社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

コア/ペリフェリー構造に基づく ネットワーク型複合現実サービスの実装と評価

高木　詩織† 　荒川　伸一† 　村田　正幸†

† 大阪大学 大学院情報科学研究科　〒565–0871 大阪府吹田市山田丘 1–5
E-mail: †{s-takagi,arakawa,murata}@ist.osaka-u.ac.jp

あらまし　近年、IoT の進展に伴い、数多くの新しいネットワークサービスが登場しており、サービスの応答性を向上させるために MEC (Multi-access Edge Computing) の標準化が進められている。MEC 環境にサービスを実装するに際しては、そのため、少ないコストでアプリケーションの振る舞いを柔軟に変更できるサービスの構成を考える必要がある。本研究では、生物システムにおける柔軟な振る舞いをするモデルとして知られる Core/Periphery 構造を導入し、ネットワーク型の複合現実サービスを設計、実装した。Core/Periphery 構造の柔軟性を活かすため、ユーザの要求や実環境に変動があっても変化しない機能をコア機能、ユーザの要求や実環境によって振る舞いが変わり得るものをペリフェリー機能とした。実験により、Core/Periphery 構造に基づいてサービスを設計することで、アプリケーションレベルの遅延の増加を 5.7 [ms] に留めつつ、実装コストの増大を抑制することがわかった。
キーワード　コアペリフェリー構造、マルチアクセスエッジコンピューティング、複合現実、テレイグジスタンスサービス、ネットワークロボット

On the Implementation and Evaluation of a Network-oriented Mixed Reality Service based on Core/Periphery Structure

Shiori TAKAGI†, Shin'ichi ARAKAWA†, and Masayuki MURATA†

† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1–5, Suita, Osaka, 565–0871 Japan
E-mail: †{s-takagi,arakawa,murata}@ist.osaka-u.ac.jp

Abstract　In recent years, many different new network-oriented services has developed, and Multi-access Edge Computing (MEC) is standardized to improve the responsiveness of services. When deploying services in MEC environment, to consider a service structure that can switch service behaviors flexibly to meet various users' demands and can change the service behavior to real-world environment at low implementation cost is needed. In this paper, we introduce a Core/Periphery structure, which is known as a model for a flexible behavior of biological systems, of service components, and design and implement a network-oriented mixed reality service based on the Core/Periphery structure. We investigated what kind of functions can be developed due to users' demands, real environment where devices are placed, and the development of new devices. Then, to utilize the flexibility of the Core/Periphery structure, we regarded functions whose behaviors do not change even when users' demands or environmental changes occurs as core functions, and others as peripheral functions. Our experiment reveals that the implementation cost is reduced while increase of service response time is less than 5.7 [ms]. This result shows that taking advantage of Core/Periphery structure enables to divide service functions appropriately and place the functions in MEC environment appropriately, with a little penalty on latency and with low implementation cost.
Key words　Core/Periphery Structure, Multi-access Edge Computing (MEC), Mixed Reality (MR), Telexistence Service, Network Robots

1. Introduction

In recent years, with development of IoT (Internet of Things), many different new network-oriented services have developed and information network has changed rapidly. In new network-oriented services, we can send information of real world retrieved from cameras and sensors to cloud, or perform high-load processing such as image recognition and/or voice/sound recognition. For example, telexistence services using robots and VR (Virtual Reality) technology or MR (Mixed Reality) technology are now being developed, such as ANA Avatar project [1]. Furthermore, tactile Internet and full-sensory digital reality can be realized by 6G [2].

In these applications, application-level delay is a significant factor for service quality. However, application-level delay significantly increases in cloud computing environment due to communication distance and load concentration [3]. Recently, Multi-access Edge Computing (MEC) [3]〜[5] is standardized to relax increase of the application-level delay for delay-sensitive services. In MEC environment, computing resources and storage are allocated at the edge of the network, so that processing end devices require is performed at the place closer to the end devices. This leads to improvement of application responsiveness by shortening communication distance and load distribution.

When deploying network services in MEC environment, to consider a service structure that can change behaviors of services in a flexible manner with low cost is needed. If developers reconstruct whole services to meet different users' demands or to adapt to the environmental variation such as device evolution, implementation cost increases. Moreover, resources in the MEC environment are limited by spatial restrictions, and not necessarily same as those in cloud computing environment. It is difficult to locate all-possible services in advance such that services on edge can adapt to each users' demand and environmental variation.

Our research group has been investigated a Core/Periphery structure [6],[7] of service components to effectively adapt to each user's demand and environmental variation. Core/Periphery structure is a model for a flexible and efficient information processing mechanism in biological systems. Information processing units with Core/Periphery structure is classified as Core or Periphery. Core is composed densely with system constraint, and process information more efficiently, whereas the Periphery, which is connected with Core, can have various configurations, flexibly adapts to environmental changes surrounding the system, and builds flexible and efficient information processing mechanisms with Core. The advantage of Core/Periphery structure on accommodating information services, represented by chains of functions, is numerically investigated in [8], and the results show that Core/Periphery structure leads to less developmental costs for accommodating various kind of information services.

In this paper, based on the advantage of Core/Periphery structure, we aim to realize a service system which adapts service behaviors to users' various demands, environmental changes such as real environment where the end devices are located, or various devices. Unlike the model-based evaluation of [8], we implement the shopping service using mixed reality (MR) devices and robots, using actual devices, and evaluate the effect of designing services based on Core/Periphery structure by experiment. First, in our supposed service, we investigate what kind of functions can arise due to users' demands, real environment where devices are placed, and the development of new devices. To utilize the flexibility of the Core/Periphery structure, we regard functions whose behaviors do not change when users' demands or environmental changes occurs as core functions, and functions whose behaviors can be changed due to users' demands or environmental changes as peripheral functions.

Our experiment proved that the implementation cost is reduced without increasing the service response time compared to the case where the service functions are not divided, because core functions deployed on edge servers cooperate with peripheral functions deployed on end devices.

The remainder of this paper is organized as follows. Section 2 explains the service targeted in this paper and the service design based on Core/Periphery structure. Section 3 explains the details of the service implementation and evaluation. Finally, Section 4 explains the conclusions and future works.

2. Service Design based on Core/Periphery Structure

2.1 Network-oriented Mixed Reality Services under Study

The supposed service is a shopping service using MR and robots. Robots are places in real stores, and users enjoy shopping as if they were in the stores though they are actually at home. Robots take video of the real store while moving as per instructions of users. Real-world information on store-side is attached on video and sent to users. Users can move robots with controllers, gestures, or their gaze. Figure 1 shows the whole service image and service functions.

Robot-side application requires functions for moving, taking video, processing images, collecting and aggregating information around the robot, and adjusting moving speed so as not to hit people or objects. User-side application requires functions for displaying video, messaging instruction to robots, collecting and aggregating information around users, and detecting object based on the granularity users want.

For video processing and live streaming system, we suppose that users' demands are, for example. to watch real-time video, to watch high resolution video, or to adopt high accuracy object detection methods. In addition, we suppose that demands from people who place robots in stores, or who provide video are, for example, to distribute video on a large scale, to send video to a single user, and limit bit rate per user of video.

To meet the above demands, the video system provides functions

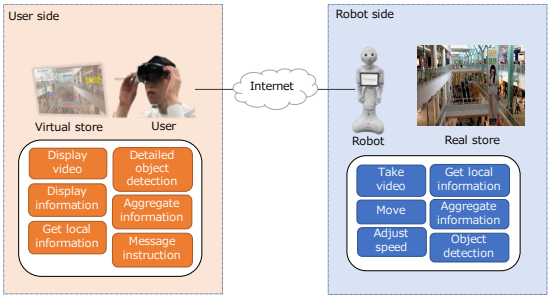


Figure 1: Supposed service and functions.

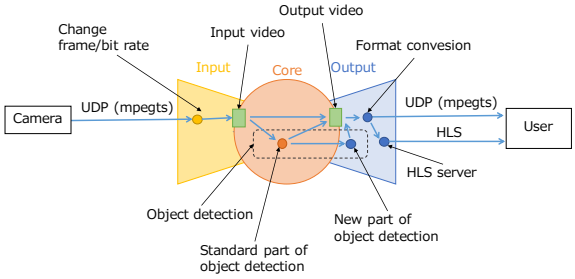


Figure 2: Video system based on Core/Periphery structure.

Table 1: Examples of demands and functions in video system.

Function	Users' demand	Behavior variation
Video I/O	Real time video	Change rates
	High-resolution video	Change resolution
Object detection	Fast and standard	Select method
	New but slow	
Distribute video	To one user	With UDP
	On a large scale	With HTTP

Table 2: Examples of demands and functions in robot operation system.

Function	Demands/Real world	Behavior variation
Get users' action	Use gestures	Select interface
	Use controllers	
	Use gazes	
Access to APIs	Operate robots	Switch APIs
	Operate drones	
Adjust robot speed	No obstacles	Move robot speedily
	Some obstacles	Move robot slowly

for capturing and outputting video, for performing object detection, and for distributing video to users. Table 1 shows correspondence of uses' demand to functions in the video system.

In the robot operation system, we suppose that users' demands are, for example, to select which robot to access, to change robot speed, to move robots' arms, to select how to operate robots by users' gestures or controllers, and to follow users' gaze to change the direction of robots. Service behavior can be changed due to variation of real-world environment, such as communication quality, obstacles, or crowd of people. Furthermore, new devices, such as new controllers, new robots or drones, can be used.

To meet the above demands, the robot operation system provides functions for recognizing users' instruction such as gestures, gazes, and controller status, for sending message from users, for accessing APIs, for adjusting robot speed in order to avoid obstacles, for collecting and aggregating information obtained from robots.

Table 2 shows correspondence of uses' demand/real-world environment to functions in robot operation system.

2.2 Function Placements based on Core/Periphery Structure

This section explains function placements based on Core/Periphery structure in video processing and live streaming system, and robot operation system, respectively. To place service functions appropriately, we considered which functions explained in Section 2. 1 are core functions and which functions are peripheral functions, based on the concept of Core/Periphery structure that Core processes information more efficiently, and Periphery has various configurations and flexibly adapts to environmental changes surrounding the system.

2.2.1 Video Processing and Live Streaming

Among functions shown in Section 2. 1, capturing and outputting video, and standardized method of object detection are common functions, and they are core functions.

Figure 2 shows Core/Periphery structure of video system. Orange-colored field represents Core. Light orange-colored field represents Periphery on user side and blue-colored fields represents Periphery on robot side. Video is sent from camera, and its frame/bit rate is adjusted based on providers' demands as peripheral function. Then, the video passes through core function including functions for inputting video, outputting video, and standard part of object detection. Finally, the video format and distribution protocol are selected and sent to users.

2.2.2 Robot Operation

Figure 3 shows Core/Periphery structure of robot operation system. The function to send instruction messages from users and aggregating information obtained from robots are common, so they are core functions. Functions that can change to adapt user demands and changes in the real environment, such as how to input users' instructions are peripheral functions. In addition, functions to access the API of robots, functions to collect information such as current robot position, and functions to adjust the speed are peripheral function because they change due to the type of devices and real-world environment where devices are placed.

2.3 Service Scenarios

This section explains service scenarios and function placements in MEC environment.

2.3.1 Behavior based on the Real Environment on the Robot Side

We explain a scenario in which behavior based on real environment where robots are placed. Core functions are functions for

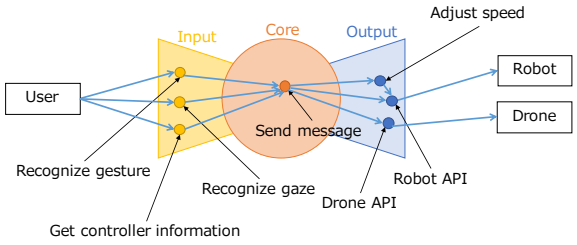


Figure 3: Robot operation system based on Core/Periphery structure.

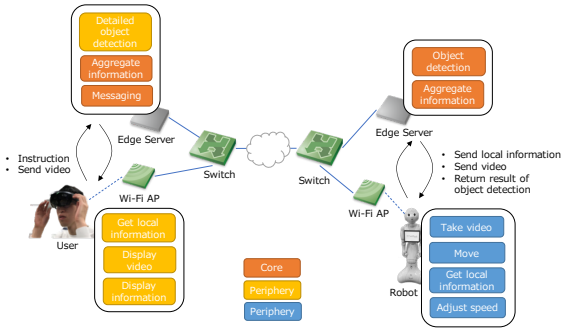


Figure 4: Scenario: behavior based on the real environment on the robot side.

transmitting instructions from the user to the robot and functions for object detection. Peripheral functions are functions to get information near the robot, functions to adjust the movement speed of the robot, and functions to aggregate information sent from multiple robots.

Figure 4 shows this scenario. There are users with MR headsets, robots, cameras, and edge servers. Orange-colored functions are core functions. Light blue-colored functions are peripheral functions on robot side, and light orange-colored functions are peripheral functions on user side.

Users send their instruction to robot, and move bodies and heads of the robots by controllers, gestures, and their gaze. In addition, video captured by the camera on robots are sent to the edge servers. Edge servers perform object detection on the video and recognize objects and persons around robots. Object detection function, one of core functions, needs to be performed in real time, and required high specification servers and thus should be deployed on edge servers. Results of the object detection is returned to robots. When there are many people around them the robot reduce its speed in order to avoid collision. Moreover, information obtained from robots can be aggregated in the edge server and shared with other robots. Information sharing enables users to avoid other robots while operating their robots.

2.3.2 Behavior based on the Real Environment on the User Side

We explain a scenario in which behavior based on real environment on user side. Core functions are functions for instructions from the user to robots and function to aggregate information of multiple

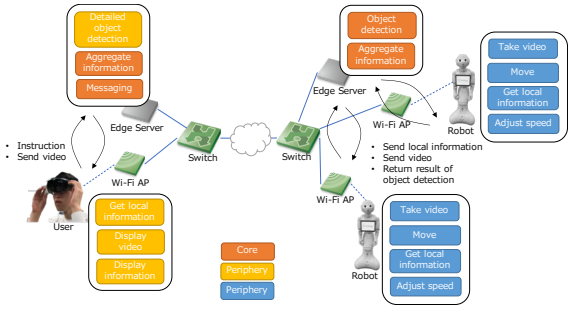


Figure 5: Scenario: behavior based on the real environment on the user side

robots. Peripheral functions are functions for displaying video, detailed information of objects, information about each robot.

Figure 5 shows this scenario. Orange-colored functions are core functions. Light blue-colored functions are peripheral functions on robot side, and light orange-colored functions are peripheral functions on user side. Information of stores and robots such as product information or communication status is collected at the edge severs on user side. Users select which robot to operate only by communicating with their own edge servers while seeing the aggregated information of the stores and robots.

As a core function, the video sent from cameras is roughly classified into each object type on edge servers on the robot side. Then, as a peripheral function, detailed object detection is performed on the edge servers or users' devices. Users' devices collect personal information such as the users' tastes, information about what the user has, and purchase history. Using the personal information, contents which users want to know are displayed. For example, if purchase history has a commodity that is regularly purchased, the application recommends the commodity based on the history, or the expiration date of a food in users' home is displayed.

3. Implementation and Evaluation of the Service based on Core/Periphery Structure

This section explains implementation detail and evaluation of the service based on Section 2. 3.

3.1 Implementation of the Service based on Core/Periphery Structure

3.1.1 Video Processing and Live Streaming

Video from cameras is sent to the edge server on robot side and captured with OpenCV [9]. Then, object detection with YOLOv3 [10] and PyTorch, a library for deep learning is performed. Mask R-CNN (Region-based Convolutional Neural Networks) [11], an algorithm that not only surrounds certain areas where objects are detected with a rectangle but also recognizes the type of object for each pixel and colors it, can be used. The processed video is transmitted to the HoloLens [12], MR headset worn by users using ffmpeg [13] with UDP and displayed.

3.1.2 Robot Operation

We used Xbox controller, which can connect to HoloLens, as the users' controller. Information of the controller by HoloLens is transmitted with MQTT (Message Queuing Telemetry Transport), which is a publish/subscribe type protocol. We built an MQTT messaging system on the edge server on user side with mosquitto [14], an open source message broker, and Node-RED [15], a programming tool for event-driven applications. The MQTT broker gets messages from HoloLens and send it to Choregraphe, the programming tool for Pepper [16], robot. A program on Choregraphe accesses to Pepper API when it gets messages from MQTT broker.

Pepper gets lists of object names from the edge server which performs object detection, and when there is people, it reduces its speed. Furthermore, Pepper has a map of an area where it can move, and get its position regularly. Each Pepper's position is aggregated in the edge server and shared with the other Pepper. Pepper's position is also plotted on map, sent to users' HoloLens, and displayed on HoloLens.

3.2 Evaluation and Experiment Settings

3.2.1 Implementation Cost

With the implemented service, we show that the implementation cost is reduced by adopting to the Core/Periphery structure. We compare amount of the source code for sending users' instruction when functions are divided to core and peripheral functions with when whole service is implemented on an end device to evaluate the implementation cost.

3.2.2 Responsiveness of Service

Since application-level delay may increase when users' instructions are sent via the edge server, compared with when they are sent directly to robots, we measured and evaluated application-level delay as penalty of using edge server.

We sent messages about 50 times regularly from the HoloLens application and save each time when it sent messages as t_1, t_2, ..., t_n, where n is number of inputs. On the robot side, a digital clock that can display milliseconds is displayed on a laptop nearby Pepper. We took a video and made records of each time when Pepper started moving as t'_1, t'_2, ..., t'_n. Then, we calculated average of $t'_1 - t_1$, $t'_2 - t_2$, ..., $t'_n - t_n$ as the absolute time from each input to the start of Pepper with time difference between devices. Since the clocks of HoloLens and Pepper are not synchronized, the absolute time from each input to the start of Pepper cannot be measured. Therefore, we calculated the penalty of using the edge server by subtraction these two average times.

Furthermore, we measured the improvement of service responsiveness by placing core functions on edge servers. When allocating core functions and peripheral functions respectively, core function can be deployed on the cloud or on the edge server, and the peripheral function can be deployed on the edge server or on the end devices. Core functions perform real-time image processing or aggregation of information, and cannot be executed on end devices. To improve responsiveness, core functions should be placed on the edge server instead of the cloud.

We constructed MEC environment using OpenStack and Amazon Web Service (AWS). The AWS cloud is in Ohio, and connected to OpensStack environment via VPN. Then, we measured the time from when a user inputs an instruction to when the robot starts moving in MEC environment and in cloud environment, respectively, and compared the time and evaluate the effect of allocating core functions on the edge server. For each input, Pepper saved each time when it started moving. Then, we calculated average the absolute time from each input to the start of Pepper with time difference between devices. Finally, the application-level delay in cloud environment by subtraction these two average times.

3.3 Results

3.3.1 Implementation Cost

Figure 6 shows the relation between the number of device types and the number of lines of source code. Both Figure (a), the connection establishment part, and Figure (b), the messaging part, show that designing services based on Core/Periphery structure is effective when the number of device types.

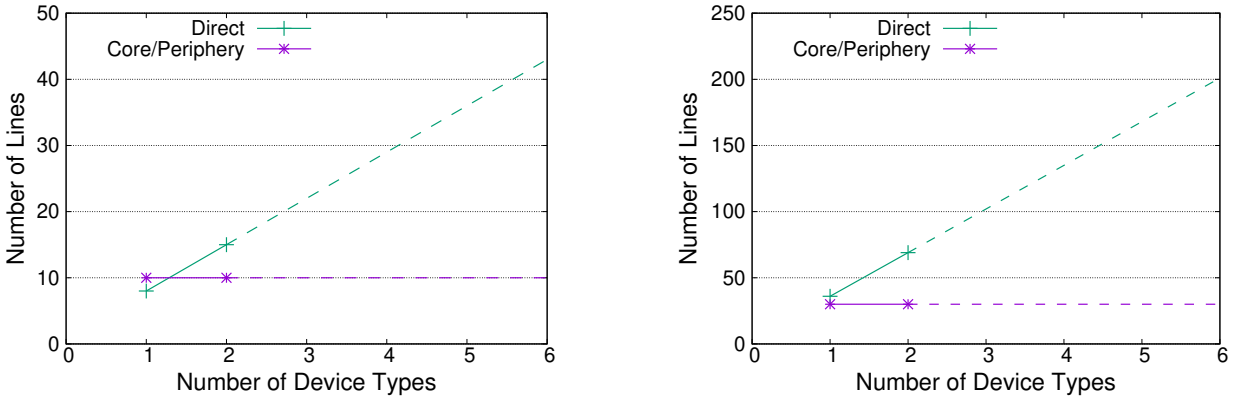
Compared to the case where the functions are not divided, in case the functions are divided and MQTT is used, developers do not need to change nor add source code for accessing each device APIs to establish connections, to disconnect, and to move devices in remote places, and settings of relationship between information of controller and movement distance of devices when the type of devices increases. The case when devices on user side establish connections directly with devices in remote device, developers cannot write many parts of source code unless they know APIs of different devices. Not only variation of device in remote places, but also variation of devices on user side. In both services based on Core/Periphery and services not based on Core/Periphery structure, developers have to add source code to get information of controller, because this function is peripheral. However, the more types of controllers, the more parts of source code to add when the types of devices on the remote side increase, effect obtained by designing services based on Core/Periphery structure. Therefore, developers can implement applications more easily by adopt Core/Periphery structure.

3.3.2 Responsiveness of the Service

Figure 7 shows the difference of application-level delay between the case the HoloLens application directly connects the robot Pepper, and the case of the HoloLens application connects Pepper via the edge server. The delay due to MQTT on the edge server was 5.7 [ms].

Difference of application-level delay between using the edge server and using the cloud was 474.3 [ms]. The difference of RTT between PC-to-Edge and PC-to-Cloud is 140 [ms], and the difference of application-level delay is about 3 times larger. Therefore, deploying core functions on edge servers is significant.

Combining with the results shown in Section3. 3. 1, the service



(a) Connection establishment part

(b) Messaging part

Figure 6: Number of lines of source code.

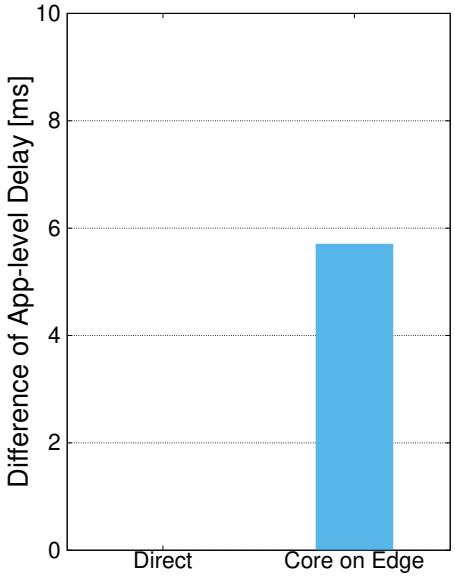


Figure 7: Difference of application-level delay.

design based on Core/Periphery structure enables to reduce the implementation cost without deteriorating responsiveness of the service much.

4. Conclusion

In this paper, we introduced a Core/Periphery structure, and designed and implemented a network-oriented mixed reality service based on the Core/Periphery structure. Furthermore, we evaluated the effect of the design based on Core/Periphery structure by our experiment environment in our laboratory. Our experiment revealed that the application-level delay due to the edge server was 5.7 [ms]. Taking advantage of Core/Periphery structure enables to divide service functions appropriately and place the functions in MEC environment appropriately, so that implementation cost is reduced with little penalty.

As future works, we will evaluate the implementation cost and for object detection and feedback to robots, for sharing information among robots. Furthermore, implementation and evaluation of the service using different devices such as robots rather than Pepper is needed.

Acknowledgment A part of this work was supported by National Institute of Information and Communications Technology (NICT) in Japan.

References

[1] "ANA Avatar." https://ana-avatar.com.

[2] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, pp. 28–41, Sept. 2019.

[3] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 2359–2391, June 2017.

[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5G," *ETSI White Paper*, Sept. 2015.

[5] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 1657–1681, May 2017.

[6] P. Csermely, A. London, L.-Y. Wu, and B. Uzzi, "Structure and dynamics of core-periphery networks," *Journal of Complex Networks*, vol. 1, pp. 93–123, Sept. 2013.

[7] V. Miele, R. Ramos-Jiliberto, and D. P. Vázquez, "Core–periphery dynamics in a plant–pollinator network," *bioRxiv*, July 2019.

[8] Y. Tsukui, "On network function virtualization for dynamically changing service requests based on a core/periphery structure," Master's thesis, Graduate School of Information Science and Technology, Osaka University, Feb. 2020.

[9] "OpenCV." https://opencv.org.

[10] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *CoRR*, vol. abs/1804.02767, Apr. 2018.

[11] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of 2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988, Oct. 2017.

[12] "Microsoft HoloLens." https://www.microsoft.com/ja-jp/hololens.

[13] "FFmpeg." https://www.ffmpeg.org/.

[14] "Eclipse Mosquitto." https://mosquitto.org.

[15] "Node-RED." https://nodered.org.

[16] "Pepper the humanoid robot - SoftBank Robotics." https://www.softbankrobotics.com/emea/en/pepper.