Communication Framework for 5G-Enabled Network Applications

Zdeněk Materna[1], Michal Kapinus[1], Petr Dobeš[1], Roman Juránek[2] and Pavel Smrž[1,2]
[1] Brno University of Technology, Brno, CZ
[2] Cognitechna
Email: {imaterna,ikapinus,idobes,smrz}@fit.vut.cz

***Abstract*—This paper presents the framework for easily deployable, universal network applications focusing on robotic tasks. The framework defines the unifying interface between the robot and the application. It could be deployed in three locations: on the cloud, edge device, or on the robot itself. This paper describes the interface in detail, followed by the reference implementation of such a network application for an object detection task. The reference implementation is evaluated under different conditions, locations, and connection methods. Results indicate that offloading computationally demanding tasks to the edge or cloud is feasible and significantly reduces power demands on the robot as CPU and RAM are less utilized. Moreover, a GPU, needed for acceleration of detection algorithms, might be omitted from the robot's configuration.**

I. Introduction

To allow autonomous operation, robots must be able to sense their environment and learn, which requires significant computing power, leading to higher energy demands and, therefore, lower battery life. To perform tasks autonomously in a semi-structured or unstructured environment, a robot may need, for instance, an object detection module, which depends on a continuous stream of sensor data, specifically camera images. Typically, this has to be run locally on the robot because of insufficient bandwidth, latency, and reliability of networking technologies such as Wi-Fi.

With the increasing availability of reliable and low-latency 5G networks, it is achievable to offload the most demanding computational tasks to another machine over the network, which could be edge or cloud, depending on actual requirements and limitations. Moreover, having a shared cognition module in a cloud might enable collective intelligence, where data from multiple robots are aggregated. Another reason for offloading algorithms might be that specialized hardware, such as an AI accelerator or GPU, might be required for optimal performance, which is unavailable on the robot.

This work presents an approach for cloud-native network applications and is part of broader efforts within the 5G-ERA Project [1–3]. The project aims to provide a complete infrastructure for the design, development, deployment, and provisioning of network applications to improve the autonomy of robots and the quality of experience for customers of vertical sectors such as PPDR[1], transport, healthcare, and Industry 4.0. Within this paper, we deal with a subset of the project's scope – namely, we propose guidelines for the architecture of network applications and define a common interface for stateless and stateful applications. A reference network application, interface definition, and a universal client for Python (currently the most widely used scripting language) are publicly provided. To evaluate the performance of the reference application, an experiment was carried out, comparing multiple options for image transport and deployment directly on the robot, on edge, and in a cloud.

[1]Public Protection & Disaster Relief

II. Related work

For mobile robots, the first generation of ROS [4] used to be the de-facto standard. Its custom communication protocol based on XML-RPC offers TCP and, to a limited extent, UDP transports; however, not supported in Python implementation and without multicast. One of the disadvantages is that it needs a central node, which facilitates connections between other nodes. ROS was intended to be used over a local, reliable network. There were various attempts to involve ROS-based robots in cloud architecture, either using a custom protocol [5] or based on VPN [6]. It is also possible to containerize ROS nodes [7], which is one of the steps toward cloud-native applications. The second generation, ROS2 [8], is based on DDS, supports QoS and multicast, and uses a discovery mechanism to facilitate connections between nodes, which removes the need for the central node. However, ROS2 on its own is not production-ready for the flexible deployment of cloud-native applications. For instance, FogROS2 [9] is an extension of ROS2, utilizing Kubernetes to deploy computational modules and h264 stream for image transport. Robot and cloud nodes are connected through VPN. It was shown it could reduce SLAM latency by 50 %. However, not all robots use ROS, or, in some cases, DDS multicast-based communication might be infeasible due to the security settings of a cloud provider, so it also makes sense to create ROS-agnostic frameworks. An example could be Kube5G [10], which uses snap packages or Docker containers and can deploy them on a bare metal or cloud. The issue that must also be considered when offloading computational tasks to the cloud is necessary bandwidth, latency, and reliability. With 5G networks, achieving a reliable deterministic performance is possible, which is unattainable with Wi-Fi (IEEE 802.11ac or even the latest ax variant) [11].

III. Proposed network application interface

We propose using network applications with a specific design to offload data processing from the robot to the external computational node. The edge device, i.e., geographically close hardware, could represent the computational node, offering

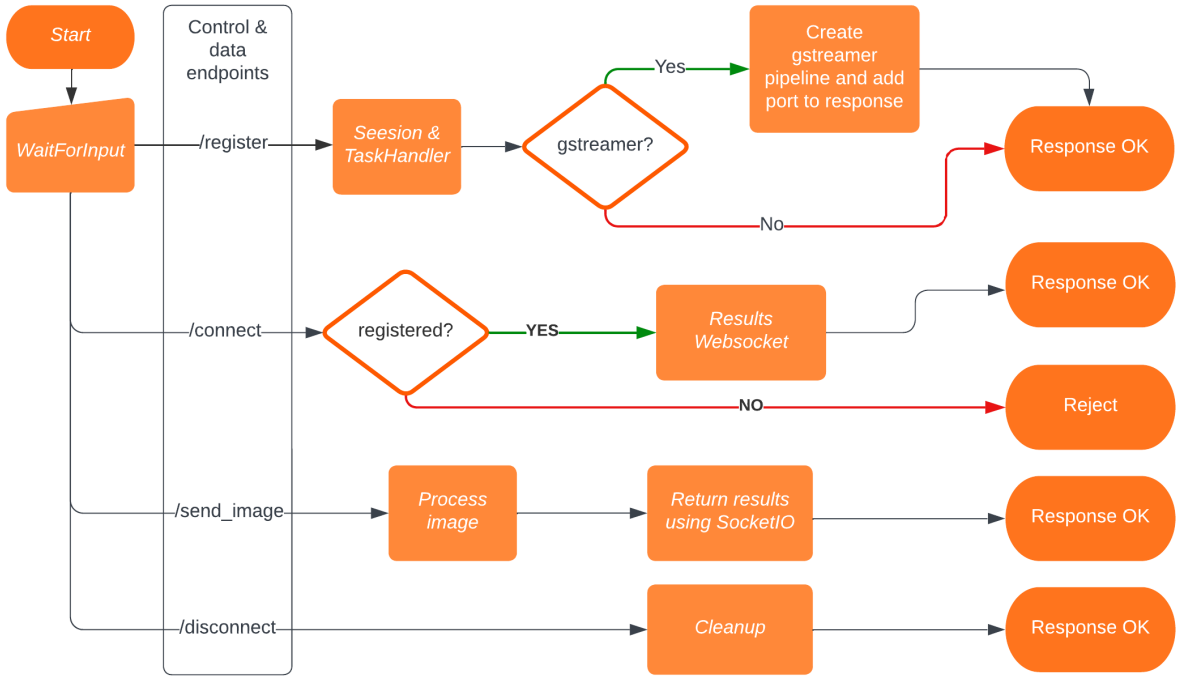

Fig. 1. The life-cycle of the network application. It describes the control endpoints required for establishing communication between the robot and the network application and the */send_image* endpoint used for sending the images to the network application.

a low-latency connection with potentially high throughput. Applications where latency is not crucial could also use cloud computing for the offloading, especially if an edge device is unavailable. The cloud-based solution provides higher scalability and a more efficient resource allocation as they can be shared between multiple robots.

The network application can handle stateless and stateful services, although both have limitations and advantages. Implementing the stateless services as a network application with high scalability and great performance is relatively easy because no data model must be updated over time and shared between all computing nodes. Scalability is more challenging for stateful services because the time and model synchronization between all computing nodes must be considered.

For deploying the network application in a real-world scenario, orchestration is a crucial component. In the scope of the 5G-ERA project, a complex middleware is being developed[2], which provides the PaaS (platform as a service) solution for the network application. Therefore, the network application does not have to take care of the orchestration procedure because the 5G-ERA Middleware will deploy the application to suitable HW based on the robot's requirements and enables the robot to communicate with the application directly. The proposed interface supports containerization, and the resulting network application could therefore be orchestrated by the 5G-ERA Middleware in the Kubernetes cluster.

[2]https://github.com/5G-ERA/middleware

A. Architecture and life-cycle

The proposed architecture defines a protocol for connecting the robot (or any other type of client) and the network application. The reference implementation utilizes a combination of HTTP and WebSocket communication to send data to the network application and obtain results from it. However, the protocol is general and enables the utilization of various data transmission channels, such as GStreamer, for video stream transmission or a message broker, such as Apache Kafka or RabbitMQ.

The network application's life-cycle is presented on fig. 1. To establish the connection between the robot and the application, the robot uses the HTTP endpoint `/register`, which ensures that the corresponding record of the robot is created on the network application side. After the record and corresponding session is made, the client can optionally initiate the bi-directional WebSocket (SocketIO) connection to the network application to pass data and obtain results asynchronously, without polling. This pattern enables to create two types of connections:

- Unidirectional HTTP connection.
- Bidirectional HTTP+SocketIO connection.

The former uses the HTTP requests only for passing data and obtaining results, resulting in a more straightforward integration of virtually any robot capable of creating HTTP requests. It could connect simple robots, IoT or mobile devices, or others. Since the connection is unidirectional (where the robot is the client and the network application is the server), the data from the network application could be passed to the

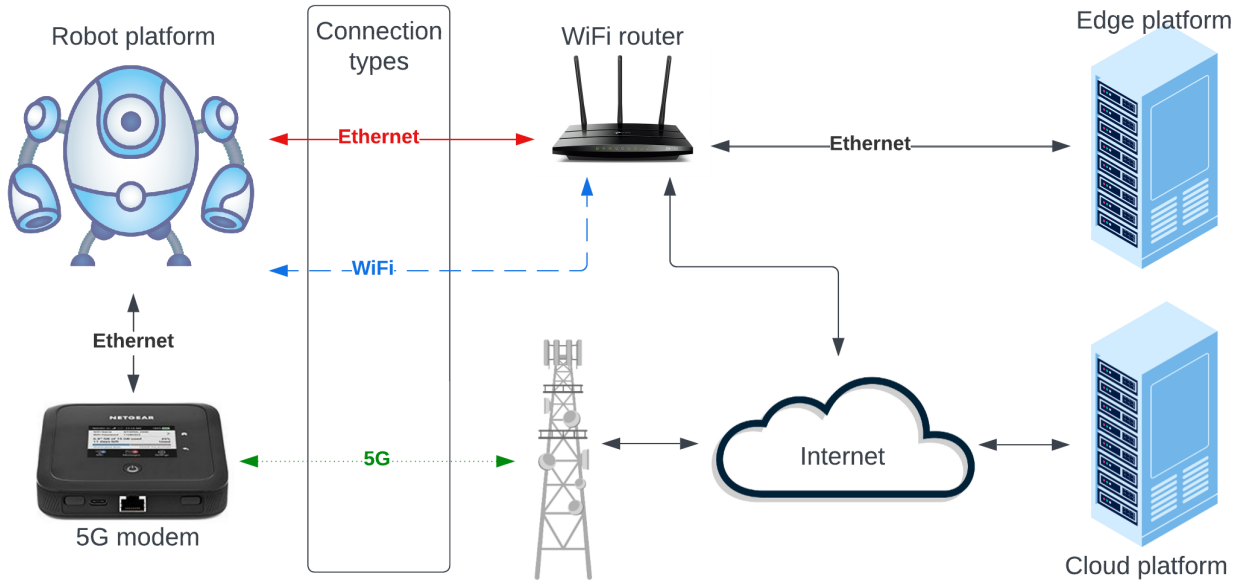

Fig. 2. The network topology used for the evaluation with all three connection types - Ethernet (solid red line), Wi-Fi (blue dashed line), and 5G (green dotted line). Only one of the connection types was enabled for each experimental run. The figure contains images published by gstudioimagen, pch.vector and upklyak on Freepik.

robot only when the robot explicitly asks for them, e.g., using the polling mechanism. The latter combines the simplicity of HTTP requests with the versatility of WebSockets through the SocketIO library. Since the WebSockets are bidirectional, both parties could initiate data transmission anytime.

This approach enables multi-domain communication between the robot and the network application. The communication can be initialized from behind the NAT or firewall and can benefit from standard HTTP authentication and authorization.

B. Interface and client library

The client library for connecting the robot and the network application is publicly available as `era-5g-client`[3] package. It is written in Python 3 and simplifies the communication protocol integration into the robot code. The client library is compatible with ROS Noetic, Foxy, Galactic, and Humble. It consists of two main classes, one of which contains methods for communication with the network application. The other one serves as an interface between the robot and the 5G-ERA Middleware. It allows deployment of the requested network application and controls its life cycle.

The `era-5g-interface`[4] contains several classes which simplify the implementation of the network applications.

IV. Reference Implementation

For the reference implementation of the network application[5], we have selected the object detection problem, as it is a vital problem for almost every kind of robot to be able to handle complex tasks. Mobile robots need to detect other robots, obstacles, or pedestrians. An industrial robot needs the detection of workpieces, tools, or human workers. Assistive robots need object detection to, e.g., pick the correct food from the fridge or search for items in the house.

We have selected the YOLO [12] object detection method provided by the MMDetection package [13] due to its easy integration and high performance. The detector is stateless and therefore requires no temporal knowledge of the processed video stream. We use the DarkNet-53 [12] version of the detection model for reference implementation.

Nevertheless, the MMDetection package also offers various other pre-trained detector models, including neural networks performing instance segmentation. As a result, the implementation of the reference network application is not limited to using only the YOLO detector. Any other object detection model provided in the MMDetection package can be readily selected instead of the YOLO model simply by changing the initialization parameters of the network application. Because of this, the reference application can be used with minimal changes in various use cases. Even when the use case differs significantly, it acts as a template and reduces necessary development effort significantly.

V. Evaluation

We have evaluated the reference network application concerning the power requirements, observed latency and CPU, GPU, RAM, and video RAM usage under various conditions. All the metrics were measured using three different execution platforms:

- Robot - Intel Core i5-6500 CPU, 4 cores @ 3.20GHz with 32 GB RAM and Nvidia GeForce GTX 1050 Ti with 4 GB RAM, running Ubuntu 22.04 operating system

[3]https://github.com/5G-ERA/era-5g-client
[4]https://github.com/5G-ERA/era-5g-interface
[5]https://github.com/5G-ERA/Reference-NetApp/

	Power [W]			CPU [%]			RAM [%]			GPU [%]			VRAM [%]			Latency [ms]			RPS		
	H	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**
Robot (direct)	-	88	-	-	25	-	-	10	-	-	39	-	-	26	-	-	148	-	-	10.0	-
Robot	96	97	95	32	28	25	11	11	11	39	40	43	26	26	26	152	155	182	10.0	10.0	10.0
Ethernet/Edge	49	50	48	33	33	32	4	4	4	0	0	0	0	0	0	152	158	164	9.8	10.0	10.0
Wi-Fi/Edge	49	49	49	32	33	32	5	5	5	0	0	0	0	0	0	156	165	1668	9.4	9.9	10.0
5G/Cloud	47	49	57	29	33	57	4	4	4	0	0	0	0	0	0	225	204	512	5.5	9.7	9.9
Ethernet/Cloud	38	38	37	26	26	23	4	4	4	0	0	0	0	0	0	149	175	575	10.0	10.0	10.0
Wi-Fi/Cloud	38	38	48	28	25	32	4	4	4	0	0	0	0	0	0	154	183	385	8.6	10.0	10.0

TABLE I. THE RESULTS FOR 10 FPS INPUT VIDEO. ALL VALUES ARE MEDIAN FOR 60 SECONDS OF DATA. THE RPS REPRESENTS RESULTS PER SECOND, I.E., HOW MANY FRAMES WERE PROCESSED EACH SECOND. THE H, W, AND G DENOTE THE USED IMAGE TRANSPORT METHOD, I.E., HTTP, WEBSOCKETS (SOCKETIO), AND GSTREAMER.

	Power [W]			CPU [%]			RAM [%]			GPU [%]			VRAM [%]			Latency [ms]			RPS		
	H	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**	**H**	**W**	**G**
Robot (direct)	-	130	-	-	46	-	-	10	-	-	87	-	-	26	-	-	113	-	-	21.6	-
Robot	134	133	132	55	54	51	11	11	11	79	79	86	26	26	26	125	127	127	19.6	19.7	21.3
Ethernet/Edge	52	52	52	42	42	38	4	4	4	0	0	0	0	0	0	126	129	138	17.5	18.3	20.9
Wi-Fi/Edge	51	52	51	40	41	40	5	5	5	0	0	0	0	0	0	132	142	141	15	16.4	18.9
5G/Cloud	57	51	59	29	41	63	4	4	4	0	0	0	0	0	0	236	182	272	5.2	17.2	21.1
Ethernet/Cloud	38	41	39	40	33	63	4	4	4	0	0	0	0	0	0	140	144	148	11.6	19.3	21.3
Wi-Fi/Cloud	48	40	50	34	33	37	4	4	4	0	0	0	0	0	0	130	145	166	9.9	17.5	21.1

TABLE II. THE RESULTS FOR 25 FPS INPUT VIDEO. ALL VALUES ARE MEDIAN FOR 60 SECONDS OF DATA. THE RPS REPRESENTS RESULTS PER SECOND, I.E., HOW MANY FRAMES WERE PROCESSED EACH SECOND. THE H, W, AND G DENOTE THE USED IMAGE TRANSPORT METHOD, I.E., HTTP, WEBSOCKETS (SOCKETIO), AND GSTREAMER.

- Edge - AMD Ryzen 7 1700X, 8 cores @ 3,4 GHz with 32 GB RAM and Nvidia GeForce GTX 1050 Ti with 4 GB RAM, running Ubuntu 20.04 operating system
- Cloud - AMD Ryzen 7 1700X, 8 cores @ 3,4 GHz with 32 GB RAM and Nvidia GeForce GTX 1050 Ti with 4 GB RAM, running Ubuntu 20.04 operating system

The robot and the edge computers were two different computers with similar specifications. The cloud platform was a computer with exactly the same specifications as the edge platform. We have utilized three different types of network connectivity for the evaluation. The Ethernet and the WiFi were utilized for connection to both edge and cloud platforms. In contrast, the 5G connection was only used for connection to the cloud platform because we used the services of the commercial operator without access to their infrastructure or dedicated slices, so the edge could not be accessed directly. The Ethernet connection was made using a standard Gbit Ethernet line with a SOHO switch. The Wi-Fi connection utilizes the USB-connected IEEE 802.11ac dongle and SOHO Wi-Fi router. To enable the 5G connectivity, the Netgear MR5200 5G modem, connected to the robot platform via Ethernet, was used. The modem was connected to the Sub-6GHz (n1 2100 MHz) network with PLMN code 23003. The network topology used for the evaluation is presented in fig. 2. Power consumption was measured using Shelly 1 PM, and data were obtained using a local REST API at 1 Hz. All other measurements were made at 10 Hz. The input video, obtained from a real use case of a project partner, was 60 seconds long, with h264 encoding. There were two variants of the video – 10 FPS and 25 FPS. All measurements were done for both of them.

The robot and the edge were located in the same room and within the same network subnet (for Ethernet and Wi-Fi connectivity). The cloud platform was located in a different location, reachable only through the public internet. The evaluation was carried on with three different methods of image data transfer: sending the JPEG-compressed images to the HTTP endpoint, sending the JPEG-compressed and base64 encoded image using the SocketIO library, and sending the h264 video stream using the GStreamer (denoted as H, W, and G respectively, in the tables I and II). All methods were tested while the network application was deployed on each platform, including the robot. Besides, on the robot platform, one more test was carried out. The detection algorithm was executed directly without the proposed framework to identify the overhead of the network application interface.

The results, as shown in Table I and Table II, clearly indicate that offloading makes sense for lowering power consumption on the robot. The power is lower in all cases when the algorithm does not run on the robot, even though there is some additional processing overhead associated with encoding individual images or a stream. Latency is slightly higher for Ethernet and Wi-Fi connections, which can be considered acceptable. However, it is approximately double for a 5G connection, which we attribute to the usage of a commercial

network. With a private network, it should be possible to obtain substantially better results. For the 10 FPS video, all measured variants achieved approximately 10 FPS for results. There is much higher variability for the 25 FPS one, and it seems that, for cases where the network application is offloaded, HTTP has worse performance than other transport methods. With an application in the cloud, CPU usage is lower, except for GStreamer transport, which has to be further investigated. RAM usage is lower in all cases. When running non-locally, the GPU is not used; therefore, its usage is zero. This can be seen as an advantage as GPU might be omitted, making robot hardware simpler.

VI. Conclusion

The paper presented a framework for network applications designed and developed as a part of the 5G-ERA Project efforts towards enhancing robot autonomy through intelligent offloading of computational tasks to the cloud. The framework's architecture is simple and general, using well-known and supported technologies such as HTTP, SocketIO, and GStreamer. It is ROS-agnostic and does not rely on VPN and multicast (used by DDS), which is unusable with some cloud providers. The main intent is to allow fast packaging of the existing algorithms into containerized network applications with minimal effort. There are supportive libraries for both network application (server) and client publicly available, as well as the reference implementation of the network application using the MMDetection package, which can be configured, without any changes to the code, to perform various object detection tasks or take as a template and adapted for other use cases. Such applications can then benefit from compatibility with the 5G-ERA Project ecosystem. The reference implementation was evaluated on three platforms (robot, edge, and cloud) and with various types of transport for image data (HTTP, SocketIO, GStreamer). The results have shown that deployment of an object detection algorithm in the form of the network application, based on our framework, is achievable and leads to a significant decrease in power consumption while preserving reasonable performance. It turned out that using a commercial 5G network does not possess any advantage over Wi-Fi and a private network is needed when low latency and high reliability are needed.

So far, we have focused on applications where a client sends images and receives some data, e.g., object detection results, as it is probably the most common use case for mobile robots. The following research will evaluate more delicate use cases such as SLAM or server-based augmented reality. Also, the framework will be extended from the currently supported $1:1$ communication schema to $1:n$ and $n:m$. Lastly, automatic failure detection and recovery will improve the framework's robustness.

Acknowledgment

This paper was supported by the 5G-ERA project with funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No. 101016681.

References

[1] R. Qiu, D. Li, A. L. Ibáñez, Z. Xu, and R. L. Tarazón, "Intent-based deployment for robot applications in 5g-enabled non-public networks," 2023.

[2] M. Sophocleous, C. Lessi, Z. Xu, J. Špaňhel, R. Qiu, A. Lendinez, I. Chondroulis, and I. Belikaidis, "Ai-driven intent-based networking for 5g enhanced robot autonomy," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops: MHDW 2022, 5G-PINE 2022, AIBMG 2022, ML@ HC 2022, and AIBEI 2022, Hersonissos, Crete, Greece, June 17–20, 2022, Proceedings*. Springer, 2022, pp. 61–70.

[3] C. Lessi, G. Agapiou, M. Sophocleous, I. P. Chochliouros, R. Qiu, and S. Androulidakis, "The use of robotics in critical use cases: The 5g-era project solution," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops: MHDW 2022, 5G-PINE 2022, AIBMG 2022, ML@ HC 2022, and AIBEI 2022, Hersonissos, Crete, Greece, June 17–20, 2022, Proceedings*. Springer, 2022, pp. 148–155.

[4] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[5] A. Koubaa, M. Alajlan, and B. Qureshi, "Roslink: bridging ros with the internet-of-things for cloud robotics," *Robot Operating System (ROS) The Complete Reference (Volume 2)*, pp. 265–283, 2017.

[6] J. Z. Lim and D. W.-K. Ng, "Cloud based implementation of ros through vpn," in *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. IEEE, 2019, pp. 1–5.

[7] R. White and H. Christensen, "Ros and docker," *Robot Operating System (ROS) The Complete Reference (Volume 2)*, pp. 285–307, 2017.

[8] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[9] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, H. Zhan, D. Xu, R. Ghassemi, J. Kubiatowicz *et al.*, "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2," *arXiv preprint arXiv:2205.09778*, 2022.

[10] O. Arouk and N. Nikaein, "Kube5g: A cloud-native 5g service platform," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[11] I. Rodriguez, R. S. Mogensen, A. Fink, T. Raunholt, S. Markussen, P. H. Christensen, G. Berardinelli, P. Mogensen, C. Schou, and O. Madsen, "An experimental framework for 5g wireless system integration into industry 4.0 applications," *Energies*, vol. 14, no. 15, p. 4444, 2021.

[12] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[13] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.