Article

Design of a Generic Wearable-Robot Monitoring System (GWRMS) for Patients with Lower Limb Paralysis

Ho-Won Lee ¹ , Kyung-Oh Lee ¹, In-Ae Kang ¹, Joong-Eup Kye ² and Yoon-Yong Park ^{1,*}

¹ Department of Computer and Electronics Convergence Engineering, Sunmoon University, Asan-si 31460, Korea; kck4156@gmail.com (H.-W.L.); leeko@sunmoon.ac.kr (K.-O.L.); inae5004@sunmoon.ac.kr (I.A.K.)

² Department of Electronic Engineering, Sogang University, Seoul 04107, Korea; jekye@sogang.ac.kr

* Correspondence: yypark@sunmoon.ac.kr; Tel.: +82-10-8803-2365

Received: 11 June 2020; Accepted: 9 August 2020; Published: 11 August 2020

Abstract: The development of wearable robotic technology has accelerated recently due to active research in industrial technology. Wearable robots are used to help patients with lower-body palsy to walk. This paper reports a generic wearable-robot monitoring system (GWRMS) that supports resource monitoring and status prediction. The GWRMS offers universal support for operating systems that comply with the Portable Operating System Interface (POSIX) application programming interface (API), reducing limitations that arise due to the choice of operating system and processor. This system improves optimisation of a program operating in a robot-embedded system by detecting overhead in the software process of the wearable robot. In addition, it operates using TCP-socket-based monitoring on an on-demand basis only.

Keywords: wearable robot; resource monitoring system; software optimisation; machine-learning; low-powered processor

1. Introduction

Robots comprise a collection of embedded systems whose performance is sometimes limited by the requirement for high portability and mobility [1]. In addition, in some cases, embedded systems need to guarantee real-time performance. To address these requirements, previous studies have demonstrated methods such as kernel design to guarantee real-time performance [2,3]. Improvement of the performance of embedded systems can also be achieved through code analysis and optimisation.

The objectives of this study were to optimise the code for wearable robots and to predict their status by developing a generic wearable-robot monitoring system (GWRMS). The robot described in this paper assists with walking in patients with lower-body palsy. Since such patients can walk only by using exoskeleton robots, software optimisation is critically important. This can be achieved by monitoring the system resources of the wearable robot.

The GWRMS also works with any operating system that meets Portable Operating System Interface (POSIX) [4] standards. POSIX is a set of operating system application programming interface (API) standards established by the IEEE for the purpose of developing highly portable UNIX applications by organising common APIs for different UNIX operating systems. Most UNIX-like operating systems, with the exception of Windows, also follow POSIX standards.

In addition, the GWRMS can monitor the current status of the robot by analysing various data generated by the robot and storing the data on a cloud server for later analysis. The GWRMS also provides an interface through which developers of wearable robots can efficiently change component parameters.

The resource-monitoring function of the GWRMS allows monitoring of embedded systems and optimisation of each program operating in the robot. In addition, the resource-monitoring system uses the resources of embedded systems effectively, allowing on-demand operations through TCP-socket communication.

After predicting the status of the wearable robot, data generated by the robot for analysis (e.g., upper-body tilt, plantar pressure, motor angle, etc.) are stored on a cloud server. Thus, it is possible to predict the state of the wearable robot and improve its accuracy simultaneously.

As detailed adjustments to each motor and sensor are essential in the development of wearable robots, the GWRMS described in this paper supports both the display of system resources and the adjustment of component parameters. Adjusting component parameters can help wearable-robot developers to add or change functions.

The remainder of this paper is organised as follows: Section 2 introduces related research; Section 3 presents each key component in the implementation of the GWRMS from the perspective of robots, servers, and clients; and Section 4 describes the merits of the GWRMS and its limitations.

2. Related Work

Several previous studies have examined optimisation of the performance of embedded systems [5,6]. The most general and straightforward way to improve their limited performance is through code analysis and optimisation. Most robots are based on specific platforms, such as Linux-based robot-control middleware systems, including the ROS open-source project [7,8] and the i platform provided by IBM [9]. Energy-monitoring systems that operate only on certain processor architectures, such as ARM or MIPS, have also been reported [10,11].

With the exception of Windows, which is not widely used in robot-embedded systems [12], most operating systems follow POSIX standards, but there have been no reports on general resource-monitoring systems that can be used with multiple operating systems and processors. The GWRMS reported in this paper consists only of components that can be used in operating systems conforming to POSIX standards, unlike previous research based on specific platforms with a limited range of operating-system choices.

The GWRMS discussed in this paper supports status monitoring by supporting machine-learning through component parameters generated by the wearable robot, as well as resource monitoring supported by other systems.

Table 1 compares the GWRMS with existing systems.

Table 1. Comparison between previous systems and the GWRMS.

	GWRMS	Overseer	Drums	Robot Console
Supported OS	All POSIX based OS	ROS and Nagios	ROS	IBM i platform
Visualisation	WPF (Server)	Nagios Dashboard	Graphite	Unknown
Server Type	TCP/IP	Apache HTTP	HTTP	Unknown
CPU Architecture	Non-dependent	Non-dependent	Non-Dependent	Dependent
Other	Extremely Lightweight Cross-platform Module-based	Lightweight	Feature-rich	Feature-rich Paid service

The GWRMS is compatible with a wider range of operating systems than other approaches, and it uses basic TCP/IP communication to reduce network overhead as much as possible and minimise system overhead. Unlike Overseer or Robot Console, the GWRMS supports multiple platforms and is designed to use fewer resources. The GWRMS also has lower CPU and memory requirements than previous systems.

The GWRMS is not dependent on CPU architecture or operating system, so it can be used by many types of embedded system. In many cases, monitoring systems cannot be used because operating systems and underlying CPU architectures differ.

Compared to Overseer, the GWRMS is not dependent on specific frameworks, such as ROS and Nagios, and it has low resource usage due to its use of simple programs that run on performance-constrained hardware. In comparison to Drums, the GWRMS has better support for embedded systems because Drums is also dependent on the ROS framework, as is Overseer.

The GWRMS is also lighter because it uses TCP/IP, which is a core component of HTTP, rather than an HTTP server for communication. The overhead required for data visualisation is also smaller because the GWRMS does not implement cloud-server-based visualisation on embedded hardware, unlike previous systems.

3. Materials and Methods

The GWRMS is designed to operate on any operating system that follows POSIX standards and can support the following components (libraries): Lua, LuaSocket, and GNU Software.

The wearable robot described in this paper currently uses LabView, a Linux-based operating system; Linux follows POSIX standards, and LabView is therefore compatible with the GWRMS.

To aid in robot development, such as the addition of functions, the GWRMS can output and modify data (component parameters) generated by the various hardware components of the robot.

The GWRMS consists of three elements: a wearable robot, a server, and a cloud-server, as shown in Figure 1. Each component is described below.

Figure 1. System overview of generic wearable-robot monitoring system (GWRMS).

3.1. Wearable Robot

Table 2 shows the configuration of the wearable robot. And Figure 2 shows data flow diagram showing the interactions between robot components and modules of the GWRMS.

Table 2. Components of the wearable robot that communicate with GWRMS.

OS	Components			
Linux	Motor	Servo	Skeleton	Etc..
Generic Wearable Robot Monitoring System (GWRMS)				
Resource-monitoring module				
Component-monitoring module				
Component-control module				
Status-monitoring module				
Communication module

Figure 2. Data flow diagram showing the interactions between robot components and modules of the GWRMS.

The wearable robot uses LabView, a Linux-based operating system, to control component parameters such as motors, servos, and hydraulics, as well as a GWRMS that is composed of a resource-monitoring module, a component-monitoring module, a component-control module, a status-monitoring module, and a communication module. The roles of each component are as follows.

The robot in this paper is an exoskeleton-type robot that encloses the lower body. The GWRMS is composed of various modules and supports the POSIX operating system, so it can easily be used with various robot types. As most robot monitoring systems rely on specific platforms not included in the operating system, such as ROS, support for various types of embedded system hardware is insufficient. However, many robotic systems still use Linux, which is also POSIX-compliant, so the GWRMS can easily be ported to most operating systems.

The resource-monitoring module reports the CPU, memory, and processing resource usage of the wearable robot to the server. The component-monitoring module reports the state of the component parameters of the wearable robot, such as upper-body tilt, plantar pressure, and motor angle, to the server. The component-control module is responsible for application of individual control modifications sent from the server to the robot.

The status-monitoring module is responsible for collecting and reporting the data necessary for predicting the current status of the wearable robot to the server. The status-monitoring module collects data from the resource-monitoring module and the component-monitoring module, and then it organises and reports the data required to predict the status of the wearable robot to the communication module.

TCP/IP socket communication management between the server and the wearable robot is processed by the communication module. In addition, when monitoring data are requested from the server, the communication module activates other modules of the GWRMS and transmits the data from each module to the server via a TCP/IP socket connection.

Wearable robots use embedded hardware that has low processing power. Therefore, the operation of a GWRMS should not generate a level of overhead that would interfere with the normal operation of the wearable robot. Accordingly, in a GWRMS, the program used to operate the robot should be a daemon type program, and the system resources of the wearable robot should not be used when monitoring is not in progress.

Regardless of the hardware specifications of the device using the GWRMS, it enters an idle state while waiting for TCP/IP communication. Therefore, it generates little overhead. In addition, even when collecting data while not in an idle state, only necessary information is collected and

transmitted. All operations associated with heavy CPU-resource requirements are performed on the server, so even in a non-idle state only a small overhead is generated. This is discussed in detail in Section 4.

To optimise the efficiency of the GWRMS, the communication module of the wearable robot's software uses a TCP socket to monitor data on demand instead of conducting ongoing monitoring. As shown in Table 3, this method improves the performance of wearable robots' embedded processors.

Table 3. Comparison between on-demand and ongoing monitoring.

On-Demand Monitoring	Ongoing Monitoring
Idle when monitoring is not needed	Maintains continuous monitoring
Uses resources only when needed	Always uses system resources
Monitoring is invoked only by TCP request	Maintains monitoring without requests

When monitoring proceeds through TCP socket communication, the communication module collects data from each module, packs it as JSON string data, and sends it as a response to a TCP/IP request. The GWRMS utilises the functions of the following GNU software [13]: mpstat, ps, free.

The daemon software on the wearable robot of the GWRMS was implemented with Lua, and TCP socket communication was implemented with LuaSocket. Part of the pseudo code for the communication module is shown Algorithm 1.

Algorithm 1. Pseudo code for the communication module of the GWRMS daemon software.

```

set tcp socket with ip and port

while true do
  init client = accepted tcp session
  client timeout = 0
  init partial = client data
  if partial is valid then
    init processes = {}
    init process = result of shell execution result "ps -a -o comm,pid,%cpu,sz | awk {'print
$1\", \"$2\", \"$3\", \"$4'}"
    init tbl = split table of processes by new line character

    for i = 2 to size of tbl do
      init data = split table of i th element of tbl by comma character
      push table to processes by {name=data[1], pid=data[2], cpu=data[3], mem=data[4]}
    endfor

    init data = json encoded string by process
    init datas = {}
    init dataSize = 8000

    while true do
      init doNext = data:len() > index
      init targetIndex = minimum value between index and length of data
      push string to datas by subbed string between targetIndex-(dataSize-1) and targetIndex
      increment index by dataSize
      if doNext not equal true then
        break the loop
      endif
    endwhile
  endwhile

```

```

send data length to client
close client session
for i = 1 to length of data do
    set client = accepted tcp session
    client timeout = 0
    set partial = client data
    if partial not equal nil then
        send i th string of data to client
        close client
    endif
endfor
endif
endwhile

```

The pseudo code creates a TCP socket session and receives the session corresponding to the request when executing the request from the TCP socket. Then, after executing and collecting the command specified in the shell, a new session is created. The length of the message that can be sent in one session is limited, so the message length is first sent to the server. The entire message is then sent and the session ended. The server then receives the message until the message length specified at the creation of the session is satisfied.

3.2. Server

Among the components of the GWRMS, the processing modules of the server comprise resource-processing, component-processing, status-prediction, and communication modules, as shown in Table 4 and Figure 3.

Table 4. GWRMS server components.

Generic Wearable-Robot Monitoring System (GWRMS)	
Resource-processing module	
Component-processing module	
Status-prediction module	
Communication module

Figure 3. Data flow diagram showing the interactions between server components.

The resource-processing module unpacks the CPU, memory, and resource usage of each process received via the JSON string package through the communication module from each of the robot's processors, and it displays the data in the GUI client, as shown in Figure 4.

Figure 4. GWRMS server GUI client.

The component-processing module unpacks component data received from the communication module and displays the data in the GUI client, as shown in Figure 5. In addition, using the GUI buttons shown in Figure 5, component parameters can be modified, as shown in Figure 6 (e.g., adjustments to step length, cycle, toe height, and upper-body angle). This will assist in future development of the wearable robot, such as added functions.

Figure 5. Schematic of the wearable robot's component parameters based on component-processing-module results.

Figure 6. Adjustment of the wearable robot's component parameters based on the results of the component-processing module.

The status-prediction module unpacks data received from the communication module, analyses the data in conjunction with big-data on the cloud server, and shows the robot's current status in the GUI client's status preview, as shown in Figure 4.

The communication module requests monitoring data from the wearable robot and relays the data from each module to the corresponding server module. In addition, it uploads processed data to the cloud server and adds these to the big-data used for status analysis by the monitoring module. When component parameters are modified by the component-processing module, modified data are sent to the wearable robot's component-control module.

The GWRMS monitors the wearable robot's data and sends them to the server on request. The server processes the data and displays the resource status of the robot. If the user makes a request, the server sends the request to the wearable robot for execution. In addition, the server collects the data generated by the robot, sends the data to the cloud, analyses the data in light of big-data analysis, and displays the current status of the wearable robot based on the analysis.

If a machine-learning algorithm is executed based on the data generated by the wearable robot, the data used for analysis are uploaded back to the cloud server, and the learning data are reflected in the machine-learning algorithm. This feedback learning structure is shown in Figure 7.

Figure 7. Schematic of feedback learning structure.

Among the modules of the GWRMS, the status-prediction module has the advantage that data processed for status prediction are incorporated in the big-data database of the cloud server through a feedback learning structure. The data analysis process is shown in Figure 8.

Figure 8. Schematic of the machine-learning model of the status-prediction module.

The status-prediction module analyses the data generated by the robot's status-monitoring module based on the machine-learning model and displays the predicted status through the server's GUI client, as shown in Figure 4. The machine-learning algorithm of the GWRMS is run on the server, so there is no associated overhead in the robot system. As mentioned above, the robot only transmits and receives data to minimise overhead, and it collects and processes essential data after receiving it.

Each component of the GUI client includes a status preview (Figure 9) that shows the current status of the robot's cloud settings (Figure 9). Cloud server data can be manually uploaded and

downloaded to keep the data up to date, and a resource monitor (Figure 10) shows the resource status of the wearable robot.

Figure 9. Status preview and cloud settings of the GUI client.

Figure 10. Resource monitor of the GUI client.

The PID(Process ID) and name of each process can be checked via the resource monitor's per-process information button, as can CPU and memory usage, as shown in Figure 11. In addition, each item of information can be sorted in ascending or descending order.

PER PROCESS INFORMATION			
PID	NAME	CPU USAGE	MEMORY USAGE
12	bash	1.2	6MB
33	ps	0.2	12MB
113	neofetch	10.5	45MB
135	find	7.1	32MB
141	free	3.4	10MB
155	mpstat	5.6	8MB
167	git	6.2	11MB
188	apt	7.3	15MB

Figure 11. Per-process information of the GUI client.

3.3. Cloud Server

The cloud server component of the GWRMS receives and stores data generated by the robot via the GWRMS server. Data generated by the robot are incorporated in the big-data used to predict the state of the robot. The GUI client displays the status data via the process shown in Figure 8 and the machine-learning algorithm on the server as shown in the bottom right of Figure 4. Figure 12 provides a schematic diagram of the cloud server.

Figure 12. Schematic diagram of the cloud server [14].

The cloud consists of the cloud API and the public cloud. The cloud API supports uploading to, downloading from, and browsing of the public cloud, and it communicates with the GWRMS server's communication module.

4. Experimental Results

This section covers overhead and network latency when running the resource monitoring system of the GWRMS on a relatively low-end embedded system, Raspberry PI 4. Table 5 shows the specifications of the robot client system, Raspberry Pi 4, and the server system.

Table 5. System specifications.

Raspberry Pi 4			Razer Blade Base 15 (2019)		
CPU	RAM	Storage	CPU	RAM	Storage
BCM2711	4 GB	Micro SD	i7-9750H	32 GB	Samsung 970 Pro 512 GB

Figure 13 shows the results for the case where local and remote network communication are performed 1000 times with regard to network delay, when the server requests data from the robot system with a wired connection.

Figure 13. Network delay between robot and server.

As shown in Figure 13, after measuring the network communication delay between the server and the robot client 1000 times, general network delays (average of 31.984 and 35.099 ms in local and remote network communications, respectively) were observed regardless of the type of network [15].

Table 6 shows the results regarding GWRMS daemon overhead in the robot system. Average memory usage of the GWRMS daemon was 3398 KB, and average CPU usage was 0.7% per second when each module was operated according to the TCP/IP message request received from the server.

Table 6. Overhead generated by GWRMS daemon software.

Average CPU Usage Per Second	CPU Usage When Idle	Average Memory Usage
0.7%	0.001%	3398 KB

Even in systems such as Overseer that have a small amount of overhead, there is a CPU usage gap of more than 2% between idle and running states. Therefore, the GWRMS daemon, which has a CPU usage of 0.7% while running, has a very low overhead regardless of whether it is in an idle state or not. In addition, Overseer uses 30 MB of memory even when monitoring is not being performed. In contrast, the GWRMS uses an average of 3 MB of memory, which means that it has a lower system overhead even with regard to memory usage.

5. Conclusions

In this paper, to improve the performance available from the limited specifications of a wearable robot, code optimisation was performed using a GWRMS. Existing monitoring systems are platform-specific, and no universal monitoring systems are available. The GWRMS introduced in this study can be used with any operating system that conforms to POSIX standards, and it is not dependent on a specific process, architecture, or platform.

As mentioned in related work, other systems are dependent on specific platforms, and it takes time and effort to port to an operating system or CPU architecture that is not supported by the platform. In contrast, the GWRMS can easily be ported to operating systems that meet POSIX standards. In addition, as mentioned in the Experimental Results section, the GWRMS is lightweight compared to other systems.

The system described in this study also monitors the status of the wearable robot. Future work should lead to improvements that allow status monitoring to be used to anticipate dangerous situations by determining abnormal conditions, causing the robot to shut down and thus preventing injury to the user.

The GWRMS is highly portable and supports many systems, unlike previous systems. In addition, the GWRMS utilises the most well-optimised software even when compared to previous studies that cite software optimisation as an advantage. Considering these advantages, the GWRMS can contribute to the productivity of the robotics industry.

Author Contributions: Conceptualization, H.-W.L., K.-O.L. and Y.-Y.P.; Formal analysis, H.-W.L., K.-O.L., I.-A.K. and Y.-Y.P.; Investigation, H.-W.L., K.-O.L., J.-E.K. and Y.-Y.P.; Project administration, H.-W.L.; Resources, H.-W.L.; Software, H.-W.L.; Supervision, K.-O.L., I.-A.K., J.-E.K. and Y.-Y.P.; Validation, H.-W.L., K.-O.L. and Y.-Y.P.; Writing—Original draft, H.-W.L.; Writing—Review & editing, H.-W.L., K.-O.L. and Y.-Y.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Technology Innovation Program (Robot Industry Core Technology Development Project) (20003914, Development and commercialization of motorized exoskeleton robots for the assistant of daily life exercise) funded By the Ministry of Trade, Industry & Energy(MOTIE, Korea).

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

References

1. Paulson, L.D. Low-power chips for high-powered handhelds. *Computer* **2003**, *36*, 21–23. [[CrossRef](#)]
2. Junior, J.C.; Renaux, D. Efficient monitoring of embedded real-time systems. In Proceedings of the Fifth International Conference on Information Technology: New Generations (itng 2008), Las Vegas, NV, USA, 7–9 April 2008; pp. 651–656.
3. Nasri, M.; Brandenburg, B.B. Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems (outstanding paper). In Proceedings of the 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Pittsburgh, PA, USA, 18–21 April 2017; pp. 75–86.

4. POSIX Standards 2017. Available online: <https://pubs.opengroup.org/onlinepubs/9699919799/> (accessed on 12 July 2020).
5. Newman, P.; Kotonya, G. A runtime resource-management framework for embedded service-oriented systems. In Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, Boulder, CO, USA, 20–24 June 2011; pp. 123–126.
6. Steiger, C.; Walder, H.; Platzner, M. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Trans. Comput.* **2004**, *53*, 1393–1407. [[CrossRef](#)]
7. Roman, F.; Maidana, R.G.; de Morais Amory, A. Overseer: A Multi Robot Monitoring Infrastructure. In Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Porto, Portugal, 29–31 July 2018; pp. 141–148.
8. Monajjemi, V.; Wawerla, J.; Vaughan, R. Drums: A middleware-aware distributed robot monitoring system. In Proceedings of the 2014 Canadian Conference on Computer and Robot Vision, Montreal, QC, Canada, 6–9 May 2014; pp. 211–218.
9. Losinski, C. Enhanced Resource Monitoring with Robot Console. 2018. Available online: <https://www.helpsystems.com/resources/articles/enhanced-resource-monitoring-robot-console> (accessed on 12 July 2020).
10. Shin, D.; Shim, H.; Joo, Y.; Yun, H.S.; Kim, J.; Chang, N. Energy-monitoring tool for low-power embedded programs. *IEEE Des. Test Comput.* **2002**, *19*, 7–17. [[CrossRef](#)]
11. Oltu, O.; Voiculescu, V.; Popescu, V.; Dascalu, M.; Barbilian, A. Advanced power monitoring of an ARM processor. In *Computational Engineering In Systems Applications*; Book Series: Mathematics and Computers in Science and Engineering; WSEAS: Heraklion, Greece, 2008; pp. 243–246.
12. Jabeen, Q.; Khan, F.; Hayat, M.N.; Khan, H.; Jan, S.R.; Ullah, F. A survey: Embedded systems supporting by different operating systems. *arXiv* **2016**, arXiv:1610.07899.
13. GNU Software. Available online: <https://www.gnu.org/software/software.html> (accessed on 12 July 2020).
14. Chae, Y.J.; Park, Y.Y.; Lee, K.O.; Kim, S.Y.; Lee, H.W. A Design of Data Processing Applications Based on Micro-service Architecture for Wearable Robot Platforms. *TEST Eng. Manag.* **2019**, *81*, 106–113.
15. What Is Latency? Available online: <http://www.plugthingsin.com/internet/speed/latency/> (accessed on 12 July 2020).

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).