electronics

MDPI

Article

Edge-Computing-Enabled Low-Latency Communication for a Wireless Networked Control System

Daniel Poul Mtowe [1] **and Dong Min Kim** [1,2,*]

[1] Department of ICT Convergence, Graduate School, Soonchunhyang University, Asan 31538, Republic of Korea; danielmtowe@sch.ac.kr
[2] Department of Internet of Things, SCH Media Labs, Soonchunhyang University, Asan 31538, Republic of Korea
* Correspondence: dmk@sch.ac.kr; Tel.: +82-41-530-1535

Abstract: This study proposes a novel strategy for enhancing low-latency control performance in Wireless Networked Control Systems (WNCSs) through the integration of edge computing. Traditional networked control systems require the receipt of raw data from remote sensors to enable the controller to generate an appropriate control command, a process that can result in substantial periodic communication traffic and consequent performance degradation in some applications. To counteract this, we suggest the use of edge computing to preprocess the raw data, extract the essential features, and subsequently transmit them. Additionally, we introduce an adaptive scheme designed to curtail frequent data traffic by adaptively modifying periodic data transmission based on necessity. This scheme is achieved by refraining from data transmission when a comparative analysis of the previously transmitted and newly generated data shows no significant change. The effectiveness of our proposed strategy is empirically validated through experiments conducted on a remote control system testbed using a mobile robot that navigates the road by utilizing camera information. Through leveraging edge computing, only 3.42% of the raw data was transmitted. Our adaptive scheme reduced the transmission frequency by 20%, while maintaining an acceptable control performance. Moreover, we conducted a comparative analysis between our proposed solution and the state-of-the-art communication framework, WebRTC technology. The results demonstrate that our method effectively reduces the latency by 58.16% compared to utilizing the WebRTC alone in a 5G environment. The experimental results confirm that our proposed strategy significantly improves the latency performance of a WNCS.

Keywords: wireless networked control system; edge computing; low latency; testbed

Citation: Mtowe, D.P.; Kim, D.M. Edge-Computing-Enabled Low-Latency Communication for a Wireless Networked Control System. *Electronics* **2023**, *12*, 3181. https://doi.org/10.3390/electronics12143181

Academic Editors: Soochang Park, Hosung Park and Stefano Scanzio

Received: 12 June 2023
Revised: 17 July 2023
Accepted: 20 July 2023
Published: 22 July 2023

1. Introduction

In this research, we propose an innovative technique that seamlessly blends edge computing into a wireless networked control system (WNCS), allowing a nuanced exploration of the intricate interplay between improved network performance and controlled computing power utilization. Our primary objective centers on achieving low-latency remote control. Our methodology diverges from conventional WNCS strategies by employing edge computing to enable low-latency communication, effectively surmounting the challenges imposed by unstable communication environments. This novel amalgamation of edge computing presents a compelling solution to the inherent instability often experienced in such environments, culminating in the achievement of optimal low-latency control performance. Furthermore, we propose an adaptive strategy designed to alleviate the pressure of frequent data traffic by judiciously modulating the periodic data transmission based on contextual necessity.

1.1. Background

In the wake of advancements in artificial intelligence (AI) and the Internet of Things (IoT), interest in smart cities is burgeoning. A significant portion of the smart cities' framework is predicated on networked control systems. A networked control system (NCS) constitutes sensors, controllers (or control servers), actuators, and a communication network [1]. The sensor assimilates environmental information and conveys it to the controller. The controller, in turn, formulates and dispatches control commands to the actuator. In specific instances, the sensor and actuator functions are amalgamated. The NCS epitomizes a distributed system where communication transpires through a shared network between actuators and controllers. In a wireless NCS (WNCS), the control loops are sealed via a volatile and inconsistent wireless communication link, facilitating the transmission of control and feedback signals amidst system components [2]. A distinguishing characteristic of a WNCS is its capacity to remotely execute numerous tasks geographically distant from each other. In a WNCS, the communication period between the controller and actuator plays a pivotal role in upholding optimal performance. Consequently, in contrast to conventional control systems, a WNCS can eliminate superfluous wiring amongst system components, thus diminishing the complexity and overall expense associated with designing and implementing the corresponding control systems. Additionally, a WNCS facilitates easy modifications or upgrades when there is a necessity to introduce or remove certain system components without inducing major structural changes. As such, a WNCS has been employed in a wide range of applications, including industrial control [3]. A salient example of a WNCS application is the Mars exploration rover, which was remotely operated by a team at the National Aeronautics and Space Administration (NASA), even amidst prolonged intervals of communication blackouts [4].

1.2. Problem Statement

Wireless networks, while widely used for communication, often present limitations leading to imperfect data transmission [5]. Several challenges emerge in the context of WNCS usage, including issues pertaining to sampling, network delays, and packet dropouts. Frequent periodic sampling tends to exert a significant load on the network, potentially culminating in network congestion. Transmission of extensive data volumes across communication networks, especially those with limited bit rates, is frequently accompanied by delays and packet loss. It is universally acknowledged that communication congestion can trigger prolonged latency, heightened packet loss, and diminished throughput, consequently impinging on the system's stability and reliability [6–8]. The present study grapples with the task of mitigating communication overheads, which serve as performance constraints in the context of low-latency remote control via a WNCS.

1.3. Motivation to Research

Researchers with a focus on control theory often neglect the communication aspect, while those centered on communication theory frequently overlook computational considerations. To enhance performance from a systems perspective, it is imperative to concurrently contemplate communication and computation. Guided by this rationale, we introduce an edge-computing-assisted WNCS, convinced that the application of edge computing would be optimal. The operation of the WNCS is characterized by a tradeoff between the conservation of communication bandwidth and the enhancement of control efficiency. This study scrutinizes the interplay between the network performance improvement and the computing power consumption brought about by the edge computing of a device.

1.4. Proposed Approach

Conventional WNCSs typically operate in the following manner, as illustrated in Figure 1a. They first recognize the surrounding environment using sensors, which then transmit the sensing data to a remote controller. This remote controller processes the data to formulate an appropriate control command, which is then transmitted to the actuator. The

actuator executes tasks in accordance with the received control command. In contrast to a traditional WNCS, which unconditionally transmits data in a routine manner, we propose an edge-computing-assisted WNCS as depicted in Figure 1b. This system determines whether the data warrants transmission to the remote controller, thereby adding a layer of efficient data management to the process.

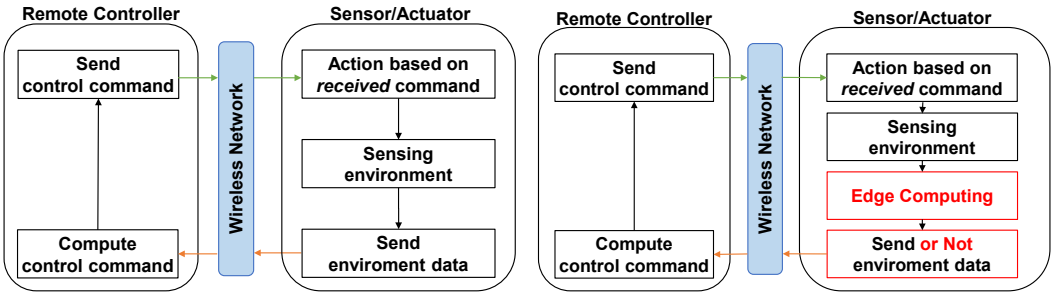


Figure 1. Brief block diagram of the conventional WNCS and the proposed WNCS. (**a**) Conventional WNCS; (**b**) Proposed WNCS.

The integration of edge computing into the WNCS forms a significant component of our proposed scheme. Edge computing redistributes computing tasks from a remote location to a proximal one [9]. Many IoT devices often grapple with constraints such as limited computing power and battery capacity, which pose barriers to the development of novel applications and services. Edge computing can circumvent the computation capacity and limited battery power [10] of IoT devices. Our approach to edge computing is distinct: instead of employing it to overcome limited computing power, we introduce edge computing as a solution to the unstable communication environment typical of the WNCS.

Offloading computational tasks to a remote server may be hindered by limited bandwidth, unreliable wireless links, or computational latency at an overloaded server. Mitigating the communication–computation latency is a formidable challenge due to its inverse correlation. The ultimate objective is to achieve low-latency control performance. Novel communication methodologies are needed to attain the desired control performance while reducing the communication–computation latency and preserving bandwidth resources.

1.5. Contributions

In this study, we analyze the WNCS that generates voluminous data, necessitating continuous transmission and computation to estimate the system state. We employ edge computing to locally preprocess the information and decrease the size of the transmitted data and then propose an adaptive scheme to decide when to transmit the data. Compared to a conventional WNCS, the adaptive scheme only updates its value when the deviation between the current state and the desired state exceeds a predetermined threshold. This method is noteworthy as it reduces the obligation to transmit sampled states at regular intervals, improving communication efficiency by avoiding superfluous transmissions.

The primary contributions of this study are summarized as follows:

1. We propose an adaptive scheme, and we corroborated its effectiveness through real-world implementation. Our approach exhibits a reduction in communication cycles between the actuator and the remote server without compromising performance.
2. We devised an algorithm to compute the threshold value for the proposed adaptive scheme. This algorithm ensures a threshold value that can balance the tradeoff between reduced communication cycles and system performance.
3. We carried out extensive experiments using a developed testbed to evaluate the performance of the proposed system. The experimental results affirmed that our proposed system decreased the communication cycles between the actuator and the remote server in comparison to a conventional system.

In our research endeavor, we strive to illuminate the scientific novelty associated with the integration of edge computing. Our focus lies in leveraging edge computing to preprocess raw data, extract crucial features, and subsequently transmit them. Additionally, we introduce an adaptive scheme designed to reduce the frequency of data traffic by dynamically adjusting the periodicity of data transmission based on necessity. We emphasize the potential implications of this scheme for enhancing network performance and improving control efficiency.

1.6. Structure of the Paper

The structure of this paper is delineated as follows: Section 2 reviews the relevant literature in the field. The system under study, the mathematical model, and the proposed adaptive scheme are elaborated in Section 3. The experimental setup, implementation, and subsequent discussion of the results are encompassed in Section 4. We articulate the limitations of the proposed approach in Section 5. Finally, the paper is concluded in Section 6 with key insights and potential future research directions.

2. Related Works

The importance of a stable network for ensuring optimal control performance is stressed in [11]. Contrary to simpler protocols such as a UDP, an advanced protocol like a TCP can validate the success of packet transmission, thereby enhancing the stability. Nonetheless, if a network's performance is ill-suited for operating a TCP, the benefits of implementing a complex protocol might be negligible. In such scenarios, we suggest that it is more advantageous to focus on mitigating the network traffic. Prior studies from this point of view include the following. Huang et al. proposed a hybrid automatic repeat request (HARQ) by optimizing the sensor's online transmission control policy [12]. This research also endeavored to curtail the overhead of the ACK signal by integrating a sophisticated communication protocol. Conversely, our proposed method is to primarily reduce the network traffic without necessitating any advancement of the communication protocol itself. Cho et al. attempted to augment the actual data rate by comprehending the channel condition and adjusting the transmission power [13]. Yet, the practicality of implementing such functionalities may be doubtful in devices with performance constraints. Gatsis et al. proposed a scheduler that monitored the wireless channel and opportunistically transmitted at a set of nonoverlapping frequencies [14]. In parallel, Huang et al. recommended a practical half-duplex controller for minimizing the long-term average cost [15]. While these studies [14,15] advocate for the use of optimal radio channels for data transmission, they do not curtail the regularly occurring data traffic in the same way as our adaptive scheme. Study [14] primarily focused on improving the communication protocol, whereas our approach involves a distinct methodology that adapts periodic data transmission according to necessity, regardless of the communication protocol in use. Our suggested approach aims to maintain the performance levels without compromising the efficiency. In their work, Demirel et al. postulated that introducing a threshold to trigger the actuator could diminish the communication burden and guard against network congestion [16]. Yet, the control efficiency is often compromised due to the constant sampling period employed in these studies. Depending on the actuator's situation, when fine control is required, data need to be transmitted more frequently. Conversely, data can be transmitted less frequently when the control required is less precise. Ref [17] stated that a control system for drones swarms in a centralized control, where a ground controller tells the drones how to navigate to their final position. The drone swarm is considered as an NCS, where the control of the overall system is enclosed within a wireless communication network. However, the wireless network model is simplified and not realistic. In their research, Tiberkak et al. employed Web Real-Time Communication (WebRTC) technology to significantly reduce the latency in a Multiple Operator Single Robot (MOSR) telerobotic platform [18]. By leveraging the WebRTC, they discovered that the latency encountered during the remote control of robots over the internet remained within tolerable bounds.

2.1. Improving the Performance of WNCS

Several strategies have been explored in an attempt to decrease communication overheads in a WNCS. The prevailing research on WNCSs [11–13] often proposes intricate communication algorithms to address the constraints associated with the communication bandwidth and network congestion. The implementation of such complex communication techniques invariably necessitates an upgrade to the communication module of the relevant devices, consequently escalating the construction cost of a WNCS. Several studies [14–16] have attempted to minimize the communication instances by elongating the sampling period. However, given that these existing studies consistently utilize the same sampling cycle, a resultant decrease in the control efficiency is a notable issue.

2.2. Edge Computing for IoT

Edge computing has provided significant opportunities for Internet of Things (IoT) technologies, as presented in [19,20], wherein the deployment of computing servers proximate to IoT devices enhances computational resources. The authors of [21] further developed this concept, introducing deep learning into IoT within edge computing to bolster network performance, suggesting the integration of additional layers into edge servers as a strategy for minimizing network traffic. In [22], a novel edge-assisted solution was proposed to augment the cognition and self-awareness of an IoT system's sensing, processing, and configuration.The authors of [23] investigated offloading time-critical control operations for a UAV to external computational resources using 5G and edge computing. Furthermore, ref. [24] presented a time-varying delay compensation system for motion control through access edge computing. This proved successful in stabilizing motor control under strenuous network load conditions, where packet loss and delay were prevalent, and traditional techniques failed to regulate the motor. While most research has focused on edge computing as a means to circumvent the performance restrictions of IoT devices, our study diverges by illustrating how the performance of a WNCS can be optimized by utilizing computational resources, despite their inherent performance limitations.

2.3. Computation Offloading

Computation offloading, conceptually akin to edge computing [25,26], is the process whereby a task allocated to one machine is transferred to another. Tasks intended for IoT devices can be offloaded to a server, and those for the server can be passed to peripheral devices. Improving computation offloading performance necessitates the formulation of strategic offloading and optimization techniques. In [27], a Lyapunov-based dynamic computing scheduling technique was proposed to mitigate the execution delay and computational task failure arising from insufficient battery energy. Ref. [28] discussed an optimal computation offloading policy for industrial IoT systems, based on a reinforcement learning (RL) algorithm. Ref. [29] put forth the concept of partial offloading, wherein each task could be arbitrarily divided into local and edge computing components. The intensive tasks were offloaded to proximal servers to decrease the computational latency and energy consumption. Refs. [30,31] addressed the computation offloading policies related to feature compression and transmission at the edge. They proposed model splitting, model compression, and feature encoding to lower the end-to-end latency. The authors of [32] suggested an autonomous computation offloading framework to confront the challenges associated with time- and resource-intensive applications. The authors of [33] explored using AI to reduce the decision-making costs in computation offloading, while [34] used machine learning (ML) strategies to enhance intelligent offloading in Mobile Edge Computing (MEC), discussing their applications in augmented reality, connected vehicles, and stream analysis. The authors of [35] used physical layer security technology and spectrum-sharing architecture to provide a joint secure offloading and resource allocation (SORA) strategy based on deep reinforcement learning to reduce the system processing latency while guaranteeing the integrity of the wireless offloading process. The authors of [36] put forward a privacy-oriented task-offloading method capable of resisting attacks from privacy attackers

possessing prior knowledge, leveraging a deep reinforcement learning model to reduce the location privacy breach risks for high-performance intelligent autonomous transport systems. The authors of [37] introduced a real-time multiple-workflow scheduling (RMWS) plan that optimized workflow scheduling to minimize the cost and meet diverse deadlines. This proposed algorithm outperformed two leading algorithms in terms of the total rental cost, resource usage, success rate, and missed deadlines. Despite the wealth of existing studies, the computational offloading characteristics of a WNCS are often overlooked. Our research aims to minimize the communication overhead by integrating edge computing to maximize the computational offloading performance of a WNCS.

The majority of the existing studies [21,24,26–28] have proposed solutions primarily based on simulation models. In contrast, our research emphasizes the practical applicability by validating the performance of our proposed scheme through its implementation on a real-world testbed. The related works are summarized in Table 1.

Table 1. Summary of Related Works.

Features Considered		Related Works	Our Work
Reducing transmission	Dynamic sampling cycle	-	**considered**
	Elongated sampling cycle	[14–16,18]	-
Edge computing	Remedy for reducing the data size to be transmitted	-	**considered**
	Remedy for limited computing power	[21–24,27–33,35,36]	-
Computation offloading		[27–33,35,36]	**considered**
Only considered communication protocol		[12–14,35,36]	-
Proposed solution	Implemented on a real-world testbed	[18,22–24]	**considered**
	Performed by simulation	[12–16,21,27–33,35,36]	-

3. System Description

This study considers a WNCS in which a remote vehicle follows a predetermined route while being operated by a server that processes camera data from the vehicle and sends appropriate control commands. Connectivity is maintained via a wireless network, which transmits the state of the actuator (in this case, the remote vehicle) and allows the server to issue control commands. The system under consideration comprises two main components: the remote vehicle and the server. The vehicle provides sensor data concerning its environment to the server, which processes this information and makes decisions in real time. These decisions are then transmitted back to the vehicle through the wireless channel, enabling it to execute assigned tasks. The ongoing communication keeps the remote control system operational. The system architecture is presented in Figure 2.

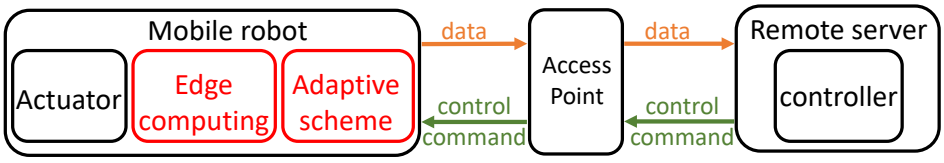


Figure 2. Diagram of a wireless networked control system.

IoT devices, such as remote vehicles, require substantial data transmission to the control server for environment recognition and responsive control. This data transmission incurs a network-induced delay, which is particularly undesirable in time-critical applications such as autonomous vehicles. Our approach employs edge computing at the actuator end to reduce the size of the data communicated and uses an adaptive scheme to determine the optimal communication timing, as depicted in Figure 2. Instead of transmitting all the collected data, which would be inefficient, our goal is to minimize the amount of data sent from the vehicle to the server without performance loss. We propose a two-pronged

approach: employing edge computing on the vehicle side (albeit with an increased computational load), and utilizing an adaptive scheme that transmits data based on the observed feature deviation within the data pattern. These strategies are detailed further in Section 3.5.

In the system under consideration, the vehicle followed a route indicated by a black line, detected by the vehicle's onboard camera. A Proportional-Integral-Derivative (PID) controller, a simple yet effective method for system stabilization without continuous oscillations, was used for this task. The structure of the PID controller is outlined in Figure 3. The PID input is the error value (e), the difference between the desired and measured values of the black line centroid in the captured image. The PID controller aims to generate a control command (f) that eliminates this error. We introduce a measure termed *deviation* to gauge the performance of the vehicle following the road correctly. The *deviation*is defined as the difference between the center point of the vehicle's camera screen and the road's center point. Therefore, in this context, e represents the deviation. The proportional gain (P) is used to ensure the immediate response of the control signal f to the error, thereby enhancing the system's response speed and reducing the steady-state error. Integral gain (I) helps eliminate the offset error, but may induce overshoot, which is curtailed using the derivative gain (D). The vehicle's camera serves as a sensor, gathering environmental information, and the processed features from the camera images assist in ensuring that the vehicle follows the track by determining whether to maintain forward motion or turn.

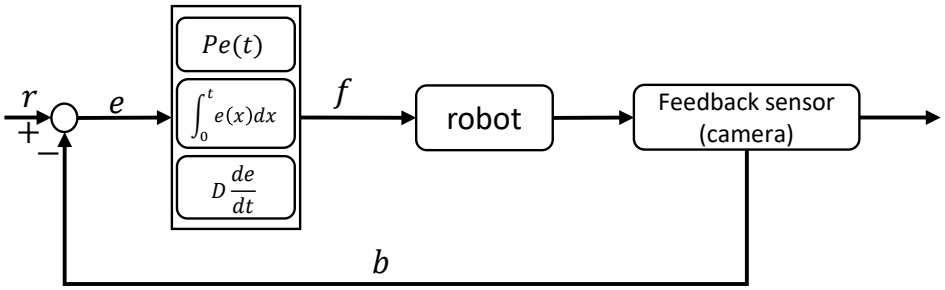


Figure 3. PID system diagram.

The remote vehicle operates in three distinct modes: (i) puppet mode, where the vehicle merely captures an image and forwards it to the server; (ii) edge computing mode, where the vehicle performs preliminary computations for data preprocessing before transmission, sending extracted features rather than all information to the server; and (iii) local mode, in which the vehicle autonomously carries out all tasks.

3.1. Puppet Mode

In puppet mode, the server controls the vehicle based on the raw information received from it. This approach enables us to leverage powerful algorithms without concerns about processing power and computational delay. Consequently, data processing and command generation are relocated to the server. A variety of wireless broadband connectivity technologies (e.g., 3G, LTE, and 5G network) are used to connect to the server. They may, however, incur network overhead. The remote vehicle's environmental state and the control commands are transmitted via socket communication. The vehicle primarily performs image acquisition, forwards the image to the server, and executes the server's control command. As a result of these operations being performed remotely, there may be undesirable side effects, such as network overhead due to the concurrent communication between the vehicle and the server. Given the requirement of real-time operation, network traffic and delays might impair the system performance. Therefore, we introduce improved techniques to reduce the data exchange between the vehicle and the server.

Mathematical Modeling of Puppet Mode

In puppet mode, the remote server handles the computational task. Here, f^s denotes the remote server's computing capability, expressed in CPU cycles per second. The compu-

tational delay caused by the server's computation is dependent on f^{s}. The data size of the ith task is represented as D_i (in bits), and the required CPU cycles to accomplish this task are symbolized as Y_i. The value of Y_i is postulated as a linear function of D_i:

$$Y_i = kD_i, \tag{1}$$

where k is a proportionality constant, and $k > 0$. The computation time at the remote server is expressed as:

$$T^{\mathrm{P,c}} = \frac{Y_i}{f^{\mathrm{s}}}. \tag{2}$$

In addition to the computation time, uploading data to the remote server induces an extra delay. The uplink rate, R (in bits/sec), and the data size, D_i, determine the transmission time for offloading the task to the server. The time needed to upload data in puppet mode, $T^{\mathrm{P,tx}}$, is thus calculated as:

$$T^{\mathrm{P,tx}} = \frac{D_i}{R}. \tag{3}$$

As per Equations (2) and (3), the total delay in puppet mode, T^{P}, is obtained:

$$T^{\mathrm{P}} = T^{\mathrm{P,tx}} + T^{\mathrm{P,c}} = \frac{D_i}{R} + \frac{Y_i}{f^{\mathrm{s}}}. \tag{4}$$

The remote server has abundant energy resources available compared to the vehicle. Therefore, we only consider the energy consumed for upload data in puppet mode from the vehicle to the remote server. The energy consumption of the vehicle upload data in puppet mode $E^{P,tx}$, is thus calculated as:

$$E^{\mathrm{P,tx}} = P^{\mathrm{tx}} T^{\mathrm{P,tx}} = \frac{P^{\mathrm{tx}} D_i}{R}, \tag{5}$$

where P^{tx} is the transit power of the vehicle to send data.

3.2. Edge Computing Mode

In edge computing mode, the computational capabilities are incorporated into the vehicle, facilitating image preprocessing before transmission to the server. By segmenting the computational tasks, different tasks can be assigned to the vehicle and server. The system under consideration used edge computing level one, which involved additional preprocessing on the vehicle beyond mere image capture and transmission. Our experimental approach applied grayscale conversion, blurring, thresholding, and contour detection to the captured image before transmission. This approach reduced the data size compared to puppet mode. Edge computing level two extended the data processing to the moment calculation at the vehicle, further reducing the data transmission size. Edge computing level three performed centroid computation on the vehicle, substantially compressing the transmitted data. The variable λ is introduced in Section 3.2.1 to represent different computational distributions.

3.2.1. Mathematical Modeling of the Edge Computing Mode

In the edge computing mode, the variable λ is introduced to illustrate the extent of the computational task distribution between the remote vehicle and the server. The variable λ indicates the proportion of tasks processed at the remote vehicle, with a range of $0 \leq \lambda \leq 1$. Considering the total task as one unit, the vehicle computes λ portion of the task and transmits the remaining $1 - \lambda$ to the server. Three types of delay are considered in this mode: the local computation delay, the transmission delay, and the server computation delay. The delay for processing Y_i at the vehicle, $T^{\mathrm{E,cv}}$, is:

$$T^{\mathrm{E,cv}} = \frac{\lambda Y_i}{f^{\mathrm{v}}}, \tag{6}$$

where f^{v} symbolizes the remote vehicle's computing capability. The delay required for transmitting $(1-\lambda)D_i$ to the remote server, $T^{\mathrm{E,tx}}$, is:

$$T^{\mathrm{E,tx}} = \frac{(1-\lambda)D_i}{R}. \tag{7}$$

The delay required for processing $(1-\lambda)Y_i$ at the remote server, $T^{\mathrm{E,cs}}$, is:

$$T^{\mathrm{E,cs}} = \frac{(1-\lambda)Y_i}{f^{\mathrm{s}}}. \tag{8}$$

Based on Equations (6)–(8), the total delay in the edge computing mode, T^{E}, can be determined:

$$T^{\mathrm{E}} = T^{\mathrm{E,cv}} + T^{\mathrm{E,tx}} + T^{\mathrm{E,cs}} = \frac{\lambda Y_i}{f^{\mathrm{v}}} + \frac{(1-\lambda)D_i}{R} + \frac{(1-\lambda)Y_i}{f^{\mathrm{s}}}. \tag{9}$$

As explained before, due to the abundant availability of energy available at the remote server, we only consider the energy consumed for the upload data in the remote sever. Hence, two types of energy consumption are considered in this mode: the local computation energy and the transmission energy. The energy consumption for processing Y_i at the vehicle, $E^{\mathrm{E,cv}}$, is:

$$E^{\mathrm{E,cv}} = \lambda P^{\mathrm{cv}} Y_i. \tag{10}$$

where P^{cv} represents the energy consumption per CPU cycle to complete the computation task. The energy required for transmitting $(1-\lambda)D_i$ to the remote server, $E^{\mathrm{E,tx}}$, is:

$$E^{\mathrm{E,tx}} = \frac{(1-\lambda)P^{\mathrm{tx}}D_i}{R}. \tag{11}$$

Based on Equations (10) and (11), the total energy consumption in the edge computing mode, E^{E}, can be determined by:

$$E^{\mathrm{E}} = E^{\mathrm{E,cv}} + E^{\mathrm{E,tx}} = \lambda P^{\mathrm{cv}} Y_i + \frac{(1-\lambda)P^{\mathrm{tx}}D_i}{R}. \tag{12}$$

3.3. Local Mode

Finally, in local mode, the vehicle carries out all the computational tasks required to generate the control command. After capturing images, they are processed individually on the vehicle by detecting the environment and calculating the black line center from the image. The error between the detected and reference center points is then calculated. This error value serves as input to the PID controller, which outputs a control command to ensure the vehicle stays on course by continuously adjusting to achieve smooth line tracking. However, performing all procedures on a resource-constrained vehicle, e.g., one with limited computational power and battery performance, may degrade the system's performance. Offloading computations to a remote server provides a promising solution to address these shortcomings.

Mathematical Modeling of Local Mode

In local mode, the computation task is handled independently of any other devices. The standalone device's computational capability is represented by f^{v}. The time necessary to execute a given task, T^{L}, is:

$$T^{\mathrm{L}} = \frac{Y_i}{f^{\mathrm{v}}}. \tag{13}$$

In addition, we have E^{L} (in Joule) as the corresponding energy consumption of the standalone device's computational capability, which is expressed as:

$$E^{\mathrm{L}} = P^{\mathrm{cv}} Y_i, \tag{14}$$

where P^{cv} represents the energy consumption per CPU cycle to complete the computation task.

3.4. Determining the Operation Mode

3.4.1. Determining the Operation Mode Considering Latency

The delay was quantified in this study to facilitate low-latency control. This analysis assumed that the communication delay incurred while receiving the control command from the server could be disregarded, primarily because the control command's size was relatively small and remained constant across different remote control modes.

The system had the capability to operate in puppet mode, edge computing mode, or local mode. However, these configurations could be represented solely by the edge computing mode. When $\lambda = 0$, the system defaulted to local mode, while it transitioned to puppet mode when $\lambda = 1$. The goal was to minimize the total delay T^E, which required determining an optimal λ value.

$$\begin{aligned} &\underset{\lambda}{\text{minimize}} \quad T^{\mathrm{E}} \\ &\text{subject to} \quad 0 \leq \lambda \leq 1. \end{aligned} \tag{15}$$

The objective function in Equation (22) can be expanded as:

$$\begin{aligned} T^{\mathrm{E}} &= \lambda \frac{Y_i}{f^{\mathrm{v}}} + (1-\lambda)\frac{D_i}{R} + (1-\lambda)\frac{Y_i}{f^{\mathrm{s}}} \\ &= \lambda \frac{Y_i}{f^{\mathrm{v}}} + (1-\lambda)\left(\frac{D_i}{R} + \frac{Y_i}{f^{\mathrm{s}}}\right) \\ &= \lambda T^{\mathrm{L}} + (1-\lambda)T^{\mathrm{P}}, \end{aligned} \tag{16}$$

where $T^{\mathrm{L}} = \frac{Y_i}{f^{\mathrm{v}}}$, and $T^{\mathrm{P}} = \frac{D_i}{R} + \frac{Y_i}{f^{\mathrm{s}}}$. If the computation offloading time is less than the local computation time, offloading computation could enhance the user experience. Specifically, if $T^{\mathrm{L}} > T^{\mathrm{P}}$, puppet mode would be the optimal solution. Conversely, if the local computation time is less than the offloading computation time, i.e., $T^{\mathrm{L}} < T^{\mathrm{P}}$, operating in local mode would be the optimal solution, as it could better enhance the user experience.

3.4.2. Determining the Operation Mode Considering Vehicle Energy Consumption

Determining the appropriate operation mode for a vehicle considering the additional energy consumption incurred due to preprocessing on the vehicle involves considering the tradeoff between channel conditions and the available energy level on the vehicle side. The channel condition refers to the quality of the communication channel, which affects the time required to transmit data. Good channel conditions result in shorter communication times, while poor channel conditions can significantly prolong the communication process. Therefore, we can associate the channel condition with the required delay for transmitting data from the vehicle to the remote server. To express the computation overhead of the vehicle, we can represent it as a linear combination of energy consumption and time delay. The vehicle's requirements for determining which operation mode to be taken can be captured by weighing parameters, denoted as W^T for the energy consumption and W^E for time delay, respectively. These parameters fall within the range [0, 1]. For example, if the channel condition is very poor, the value of W^T could be set higher than W^E to prioritize time delay. Conversely, if the vehicle has a low battery state, the value of W^E could be set higher than W^T to prioritize energy consumption.

Hence, for the puppet mode the computation overhead of the vehicle is determined by the computation time at the remote server, the time needed to upload data in puppet mode, and its corresponding transmission energy.

$$C^{\mathrm{P}} = W^{\mathrm{T}}T^{\mathrm{P}} + W^{\mathrm{E}}E^{\mathrm{P,tx}} = W^{\mathrm{T}}(T^{\mathrm{P,tx}} + T^{\mathrm{P,c}}) + W^{\mathrm{E}}P^{\mathrm{tx}}T^{\mathrm{P,tx}} = W^{\mathrm{T}}(\frac{D_i}{R} + \frac{Y_i}{f^{\mathrm{s}}}) + W^{\mathrm{E}}\frac{P^{\mathrm{tx}}D_i}{R}. \tag{17}$$

For the case of the edge computing, the computation overhead of the vehicle is determined by the computation time at the remote server, the time needed to upload data in

puppet mode, its corresponding transmission energy, its local computation time, and its corresponding energy consumption for execution on the vehicle:

$$C^{\mathrm{E}} = W^{\mathrm{T}} T^{\mathrm{E}} + W^{\mathrm{E}} E^{\mathrm{E}} = W^{\mathrm{T}} (T^{\mathrm{E,cv}} + T^{\mathrm{E,tx}} + T^{\mathrm{E,cs}}) + W^{\mathrm{E}} (E^{\mathrm{E,cv}} + E^{\mathrm{E,tx}}). \tag{18}$$

For the case of local computing, the computation overhead of the vehicle is determined by the local computing time and the consumed energy per CPU cycle:

$$C^{\mathrm{L}} = W^{\mathrm{T}} T^{\mathrm{L}} + W^{\mathrm{E}} E^{\mathrm{L}} = W^{\mathrm{T}} (\frac{Y_i}{f^{\mathrm{v}}}) + W^{\mathrm{E}} (P^{\mathrm{cv}} Y_i). \tag{19}$$

To define the total computation overhead of the vehicle, to determine the operation mode x of the vehicle, we introduced the indicator function $I(x, a)$ for the vehicle:

$$I(x, a) = \begin{cases} 1 & \text{if } x = a \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

We now express the computation overhead of the vehicle to determine the operation mode x of the vehicle as

$$C_x = C^{\mathrm{L}} I(x, 0) + C^{\mathrm{E}} I(x, 1, 2, 3) + C^{\mathrm{P}} I(x, 4), \tag{21}$$

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & C_{(x)} \\ \text{such that C1:} \quad & x \in \{0, 1, 2, 3, 4\}. \end{aligned} \tag{22}$$

In (22), the constraint $C1$ states that the vehicle can only chose one operation mode (e.g., local, edge level1, edge level 2, edge level 3, or puppet) to be used.

3.5. Adaptive Scheme

As elucidated in the preceding subsection, employing local or puppet mode would be sensible, if delay calculations can be made accurately. However, real-world scenarios present complications, such as a variable data rate R due to time-dependent channels and a fluctuating server's computational capacity resulting from multitasking. To enhance system performance under these conditions, we propose an adaptive approach.

Traditional WNCSs issue commands at regular intervals. This traditional control strategy tends to be conservative; it often leads to greater than necessary resource consumption to maintain a specific output level, primarily because it aims to assure reliability even under worst-case scenarios when the control process is executed at a high frequency. We suggest an intelligent WNCS design that smartly minimizes the communication overhead between the remote vehicle and the remote control server. The remote vehicle, in this system, does not merely act as a headless actuator; it also performs computations, and only the processed data are transmitted. Unexpected system changes disrupt the normal data flow, and the remote vehicle autonomously decides to communicate with the server. The proposed system is anticipated to require fewer communication cycles than its traditional WNCS counterpart. Figure 4a visualizes the information flow between the remote vehicle and the remote control server. In this figure, data_x are periodically transmitted from the vehicle to the server, resulting in congested links, in contrast to data_y which are adaptively transmitted based on feature difference, thus maintaining less communication traffic.

In a traditional WNCS, the actuator is controlled by exchanging information at regular intervals. However, this approach may lead to the repetitive transmission of similar information, thereby issuing the same control command repeatedly. Our proposed system provides the actuator with intelligent capabilities that enable autonomous decision making regarding the data transmission. This mitigates unnecessary data transfers, reducing the network traffic and enhancing system performance. Prior to transmission, the data are preprocessed to extract only the significant features, which are then sent to the remote server. The vehicle's onboard algorithm performs data analysis, comparing the current state with the previously executed state. If no significant changes are detected, the system utilizes

the last server command instead of sending data for processing and awaiting a response. This method ensures communication occurs only when necessary. The actuator's sampled data are released to the network only when a specific trigger condition is met. At each sampling instant, the data are analyzed to produce features for the adaptive trigger, which then decides whether to release the current feature based on a predefined feature threshold condition. A higher threshold value results in fewer data transmissions, reducing network congestion but at the risk of affecting the remote vehicle's performance due to fewer control commands. Conversely, a lower threshold increases the data release frequency and network traffic. When the threshold is zero, the adaptive scheme's trigger mechanism reverts to a general periodic mechanism with equal intervals. Hence, the threshold selection impacts the communication and performance of the remote vehicle, necessitating an optimal threshold value that minimizes the communication without compromising the vehicle performance. The operational flow is illustrated in Figure 4b. We provide an example of the adaptive scheme implemented in Section 4.6.

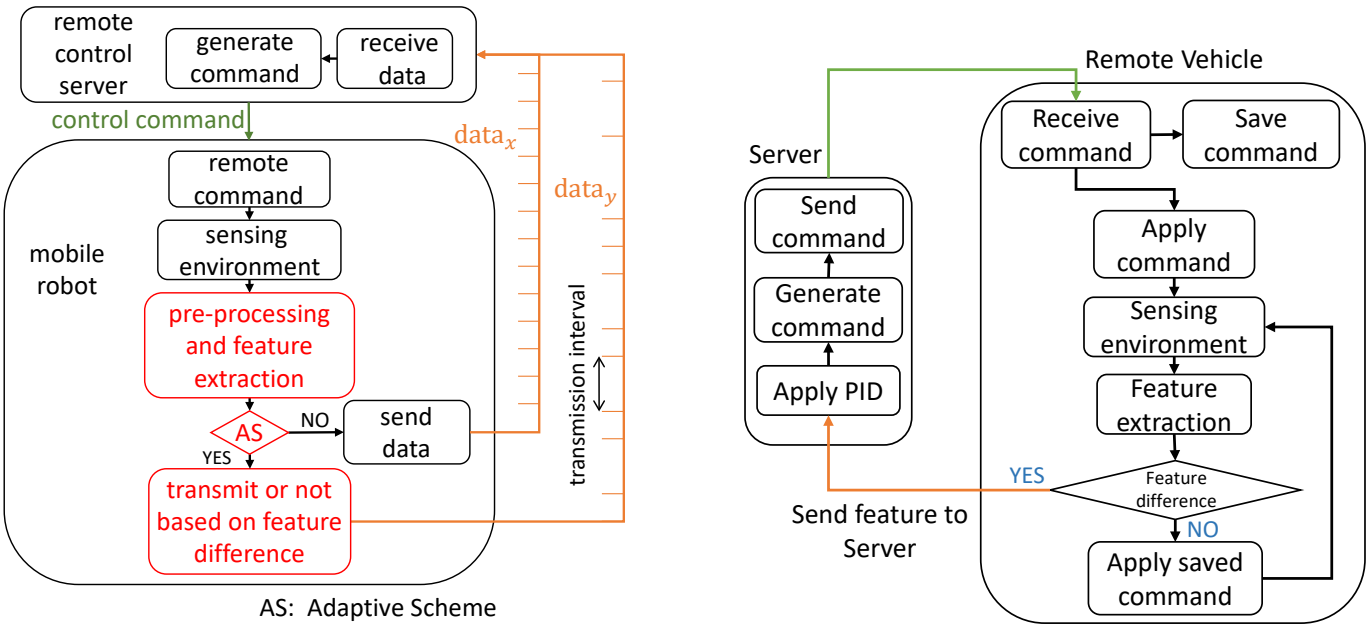


Figure 4. The proposed edge-computing-assisted WNCS. (**a**) Edge-computing-assisted WNCS. (**b**) Operational procedure of the adaptive scheme: The adaptive mechanism conducts feature extraction and comparison prior to delegating the computation to the server.

In summary, we propose two strategies to improve system performance. Firstly, to reduce data traffic, we recommend using edge computing to preprocess the raw data, extract the essential functions, and then transmit these. Secondly, atop this foundation, we propose an adaptive scheme aimed at reducing periodically the occurring data traffic. This adaptive scheme alters the periodic data transmission to only when necessary, avoiding unnecessary data transmission when no significant updates are detected, by comparing the newly generated data with previously transmitted data.

4. Experiments and Discussion

4.1. Testbed Setup

To evaluate the performance of the proposed scheme and benchmark it against alternative computational modes, we established a testbed, as depicted in Figure 5.

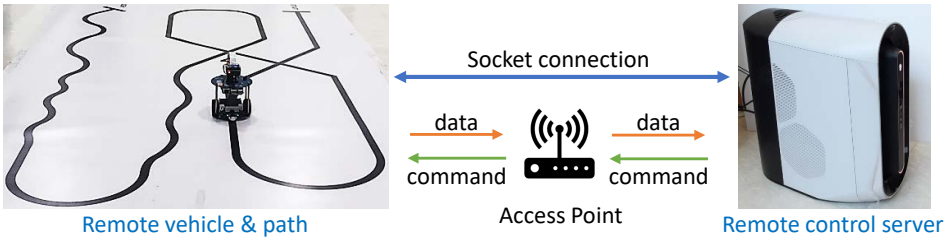


Figure 5. The testbed implementation of the wireless networked control system.

The testbed comprised a mobile robot (i.e., a remote vehicle) that traversed a set path, gathered environmental data, and transmitted this information to a remote control server. Based on the transmitted data, the remote control server dispatched a control command to the mobile robot, instructing it to follow a specific black line trajectory. Communication between the mobile robot and the remote control server was facilitated through a wireless network.

The mobile robot deployed in the testbed was manufactured by Waveshare Electronics and integrated with a Raspberry Pi 3B+. Table 2 delineates the hardware specifications of the Raspberry Pi. The mobile robot was equipped with a dual-wheel system for comprehensive mobility and a Pi camera module for environmental sensing. The camera, a Raspberry Pi camera, yielded images with a resolution of 160 × 128 pixels. These images served as the primary data input source, conveying the environmental context that enabled the navigation of the mobile robot. The path identification of the mobile robot was predominantly based on the output images from the camera. The communication module in the mobile robot enabled the transmission of the environmental data to the remote control server and the receipt of the control commands from the server. These control commands were subsequently conveyed to the motor module through a general-purpose input/output (GPIO) communication interface, inducing the desired motion response in alignment with the given control command. The control command generally encompassed values transmitted to each of the two motors, adjusting their speed to facilitate left or right turns or linear motion.

Table 2. Hardware specifications of the Raspberry Pi 3B+.

Product Details	Raspberry Pi 3B+
SOC	Broadcom BCM2837B0
Core Type	Cortex-A53 64-bit
No. of Cores	4
GPU	VideoCore IV
CPU	1.4 GHz
RAM	1 GB DDR2
Power Ratings	1.13A @ 5V

The server incorporated in this study was a Dell Alienware aurora R9, powered by a 3.6 GHz Intel Core i7-9700K with 16 GB DDR4 RAM (2666 MHz). Connectivity between the mobile robot and the server was ensured by various wireless communication technologies, including 3G, LTE, and 5G networks.

4.2. Assumptions Made

In the implementation of the edge computing-assisted WNCS, several assumptions were made. Firstly, it was assumed that the edge computing resources possessed adequate processing power and storage capacity to handle the data processing tasks, such as the real-time data analysis, decision making, and control algorithm execution. The assumption was that the edge devices could handle the computational load effectively without compromising the system performance. Secondly, it was assumed that the edge computing infrastructure was designed with energy efficiency in mind. This means that the edge devices were optimized for low power consumption while still delivering the necessary computational capabilities. The assumption was that energy-efficient designs

and algorithms were employed to maximize the overall energy efficiency of the system. Additionally, it is worth noting that the impact of the downlink was not considered in this scenario, as the size of the command sent back from the controller to the actuator was very small. This assumption implied that the focus was primarily on the uplink communication, neglecting the potential effects of the downlink transmission. These assumptions collectively shaped the implementation of the edge-computing-assisted WNCS.

4.3. Local Mode Implementation

The operational procedure of the local mode is delineated in Figure 6. The process, commencing from camera initialization and culminating in control command issuance, was segmented into nine distinct steps, from A through I. The image frames captured from the Raspberry Pi camera (A), served as the system's input data (B). For these image frames, we employed image processing techniques using OpenCV. Initially, the color image was transformed into a grayscale version (C). Subsequently, we applied blur and threshold processing to convert the image into a binary representation, with the road appearing as a continuous black line. The road contours were then detected, and the camera was aimed to center on the road to the greatest extent feasible (D). By comparing the contour's center point with the center of the camera-captured image, a control command was generated to minimize the difference between these two points. For this purpose, we computed the contour's moment (E) and its centroid (F). At this stage, the centroid was represented as x and y coordinate values. Given that the y coordinate corresponded to the forward direction and did not influence the control command, only the x coordinate value was retained. Subsequently, we calculated the difference between the image center point and the contour center point (G). The output from the image processing step was input to the PID controller to generate the corresponding control command (H). Finally, the control command was relayed to the motors, repeating the entire process (I).

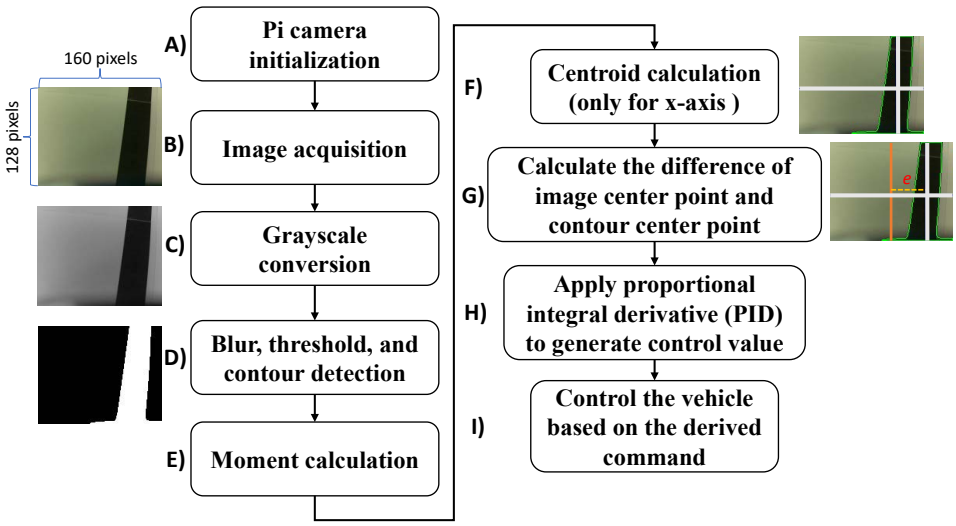


Figure 6. Local mode operated at vehicle, where all processing is performed at the vehicle.

4.4. Puppet Mode Implementation

In the puppet mode configuration, steps (C) through (H) were shifted to the remote server, as illustrated in Figure 7.

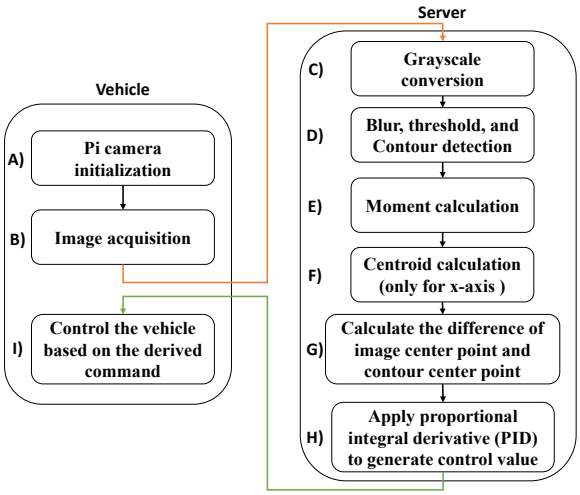


Figure 7. Puppet mode, where the computation is offloaded to the remote control server.

4.5. Edge Computing Mode Implementation

Three distinct levels of edge computing modes were identified, each differentiated by the extent of partial offloading. In edge computing level one, the steps up to contour detection (stage D) were executed on the mobile robot, as presented in Figure 8a. Steps (F) through (H) and (G) through (H) were offloaded to the server in edge computing levels two (Figure 8b) and three (Figure 8c), respectively.

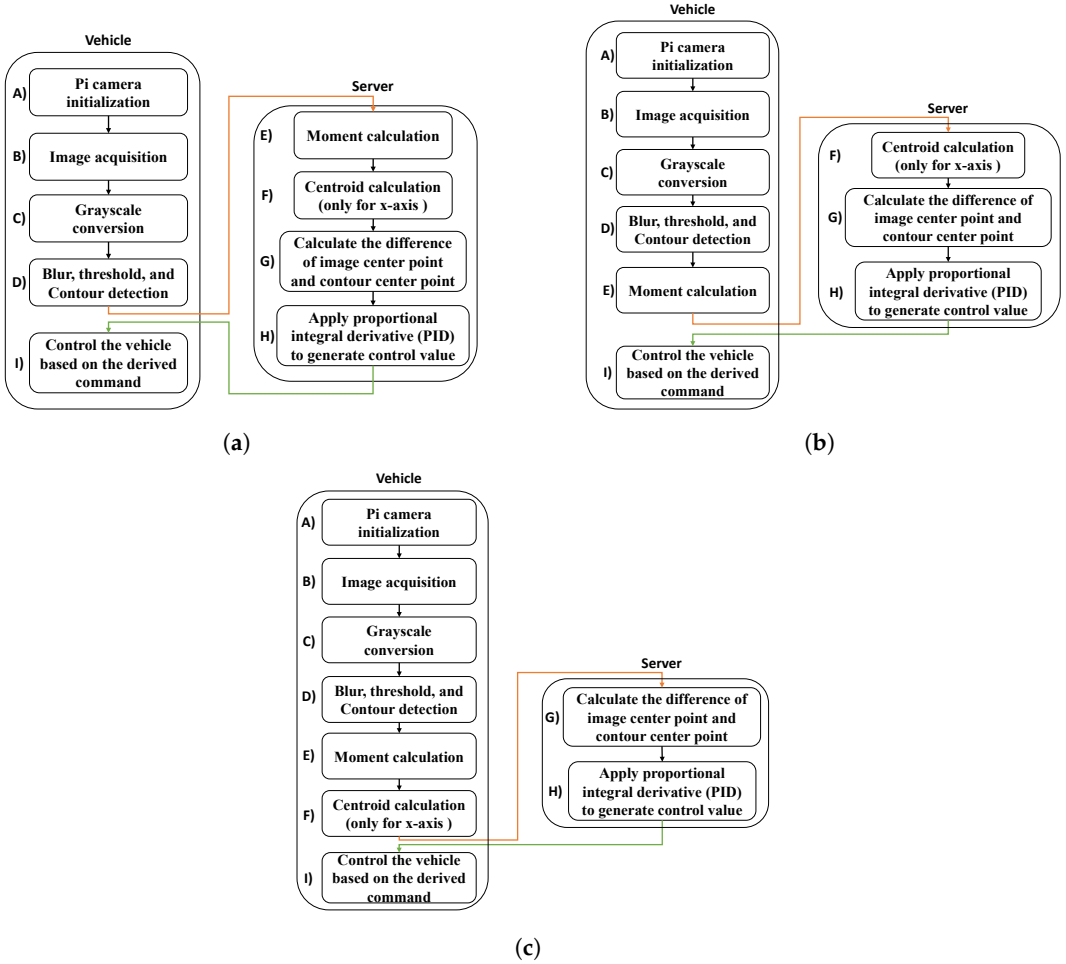
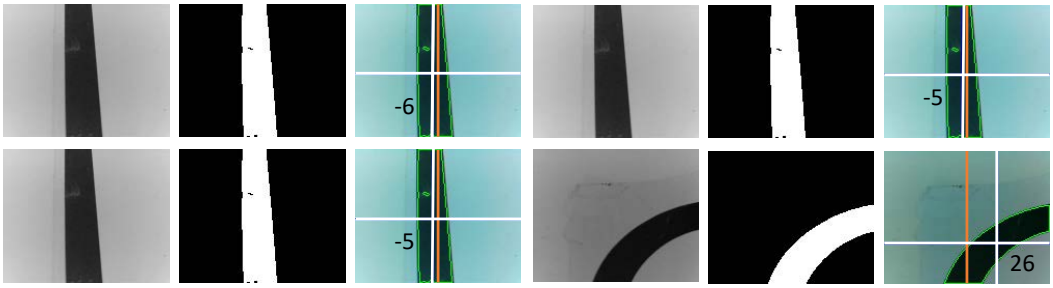


Figure 8. Illustration of the edge computing mode. In this mode, a portion of the computational function is executed on the edge device, facilitating data preprocessing prior to transmission to the

remote server. The depicted edge computing level varies based on the extent to which the data are processed prior to transmission. (**a**) Edge computing level 1; (**b**) Edge computing level 2; (**c**) Edge computing level 3.

4.6. Adaptive Mode Implementation

In the system under consideration, the system accuracy was evaluated based on the deviation from the centerline. The algorithm calculated the difference between the previous and current center points; if this difference was less than or equal to a certain threshold, communication was bypassed. However, if the difference exceeded the threshold, communication was initiated. Figure 9a provides a comprehensive depiction of the adaptive scheme's operation in instances where the feature difference between the current image and previous images did not surpass the threshold. As demonstrated in Figure 9a, the difference between the two captured images was a mere one pixel; thus, the control command from the previous image was utilized, and the current image was not dispatched for execution to generate a new control command. Figure 9b presents a scenario in the adaptive scheme where the feature difference between the current image and the previous images exceeded the threshold. In Figure 9c, the difference between the two consecutive images was 31 pixels, which was beyond the threshold, prompting the image to be transmitted to the server for the generation of a new control command.



(**a**) (**b**)

Figure 9. Adaptive scheme example. (**a**) Adaptive scheme operation. Feature extraction and comparison when the image will not be transmitted to the server. (**b**) Adaptive scheme operation. Feature extraction and comparison when the image will be transmitted to the server.

In order to determine the optimal threshold value, various threshold values were tested. The results, based on the reduction in the communication volume and system accuracy, were utilized to select the most suitable threshold value through inspection. From the experiments, a threshold value of two pixels was found to be optimal; hence, this value was employed in our proposed adaptive scheme.

We also propose Algorithm 1 for updating the threshold. We assumed that the system operated on a specific cycle, during which a decision was made at regular intervals about whether or not to transmit data. Each repetition of the same decision was considered an *event*. If the difference, ϵ, between features on an image was smaller than a certain threshold, θ, the image was not transmitted. The duration of this non-transmission status was counted during each system cycle, a process referred to as event counting. If the recurring event exceeded a predetermined period, t, the value of θ was decreased to increase the system sensitivity.

Algorithm 1 Update the threshold for the adaptive mode.

Previous center point: C_{t-1}
Current center point: C_t
$\epsilon = |C_t - C_{t-1}|$

if $\epsilon < \theta$ **then**
    do not send
    count number of consecutive events
    if consecutive events are exceeding t **then**
        $\theta = \theta - 1$
    end if
else
    send
    count number of consecutive events
    if consecutive events are exceeding t **then**
        $\theta = \theta + 1$
    end if
end if

Conversely, during image data transmission, the frequency of the consecutive image transmission occurrences was tracked. If the frequency of consecutive transmissions exceeded a predefined period, t, the value of θ was increased to decrease the system sensitivity. It should be noted that the aforementioned periods, t, used for non-transmission and transmission scenarios may not necessarily be equal. This strategy enabled the system to adjust its sensitivity dynamically based on the observed transmission patterns.

4.7. Results and Discussion

Several experiments were conducted to assess the tradeoff between the communication and computation effects at varying levels of edge computing. Our code is publicly available at https://github.com/sihsch/EC-LLC-WNCS, accessed on 21 July 2023. Video footage of the experiment can be found at https://youtu.be/UpQUypfKRns, accessed on 21 July 2023.

A critical performance metric for a WNCS is the timeliness of the command execution. This factor is quantified by measuring the command transaction time, defined as the duration from the point at which the remote vehicle senses its environment to the moment the vehicle receives and executes the control command.

Figure 10 depicts the command transaction time across different computational modes. From this figure, we derived several important conclusions. In puppet mode, where the mobile robot transmitted all its data to the remote server, we observed a longer response time compared to all the edge computing modes. Notably, as we progressed from edge level one to three, there was a consistent reduction in the command transaction time. This suggests that the system performance improved with the increasing edge computing levels, which can be attributed to the data size reduction through local preprocessing prior to transmission to the remote server. This indicates that offloading some computational workload to the actuator side can alleviate the communication overhead, thereby enhancing the overall WNCS performance. Considering the inherent instability of wireless channels, minimizing the communication load is crucial. Of all the communication methods assessed, the 5G yielded the best performance. While both the 5G and LTE showcased similar performance metrics, the 5G was consistently superior. However, the puppet mode performance notably deteriorated under 3G conditions.

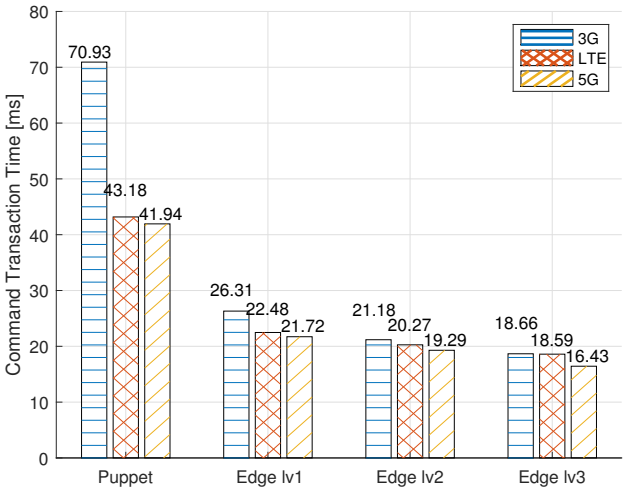


Figure 10. Representation of the command transaction time, defined as the duration from when the mobile robot senses its environment until it receives and subsequently executes the control command.

Table 3 provides a comparative overview of the data size communicated between the mobile robot and the remote control server across different modes. A significant influence of edge computing was evident, as its implementation resulted in a marked reduction in the data size, thereby streamlining the data transfer from the mobile robot to the remote server. In particular, edge computing level one transmitted only 3.42% of the data size compared to the puppet mode, indicating a substantial decrease merely through its application. Further reductions were observed at edge computing levels two and three, which transmitted a mere 0.81% and 0.03% of the puppet mode's data size, respectively. Therefore, it can be inferred that strategic utilization of edge computing can dramatically minimize the data size to be transmitted, boosting the overall efficiency.

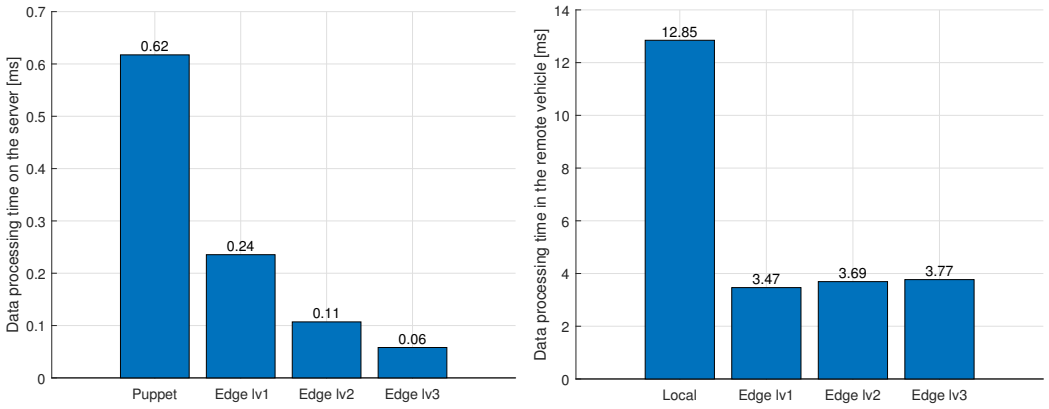
Table 3. Data size transmitted from the remote vehicle to the remote control server.

Puppet Mode	Edge Level 1	Edge Level 2	Edge Level 3
61,440 Bytes	2100 Bytes	495 Bytes	19 Bytes

The introduction of edge computing was confirmed to shorten the command transaction time by reducing the size of the transmitted data, thereby enabling stable passage through an unstable wireless channel. However, in computational terms, the introduction of edge computing leads to a decrease in throughput for the high-power server and an increase in computation on the mobile robot, which has relatively limited computing power, thereby resulting in a performance loss. Figure 11a illustrates a graph depicting the data processing time on the server. With the increasing levels of edge computing, the size of the data transmitted to the server decreased, leading to a corresponding decrease in the computation time. In contrast, as indicated in Figure 11b, the data processing time in the vehicle showed an upward trend as the edge computing level escalated.

Figure 12a presents the cumulative processing times across both locations, with edge computing level three exhibiting the longest delay among all the WNCS modes. Notably, the processing time in the local mode equaled the command transaction time due to the absence of communication; yet, this mode shows the lowest performance overall. Although edge computing level three demonstrated an impressive performance in the command transaction time, this advantage was only fully appreciated when considering the data transmission time, which Figure 12b showed to be the shortest for this mode. We introduce the concept of normalized data transmission time, which represents the data transmission time divided by the total data processing time, effectively translating to the transmission time per unit of computation time (1 s). This metric reveals the transmission time required per second of computation time consumed. As illustrated in Figure 12c, edge computing

level three emerged as the most efficient. Figure 12d mirrors Figure 12c but omits the puppet mode to emphasize the differences among the edge computing modes more clearly. In light of these findings, it is clear that the advantages of the reduced transmission time and the decreased traffic, achieved through the application of edge computing, far surpassed the potential increase in the computational time.



(**a**) (**b**)

Figure 11. Data processing time on the server and in the vehicle. (**a**) Data processing time on the server; (**b**) Data processing time in the vehicle.

While the data rate can be accurately and quantitatively calculated, and the best performing mode can be derived through mathematical analysis in a non-wireless communication channel, such analysis becomes more complex in a wireless environment, where data rates unavoidably fluctuate due to retransmission. This necessitates a mathematical analysis that we leave for future research. Instead, we propose an innovative approach to enhance the performance, the results of which are discussed in relation to the adaptive mode below.

As previously outlined, the adaptive mode experiments were conducted based on edge computing level three and 5G networks, given their superior performance. In adaptive mode, the outcome of stage (F) in edge computing level three, i.e., the x-coordinate of the road center, was compared to the preceding value. The data were not transmitted when the difference was less than or equal to the threshold value (θ). This strategy effectively diminished the communication load. A threshold $\theta = 1$ implies that data were not transmitted when the difference between the current and preceding value was less than or equal to 1 pixel. To evaluate the adaptive mode's performance, two performance indicators were utilized: deviation and the transmission ratio. The transmission ratio signified the proportion of the new image acquisitions to the feature transmissions. As presented in Figure 13a, the deviation escalated as the threshold increased. However, as Figure 13b illustrates, the transmission ratio significantly decreased with an increasing threshold. The adaptive scheme cut the total number of transmissions by nearly 20%. A high deviation implies a greater likelihood of remote control failure. If $\theta \geq 3$, the WNCS fails to function properly. Therefore, we recommend $\theta = 2$, as it substantially lessens the communication load while ensuring optimal remote control functionality. Nevertheless, the choice of threshold depends on the target application. We proposed Algorithm 1 that adjusts the threshold. Algorithm 1 resulted in more significant deviation, yet ensured successful remote control and reduced the transmission ratio. Given that the adaptive scheme is based on edge level three, the command transaction time remained consistent with that of edge level three.

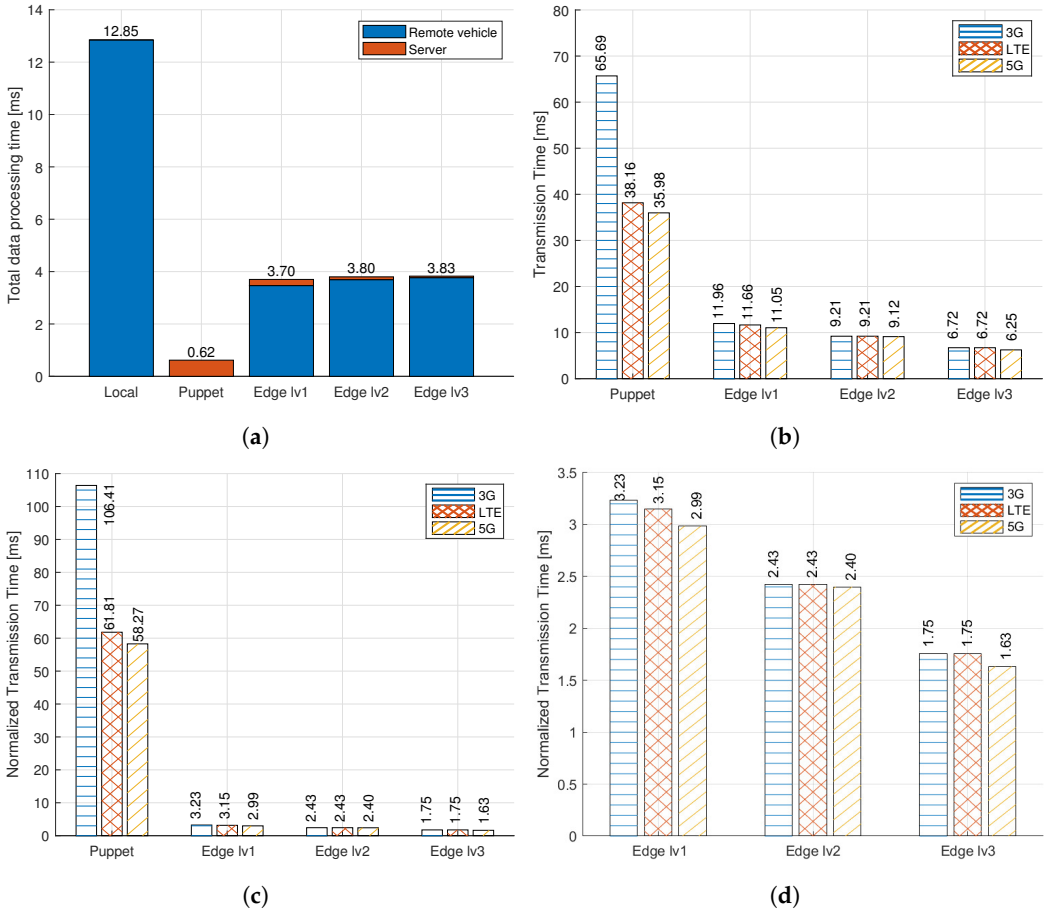


Figure 12. Total data processing time, data transmission time, and normalized data transmission time. (**a**) Total data processing time, the sum of the processing times in the vehicle and the server. (**b**) Data transmission time from the remote vehicle to the server. (**c**) Normalized data transmission time of the NCS modes. (**d**) Normalized data transmission time of the edge computing modes.

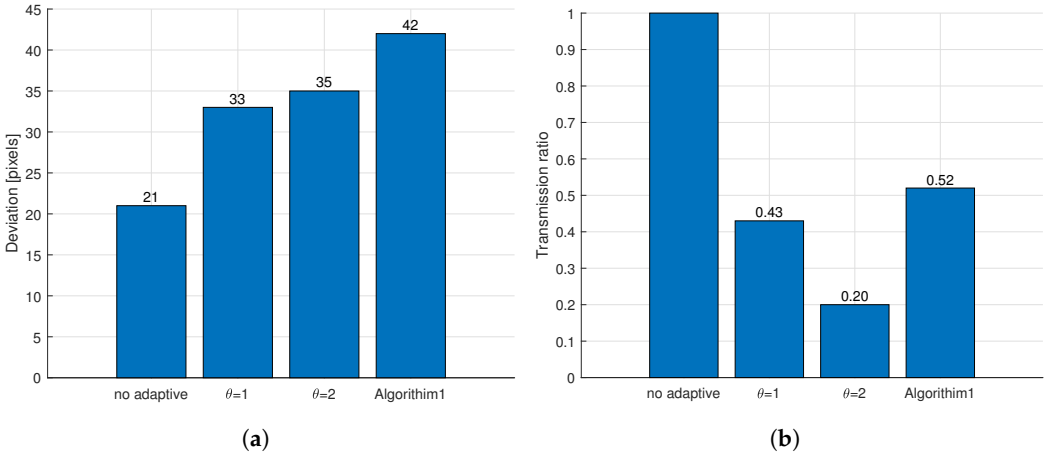


Figure 13. Performance in terms of the deviation and transmission ratio of various settings. (**a**) Deviation, i.e., the difference between the center point of the camera screen of the remote vehicle and the center point of the road; (**b**) Transmission ratio, i.e., the ratio of the number of times a new image is acquired to the number of times that a feature is transmitted for the image.

We pursued a comparative evaluation of our proposed methodology against [18]. To appraise our solution, we first adopted the WebRTC framework in our testbed. Following this, we gauged the transmission time in puppet mode while using the WebRTC for communication, contrasting this with the transmission time at the proposed edge comput-

ing mode with level three. As shown in Figure 14, across all network environments, the WebRTC improved the transmission time by approximately 58.49%, 59.74%, and 74.74% when compared to the puppet mode, respectively. On top of this, the proposed method improved the transmission time compared to the WebRTC by approximately 59.57% in a 3G environment, 56.32% in an LTE environment, and 58.16% in a 5G environment. So, it can be concluded that the proposed method outperformed the WebRTC-based remote control techniques in terms of the transmission time across all network environments.

All the experimental outcomes affirmed that our proposed method enhances the delay performance of a WNCS and demonstrates the feasibility of low-latency control.

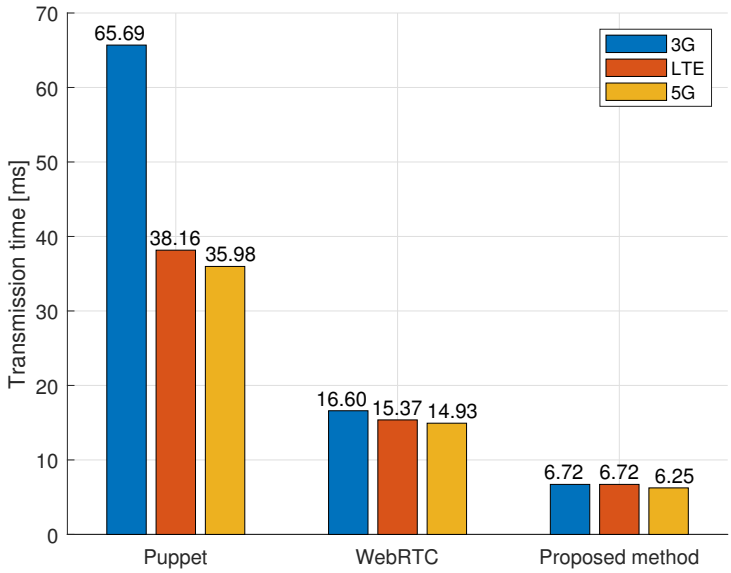


Figure 14. Performance comparison with WebRTC and our proposed method.

5. Limitations of the Proposed Approach

The utilization of edge computing to address the uncertain communication environment characteristic of WNCS sets our approach apart from others, which are primarily focused on tackling the limited computational capabilities. Nevertheless, the inherent limitations of edge devices, including constrained computing power, restricted memory, and limited storage capacity, curtail the scope and complexity of the computations that can be performed at the edge. Further, striving for increased computing power in these devices invariably impacts the energy efficiency, potentially leading to reduced battery life or increased energy consumption. This requires striking a delicate balance between enhanced computational capabilities and energy conservation. It should be noted, however, that our study did not take these aspects into account. Additionally, we did not propose a solution from a communication perspective to handle changing channels, a characteristic inherent to wireless communication. Further research is warranted to address these limitations of our study.

6. Conclusions

As data sizes within control systems and communication frequencies escalate, system performance risks significant degradation due to network-induced delays and local device processing. In this study, we proposed an innovative WNCS that harnesses computational functionality for data preprocessing prior to transmission on the device. The device, instead of forwarding all the information, transmits the extracted features to the remote server. By comparing the current and previous states and engaging in communication only when deemed essential for state alteration, we effectively reduce the communication overhead. This concept of adaptive communication contrasts with the traditional approach of periodic communication between two endpoints.

Our testbed experiments convincingly demonstrated that this methodology achieved significant reductions in communication workload while maintaining satisfactory control performance. Leveraging edge computing, our experiments revealed that only 3.42% of the raw data needed to be transmitted, and the adaptive scheme successfully reduced the transmission frequency by 20%, all without compromising the control performance. Furthermore, the experimental results confirmed that our approach outperformed the WebRTC-based remote control, achieving a remarkable 58.16% reduction in latency in a 5G environment. These experimental findings strongly support the effectiveness of our proposed strategy in improving the latency performance of wireless networked control systems (WNCSs). Although our assertions were substantiated through the implementation of a specific system, our findings appear to have broader applicability to other types of WNCSs.

A promising area for future study involves an analysis related to energy consumption. While servers have access to a stable power supply, remote vehicles operate on battery power. Consequently, an increase in edge computing results in greater battery usage, thereby reducing the operational time of the remote vehicle. To build a more realistic WNCS, simultaneous consideration of low-delay communication and energy consumption may be beneficial.

Author Contributions: Conceptualization, D.M.K.; methodology, D.P.M.; formal analysis, D.M.K.; writing—original draft preparation, D.P.M.; writing—review and editing, D.M.K. and D.P.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2019R1G1A1100699). This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ICAN (ICT Challenge and Advanced Network of HRD) program (IITP-2023-2020-0-01832) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation). This work was supported by the Soonchunhyang University Research Fund.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Klügel, M.; Mamduhi, M.; Ayan, O.; Vilgelm, M.; Johansson, K.H.; Hirche, S.; Kellerer, W. Joint cross-layer optimization in real-time networked control systems. *IEEE Trans. Control Netw. Syst.* **2020**, *7*, 1903–1915. [CrossRef]
2. Pillajo, C.; Hincapié, R. Stochastic control for a wireless network control system (WNCS). In Proceedings of the 2020 IEEE ANDESCON, Quito, Ecuador, 13–16 October 2020; IEEE: Piscataway, NJ, USA, 2020.
3. Huang, K.; Liu, W.; Li, Y.; Vucetic, B. To sense or to control: Wireless networked control using a half-duplex controller for IIoT. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019; IEEE: Piscataway, NJ, USA, 2019.
4. Wiens, R.C.; Maurice, S.; Robinson, S.H.; Nelson, A.E.; Cais, P.; Bernardi, P.; Newell, R.T.; Clegg, S.; Sharma, S.K.; Storms, S.; et al. The SuperCam instrument suite on the NASA Mars 2020 rover: Body unit and combined system tests. *Space Sci. Rev.* **2021**, *217*, 4. [CrossRef]
5. Khanh, Q.V.; Hoai, N.V.; Manh, L.D.; Le, A.N.; Jeon, G. Wireless communication technologies for IoT in 5G: Vision, applications, and challenges. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 3229294. [CrossRef]
6. Sun, J.; Yang, J.; Zeng, Z. Predictor-based periodic event-triggered control for nonlinear uncertain systems with input delay. *Automatica* **2022**, *136*, 110055. [CrossRef]
7. Akasaka, H.; Hakamada, K.; Morohashi, H.; Kanno, T.; Kawashima, K.; Ebihara, Y.; Oki, E.; Hirano, S.; Mori, M. Impact of the suboptimal communication network environment on telerobotic surgery performance and surgeon fatigue. *PLoS ONE* **2022**, *17*, e0270039. [CrossRef]
8. Zhang, B.L.; Han, Q.L.; Zhang, X.M.; Yu, X. Sliding mode control with mixed current and delayed states for offshore steel jacket platforms. *IEEE Trans. Control Syst. Technol.* **2014**, *22*, 1769–1783. [CrossRef]
9. Saleem, U.; Liu, Y.; Jangsher, S.; Tao, X.; Li, Y. Latency minimization for D2D-enabled partial computation offloading in mobile edge computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 4472–4486. [CrossRef]
10. Almutairi, J.; Aldossary, M. Modeling and analyzing offloading strategies of IoT applications over edge computing and joint clouds. *Symmetry* **2021**, *13*, 402. [CrossRef]
11. Ma, X. Optimal control of whole network control system using improved genetic algorithm and information integrity scale. *Comput. Intell. Neurosci.* **2022**, *2022*, 9897894. [CrossRef] [PubMed]

12. Huang, K.; Liu, W.; Shirvanimoghaddam, M.; Li, Y.; Vucetic, B. Real-time remote estimation with hybrid ARQ in wireless networked control. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 3490–3504. [CrossRef]
13. Cho, B.M.; Jang, M.S.; Park, K.J. Channel-aware congestion control in vehicular cyber-physical systems. *IEEE Access* **2020**, *8*, 73193–73203. [CrossRef]
14. Gatsis, K.; Pajic, M.; Ribeiro, A.; Pappas, G.J. Opportunistic control over shared wireless channels. *IEEE Trans. Autom. Control* **2015**, *60*, 3140–3155. [CrossRef]
15. Huang, K.; Liu, W.; Li, Y.; Vucetic, B.; Savkin, A. Optimal downlink–uplink scheduling of wireless networked control for Industrial IoT. *IEEE Internet Things J.* **2019**, *7*, 1756–1772. [CrossRef]
16. Demirel, B.; Gupta, V.; Quevedo, D.E.; Johansson, M. On the trade-off between communication and control cost in event-triggered dead-beat control. *IEEE Trans. Autom. Control* **2016**, *62*, 2973–2980. [CrossRef]
17. Zeng, T.; Mozaffari, M.; Semiari, O.; Saad, W.; Bennis, M.; Debbah, M. Wireless communications and control for swarms of cellular-connected UAVs. In Proceedings of the 2018 52nd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 28–31 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 719–723.
18. Tiberkak, A.; Hentout, A.; Belkhir, A. WebRTC-based MOSR remote control of mobile manipulators. *Int. J. Intell. Robot. Appl.* **2023**, *7*, 304–320. [CrossRef]
19. Gong, C.; Lin, F.; Gong, X.; Lu, Y. Intelligent cooperative edge computing in internet of things. *IEEE Internet Things J.* **2020**, *7*, 9372–9382. [CrossRef]
20. Premsankar, G.; Di Francesco, M.; Taleb, T. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [CrossRef]
21. Li, H.; Ota, K.; Dong, M. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. *IEEE Netw.* **2018**, *32*, 96–101. [CrossRef]
22. Anzanpour, A.; Amiri, D.; Azimi, I.; Levorato, M.; Dutt, N.; Liljeberg, P.; Rahmani, A.M. Edge-Assisted Control for Healthcare Internet of Things: A Case Study on PPG-Based Early Warning Score. *ACM Trans. Internet Things* **2021**, *2*, 1–21. [CrossRef]
23. Damigos, G.; Lindgren, T.; Nikolakopoulos, G. Toward 5G Edge Computing for Enabling Autonomous Aerial Vehicles. *IEEE Access* **2023**, *11*, 3926–3941. [CrossRef]
24. Koyasako, Y.; Suzuki, T.; Kim, S.Y.; Kani, J.I.; Terada, J. Motion control system with time-varying delay compensation for access edge computing. *IEEE Access* **2021**, *9*, 90669–90676. [CrossRef]
25. Shahhosseini, S.; Anzanpour, A.; Azimi, I.; Labbaf, S.; Seo, D.; Lim, S.S.; Liljeberg, P.; Dutt, N.; Rahmani, A.M. Exploring computation offloading in IoT systems. *Inf. Syst.* **2022**, *107*, 101860. [CrossRef]
26. Zhang, K.; Mao, Y.; Leng, S.; Zhao, Q.; Li, L.; Peng, X.; Pan, L.; Maharjan, S.; Zhang, Y. Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks. *IEEE Access* **2016**, *4*, 5896–5907. [CrossRef]
27. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3590–3605. [CrossRef]
28. Hossain, M.S.; Nwakanma, C.I.; Lee, J.M.; Kim, D.S. Edge computational task offloading scheme using reinforcement learning for IIoT scenario. *ICT Express* **2020**, *6*, 291–299. [CrossRef]
29. Wang, K.; Xiong, Z.; Chen, L.; Zhou, P.; Shin, H. Joint time delay and energy optimization with intelligent overclocking in edge computing. *Sci. China Inf. Sci.* **2020**, *63*, 140313. [CrossRef]
30. Shao, J.; Zhang, J. Communication-Computation Trade-off in Resource-Constrained Edge Inference. *IEEE Commun. Mag.* **2020**, *58*, 20–26. [CrossRef]
31. Zhang, W.; Wen, Y.; Guan, K.; Kilper, D.; Luo, H.; Wu, D.O. Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Trans. Wirel. Commun.* **2013**, *12*, 4569–4581. [CrossRef]
32. Shakarami, A.; Shahidinejad, A.; Ghobaei-Arani, M. An autonomous computation offloading strategy in Mobile Edge Computing: A deep learning-based hybrid approach. *J. Netw. Comput. Appl.* **2021**, *178*, 102974. [CrossRef]
33. Carvalho, G.; Cabral, B.; Pereira, V.; Bernardino, J. Computation offloading in Edge Computing environments using Artificial Intelligence techniques. *Eng. Appl. Artif. Intell.* **2020**, *95*, 103840. [CrossRef]
34. Chen, M.; Wang, T.; Zhang, S.; Liu, A. Deep reinforcement learning for computation offloading in mobile edge computing environment. *Comput. Commun.* **2021**, *175*, 1–12. [CrossRef]
35. Ju, Y.; Chen, Y.; Cao, Z.; Liu, L.; Pei, Q.; Xiao, M.; Ota, K.; Dong, M.; Leung, V.C.M. Joint Secure Offloading and Resource Allocation for Vehicular Edge Computing Network: A Multi-Agent Deep Reinforcement Learning Approach. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 5555–5569. [CrossRef]
36. Gao, H.; Huang, W.; Liu, T.; Yin, Y.; Li, Y. PPO2: Location Privacy-Oriented Task Offloading to Edge Computing Using Reinforcement Learning for Intelligent Autonomous Transport Systems. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 7599–7612. [CrossRef]
37. Ma, X.; Xu, H.; Gao, H.; Bian, M. Real-time multiple-workflow scheduling in cloud environments. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4002–4018. [CrossRef]