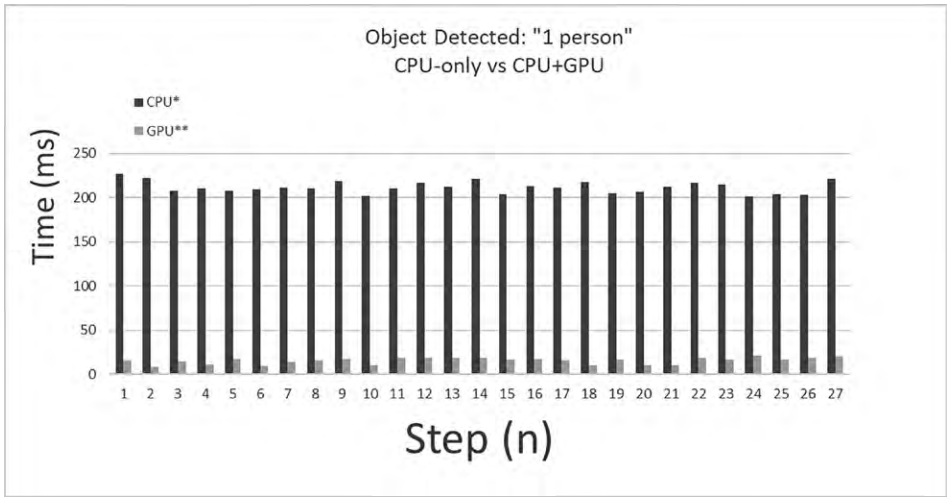


Figure 11. Figure 11. Time Comparison between CPU-only Vs CPU+GPU
*13th Gen Intel(R) Core (TM) i7-13700F 2.10 GHz
**NVIDIA GeForce RTX 3060.

(5) Run the trained YOLOv8 in local PC as a client

Figure 5 lines 31-46 demonstrates the foundational structure of a client setup. In the client code, it is essential to implement procedures for requesting outgoing server responses and generating another request. These actions encompass feedback movement for EV3-robot. Figure 5 line 36 shows the IP address of the EV3-robot for the target connection from the PC (client) to the server (EV-robot).

4 EXPERIMENTAL AND DISCUSSION

In this section, we performed feedback setup movement from EV3-robot, three distinct experiments, and an overall discussion. The first experiment involves a basic Remote-Control Communication. In the second experiment, we implement real-time Object Detection using YOLOv8 with CPU processing. The last one, in the third experiment, we perform real-time Object Detection using YOLOv8 with GPU processing. Finally, we evaluate the overall system.

(1) Experiment setup for feedback

In Figure 6, feedback from detection utilizing the EV3-robot motor functions. Scanning the surrounding environment is necessary for detection. The directions of the EV3-robot rotating towards-to during the real-time object detection, we set to the four directional points. Denoted as “A,” “B,” “C,” and “D,” with every rotation 45° clockwise turn the direction orderly at regular intervals, in this case, we set it to 1 seconds (sec).

Figure 7 shows a sequence representation of the motor rotation process during experiments 2 and 3. The standby mode which is called "scanning person" is created to enable the robot to scan while executing real-time processes of object detection until the desired class which is "person" is detected. Lastly, we set a break-processing step by incorporating an "Esc" command triggered by keyboard inputs to conclude the sequences.

(2) Experiment-1

Figure 8 shows the results of a real-time remote control communication system that utilizes arrow direction keys from the keyboard. We measured the latency using (4). The outcomes of the initial four command keys are displayed in Table 1. Each key is associated with a confirmation message, allowing us to verify when the instruction was issued. Our findings indicate that the latency remains below 1000 ms.

(3) Experiment-2

Figure 9 shows the result of YOLOv8 Real-time Object Detection using CPU-only as a client while connecting with EV3-robot as a server.

Table 2 provides a summary of the findings from experiment 2. In this real-time experiment, the total of Real-time is 4.5 sec within the real-world scenario. The AP of the detection process did not surpass 0.85.

(4) Experiment-3

Figure 10 shows the result of YOLOv8 Real-time Object Detection using CPU+GPU as a client while connecting with EV3-robot as a server.

Table 3 provides a summary of the findings from experiment 3. In this real-time experiment, the total of Real-time is 4.5 sec within the real-world scenario. The AP of the detection process did not surpass 0.93.

(5) Evaluation

(5.1) Latency Consideration

We acknowledge that latency for the current system is still far from the current wireless standard real-time situation[18] which is not more than 0.3 ms. It potentially varies widely depending on several factors, including the network infrastructure, the distance

between the client and server, the complexity of the communication protocol, and the load on the network[19].

On the flip side, it is crucial to acknowledge that the latency in our proposed client-server communication system may influenced by noises from the environments including internal and external factors. The acceptable latency thresholds may differ significantly based on the specific requirements of the application.

For instance, a safety teaching robot is designed to serve as educational material for young learners. In this context, the robot's primary audience is youth, and an important consideration is to minimize the robot's impact on the environment or less harm. As a result, the "standard" latency may exhibit variations depending on the particular context and the use case at hand.

(5.2) YOLOv8 Performances in the Ev3-robot cases: CPU-only Vs CPU+GPU

During performance evaluation, we repeat the same procedure in experiments 2-3 and expand it more.

Figure 11 shows the processing time for results between CPU-only vs CPU+GPU. The CPU+GPU results is overpowering the CPU-only in terms of time process by 20 times faster. Additionally, Tables 2 and 3 indicate it may also the accuracy have higher stability. A factor that processing speed scenarios where real-time performance is critical, a CPU+GPU for EV3-robot may be preferred.

On the flip side, for resource-constrained or cost-sensitive applications, a CPU+EV3-robot still potentially delivers reasonable performance with trade-offs in processing speed.

5. CONCLUSION AND FUTURE WORK

In conclusion, our proposed system provides the practical application of the YOLO CNN for real-time object detection. The approach of using an educational kit EV3-robot as a server and a local personal computer (PC) as a client to provide object detection. Our method demonstrates the communication between the EV3-robot and YOLOs and highlights the real-time object detection achieved. Furthermore, we conduct evaluations of processing times on both the CPU and GPU during real-time object detection while integrating with the EV3-robot's motor. In these findings, our research offers a promising solution that balances the need for efficient object detection in robotics with cost-effective educational platforms. This facilitates the integration of the NN field for young learners. This approach may open doors for future developments in real-time object detection and inspire the next generation of engineers and researchers.

Our future studies aim to extend the scope of our investigations by incorporating additional components, including actuators and sensors within the EV3-robot. We are also looking forward to various applications, such as service robot tasks in the "follow me" scenarios. Furthermore, we intend to integrate these methodologies with motor functions and robot sensors to maintain the distance between the robot and the operator. By creating a more comprehensive and responsive robotic system, we expect to minimize the latency.

We acknowledge that latency for the current system is still far from the current wireless standard real-time situation[18] which is not more than 0.3 ms. It potentially varies widely depending on several factors, including the network infrastructure, the distance between the client and server, the complexity of the communication protocol, and the load on the network[19].

Acknowledgement

This study represents a collaborative joint research effort with partial support JSPS KAKENHI number 23H03468.

References

1) F. Coulmas, *Population decline and ageing in Japan the social consequences*. Routledge, 2007, vol. 16.
2) Y. Yoshimoto and H. Tamukoh, “Fpga implementation of a binarized dual stream convolutional neural network for service robots,” *Journal of Robotics and Mechatronics*, vol. 33, no. 2, pp. 386–399, 2021.
3) S. Hori, Y. Ishida, Y. Kiyama, Y. Tanaka, Y. Kuroda, M. Hisano, Y. Imamura, T. Himaki, Y. Yoshimoto, Y. Aratani *et al*., “Hibikino-musashi@ home 2017 team description paper,” *arXiv preprint arXiv:1711.05457*, 2017.
4) G. Hinton, “Deep learning a technology with the potential to transform health care,” *Jama*, vol. 320, no. 11, pp. 1101–1102, 2018.
5) Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
6) H. Paugam-Moisy and S. M. Bohte, “Computing with spiking neuron networks.” *Handbook of natural computing*, vol. 1, pp. 1–47, 2012.
7) J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779– 788.
8) D. Pramanta, T. Morie, and H. Tamukoh, “Synchronization of pulse-coupled phase oscillators over multi-fpga communication links,” *Journal of Robotics, Networking and Artificial Life*, vol. 4, no. 1, pp. 91–96, 2017.
9) D. Pramanta and H. Tamukoh, “High-speed synchronization of pulse-coupled phase oscillators on multi-fpga,” in *International Conference on Neural Information Processing*. Springer, 2019, pp. 318–329.
10) G. Jocher, A. Chaurasia, and J. Qiu, “Ultralytics yolov8,” 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
11) K. Boyd, K. H. Eng, and C. D. Page, “Area under the precision-recall curve: point estimates and confidence intervals,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23- 27, 2013, Proceedings, Part III 13*. Springer, 2013, pp. 451–466.
12) H. S. Oluwatosin, “Client-server model,” *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 67– 71, 2014.
13) S. Kratky and C. Reichenberger, “Client/server development based on the apple event object model,” 2013.
14) R. V. Aroca, V. P. Torres, L. M. G. Goncalves, A. Negreiros, and A. Burlamaqui, “Cloud based low-cost educational robot,” in *2013 16th International Conference on Advanced Robotics (ICAR)*. IEEE, 2013, pp. 1–6.
15) “Lego education ev3” , 2023. Available online: https://education.lego.com/en-us/downloads/mindstorms-ev3/software, accessed on 30 January 2023. [Online]. Available: https://education.lego.com/en-us/downloads/ mindstorms-ev3/software
16) R. Hempel and D. Lechner, “ev3dev,” 2023. Available online: https://www.ev3dev.org/, accessed on 30 January 2023. [Online]. Available: https://www. ev3dev.org/
17) T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft coco: Common objects in context,” 2015.
18) J. Dean, “Designs, lessons and advice from building large distributed systems,” *Keynote from LADIS*, vol. 1, 2009.
19) S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, “It’s time for low latency,” in *13th Workshop on Hot Topics in Operating Systems (HotOS XIII)*, 2011.