Pervasive Service Discovery Implementation Using UDP Protocol on Raspberry Pi and MyRIO

Mochammad Hannats Hanafi Ichsan, Wijaya Kurniawan, and Joniar Dimas Wicaksono
Computer Engineering Department, Brawijaya University, Malang, Indonesia
wjaykurnia@ub.ac.id

***Abstract*— Smart home environment is an environment by which there are equipments that are able to communicate with each other and can be monitored or controlled remotely through the internet. Nowadays, it still requires complex configuration to achieve those requirements. Pervasive computing is a method, which facilitates humans to ease configuring the devices. Based on previous researches that designed and tested the Pervasive system using UDP and LabVIEW on Personal Computer (PC), this research focused on implementing it on embedded systems, which are Raspberry Pi 3 as the host and NI MyRIOs as the clients. UDP protocol was used because it has lightweight attribute and does not require large memory. Several experiments have been done, such as measuring discovery time for each 86.62 bytes of data. Discovery time on the host was 56.417 ms, while the discovery on the client was 251.067 ms. Therefore, the whole discovery process was 313.417 ms. Whereas if the host fails, the time which client needs to reconnect was 10384.23 ms. When sending data testing between the host and client, the average data being send was 86.3 bytes, data transmission sensor took 58.26 ms, LED control took 5350.926 ms, and push button took 255.67 ms.**

***Index Terms*— MyRIO; Pervasive; Raspberry; Smart Home; UDP Protocol.**

I. INTRODUCTION

The smart home environment is an environment by which many equipments communicate each other [1] and they can be monitored or controlled remotely through the internet [2] for better human living[3]. With this technology, it eases us to monitor and control various equipments in the house such as electrical equipments [4], room's temperature [5], home securities [6], surveillance cameras [7], and so on. In the future, smart home is a choice to facilitate people's wellbeing using technology.

Currently, pervasive computing has been developed to facilitate the usage of connected devices without complex configuration [8], such as declaring types and functions of the devices, configuring the address of devices in network, or making relationships between devices. This technology makes it possible to enjoy each service facilitated by interconnected devices. Every task, job, or process will become easier, faster and more efficient because it is processed automatically [9].

The commonly used network protocols are the Transmission Control Protocol (TCP) [10] and User Datagram Protocol (UDP). Each of these protocols have its own advantages and disadvantages depending on the desired objectives. UDP is a lightweight protocol that can save memory and processor resources [11-13]. In a smart home environment, it is suitable to use UDP protocol since the data to be sent is small [14]. TCP protocol needs a three-way-handshaking process, causing traffic jams. Thus, processes need longer time to be completed [15].

Based on previous research, this pervasive system has already been designed in LabVIEW that works on Personal Computer (PC). The state machine, adopting in the research, gives the pervasive computing models of a communication between a host and a client. A research about state machine implementation between one host and more than one client has also been done and they were successfully tested and running well [16]. Another research was conducted by integrating the state machine with the cloud server [17]. In this research, the host communicate with the client and the communication between them is sent into the cloud server so that it can be observed and controlled via internet.

Despite of PC, this research emphasized implementing the pervasive computing on the embedded devices, specifically NI MyRIO and Raspberry Pi 3 devices. These embedded devices have an advantage by which they can be placed anywhere [18]. Besides, the other advantages of these two devices are: NI MyRIO already contains accelerometer sensor and it has the ease of adding other needed external sensors such as EEG, PIR, heat, rain, ultrasonic, infrared etc [19-21]. While Raspberry Pi 3 is a mini computer that does not require large electrical power [22]. In this research, NI MyRIO as the client acted as a sensor node and Raspberry Pi 3 acted as the host that has functions to store the detected sensor nodes around it. It is also used to monitor and control those sensor nodes. The system took all data communication and operated based on some predefined system requirements [23].

Based on the explanation above, this research proposed a technology that allows people to use devices that have the ability to know each other without complex configuration [23]. In this research, the Raspberry Pi 3 has already suceeded to recognize the active NI MyRIO sensor node devices around it. All these devices are connected via Wi-Fi on the local network using the UDP protocol. The used program is data-flow programming, namely LabVIEW, with the state machine method implemented on the devices. LINX LabVIEW Library is needed so that the LabVIEW program code can be downloaded on Raspberry Pi 3 device. All these are necessary to meet the requirements so that MyRIO and Raspberry Pi 3 can be used as representatives for smart home devices that have the ability to communicate each other.

II. SYSTEM DESIGN

This section explains the general description of the System Architecture and System Design. The System Architecture describes the topology of data communication, while the system design describes the communication between the Host and the Clients.

A. System Architecture

The goal of this research is that the devices in a smart home environment can be connected pervasively, so that there is no need for human to do a manual configuration. The device is NI MyRIOs, which are used as clients and Raspberry Pi 3 used as the host. The clients represent sensor nodes, wherein they sends the accelerometer data and LED status to the host. The LED on these clients are controlled by the host.The host also operates monitor and control features to be owned by the clients.

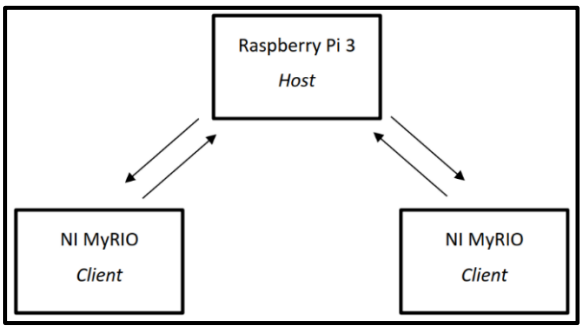

Figure 1: System architecture

There are three devices on Figure 1 that are proposed in the system architecture, the Raspberry Pi 3 device and two MyRIOs devices. The Raspberry Pi 3 device can be called the host and NI MyRIOs can be called the clients. Both of these devices (client and host) are connected via Wi-Fi with the pervasive method. The clients sent a broadcast message on connected network containing client's informations such as IP and its available sensors. After the broadcast, the messages are received and stored by the host. Subsequently, the host replied the messages to the client's IP that contains the host's information. This delivery process utilizes the UDP protocol provided by the LabVIEW.

B. System Design

Based on previous research [9][17], system design in Figure 2 is used on this research. Unlike the previous research that was conducted by testing the design suitability on PC, this research was conducted by implementing this system on embedded systems. In the first state, the host is in a condition of listening state, where the host listens to broadcast messages sent by the clients. When clients sends a broadcast message, the host checks whether the client information is a duplicated one. In cases where the client information is new, the host saves the information. The contents of the information are the IP, the client's name, and the provided features. After saving the information, the host sends a reply message to the client IP. The Send ACK state contains IP information of the host.

After the host and client know each other's information, the host can monitor and control the client's features. The host sends a request message to the clients, and the clients start sending sensor data and wait for the command sent by the host. The host receives the sensor data and controls the client's features, which is LED available on the client.

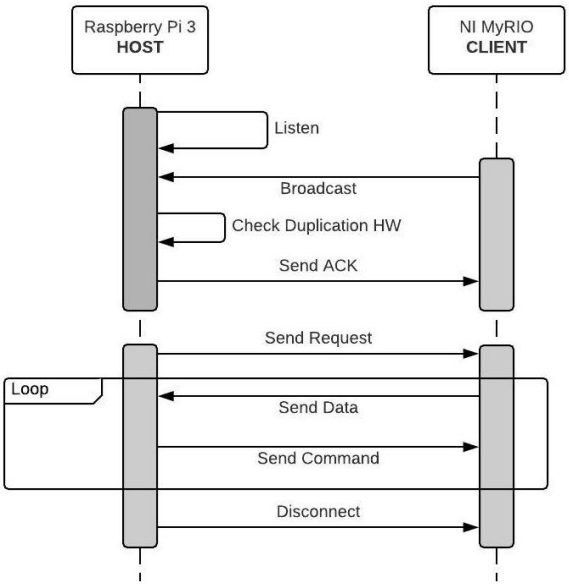

Figure 2: Proposed pervasive algorithm machine to machine area network

Table 1
Host Event from State Machine

Code	Host Event (e)
e0	Port initialization
e1	Receive broadcast message
e2	Receive client information
e3	Sent ACK message
e4	Hardware Push button
e5	Client control feature
e6	Finish hardware control from client

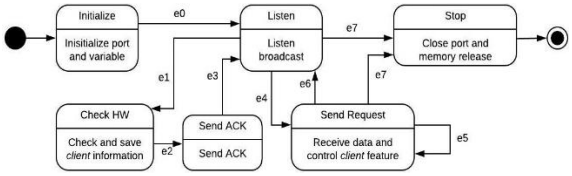

Figure 3: Host state machine diagram

Figure 3 shows the state machine used on the host and the related information, shown in Table 1. Firstly, the host enters the Initialize state where it initializes the required variables and opens the UDP protocol port. Afterward, "e0" event occurs when the port initialization and variable have been finished. It is then moved to the Listen state. This state is where the host's condition listena to the broadcast messages sent by the clients.

When the host receives a broadcast message from the clients, event "e1" occurs, and it then does a hardware duplication check in Check HW state. The host checks the received information with the stored information. If the client information is new, the information is stored by the host, whereas if the information has been previously saved, the previous information is deleted and replaced with a new one. After it is completed, it triggers the "e2" event and moves to Send ACK state. At this state, it sends a reply message to the client containing the host's information. Afterward, event "e3" occurs, by which it returns to the Listen condition state.

In Listen state, the host can control and monitor the sensor nodes owned by the clients whose information has been

previously saved. The host sends a request message to the client, where event "e4" occurs and waits for a response whether the messages have been received or otherwise. If a message is received, it goes to the Send Request state, otherwise it returns to the Listen state condition. In the Send Request status, the host receives accelerometer sensor data, push-button status data, and client's LED data. LEDs on the clients can be controlled directly by the host in this situation. In the process of sending and receiving data, event "e5" occurs, which is a time out process and it repeats the Send Request state .

When event "e5" finished, the Finish Hardware button in the host can be pressed, and subsequently it triggers the event "e6" and then it moves to the Listen state again. In the Listen and Send Status request state, the host can be stopped by pressing the Stop button. When the stop button is pressed, event "e7" occurs causing it to move to the Stop state. In this stop state, the host closes the port, while initializing and releasing the used memory.

Table 2
Client Event from State Machine

Code	Host Event (e)
e0	Port initialization
e1	Broadcast message was not received by host
e2	Broadcast message received by host
e3	No reply from host
e4	Received reply from host
e5	Wait reply from host
e6	Didn't receive ping message from host
e7	Receive request message
e8	Request done
e9	Send and receive data from host
e10	Stop

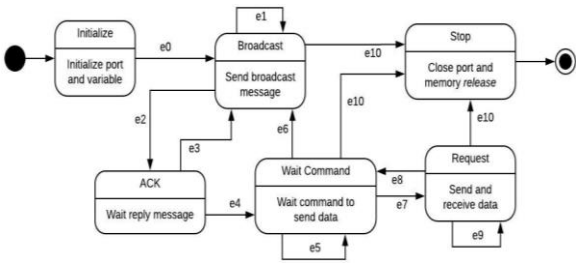

Figure 4: Client state machine diagram

Figure 4 is related to Table 2, which is the state machine used on the client. In the initial conditions, the client enters the Initialize state where client initializes the required variables and opens the UDP protocol port. Then event "e0" occurs where the port initialization and variables are completed. It then moves towards to Broadcast state. Broadcast State is a condition where the client sends broadcast messages to IP broadcasts. If the message has no reply, it triggers event "e1" which is a time out to repeat the Broadcast status again. If the message is received by the host, it triggers event "e2" and then it moves to the ACK state. In the ACK state, the client waits for a reply from the host regarding the information held by it. If the client does not receive a reply, it triggers event "e3" where it returns to broadcast status. If it receives a reply from the host, it triggers event "e4" and moves to the Wait Command state.

In the Wait Command state, the client waits for the request message sent by the host. If no message is sent within the specified time, it triggers event "e5" as a time out to repeat the Wait Command state. If the client does not receive the ping sent by the host at a specified time, it triggers event "e6", where the client returns to broadcast status again to find a new host because the previously host is assumed to be inactive.

When the client receives a request message, it triggers event "e7", where it moves to the Request state. In this Request state, the client sends the accelerometer sensor data and the information from the push button, and also the state of LED owned by the client. In this case, when some time out have been passed away, it triggers event "e9" that repeats the Request state so that it can always send information to the host. If the host has finished requesting data, it triggers event "e7", where the client moves to the Wait Command state again. In the Broadcast, Wait for Command and Request state, the client can be stopped by pressing the Stop button. When the stop button is pressed, event "e10" occurs, by which it then moves to the Stop status state. In this stop condition, the client closes the opened port while initializing and releasing the used memory.

III. Experiment Results

This section explains the experiment and its results analysis. The experiment is tested in two phases, the first phase is the discovery and the re-discovery experiment scenario and the second phase is the feature experiment scenario. Based on both scenario, it evaluates the fulfillment of the system requirements about whether the system is acceptable to be implemented in real-time or real environment condition.

A. Discovery Experiment Scenario

This scenario is carried out on three devices: one host and two clients. The host stores information from the client and it saves the transition time since receiving a broadcast message until replying the message. The client stores information from the host after receiving the ACK message and saves the transition time since sending broadcast messages until it waits for a request.

The experiment was done 60 times by which it was 30 attempts at each client. The results showed that the system works well. The average of discovery time on the host was 56.417 ms. This time is measured from the time when the host receives a broadcast message, stores information, and replies the message with the ACK message. While on the client, it has 257 ms for the transition time from broadcast transmission to Wait Command.

Figure 5 is a graph drawn from the discovery experiment result. At the early first ten discovery experiments, the result was fluctuating, but after that, until the end of experiments, the results was stable. This result is expected because of the hardware condition on the device, by which it is an old one with older firmware version compared to other devices. On the graph, it can be seen that there were different results on client1 with a value of 1036 ms, while the minimum value was 270 ms. The average results of discovery in client1 was 333.633 ms. The graph for discovery time in client2 was stable with a maximum value of 441 ms and a minimum value of 274 ms. The average value obtained on the client2 was 293.2 ms. Thus, it can be concluded that the entire time of the

discovery process since sending broadcast messages until the client is ready to receive a request was 313.417 ms.

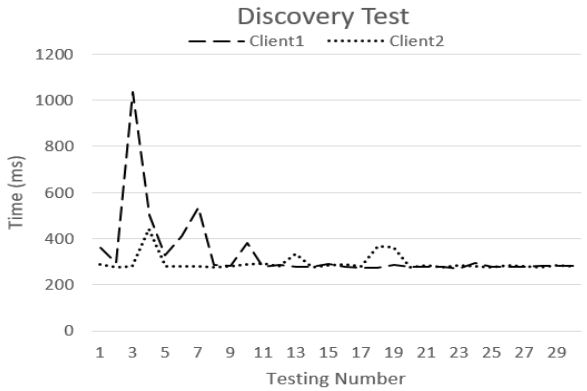

Figure 5: Discovery testing result

B. Re-Discovery Experiment Scenario

This experiment results can be seen on Figure 6. The host is created by forcing it to be failed or disconnected. It is done by stopping the program and re-deploying it. The client waits for the specified time until the client re-discovery the host and reconnected to it. The test was carried out 60 times, where each client made 30 attempts. Time is measured from when the client is disconnected from the host. The client has a timeout to return to the discovery condition which is 10 seconds. Client's return time to the discovery condition was 10056.88 ms. After returning to the discovery process, the client returns to the Wait Command state with an average of 291 ms. On the host, the discovery process is obtained at an average of 36.35ms.

Overall, the result of a re-discovery experiments were fluctuative. The data given by Figure 6 shows that each scenario was successfully tested without error, but the result was not stable both for client1 and client2. The obtained time were different in each experiment. In client1, the maximum value was 10568 ms and the minimum value was 10281 ms so that the average value of client1 was 10391.2 ms. Whereas in the client2, the maximum value was 10579 ms and the minimum value was 10568 ms so the average value obtained from the client2 was 10377.27 ms. The total time needed to do a reconnection has an average of 10.384.23 ms. It is expected that this fluctuation is caused by the difference in time needed from the host to shut down and restart its program.

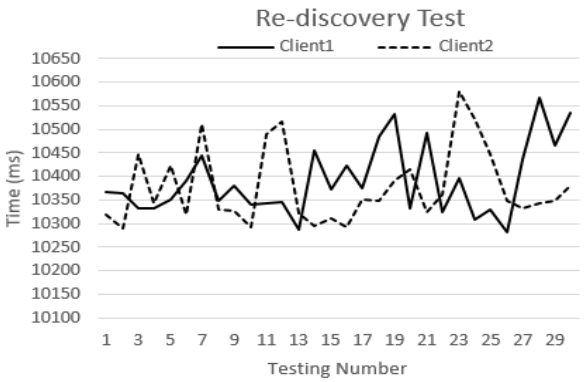

Figure 6: Re-discovery testing result

C. Feature Experiment Scenario

In this test, the host sends a request message to control and monitor features owned by the client. There are three used data: the accelerometer sensor data, LED control data, and the client's push button data. The experiment was carried out 30 times in each client. Testing run well without any error.

Table 3 shows the overall value at the client1 and client2. The data produced by each client was 86.3 bytes in client1 and 86.93 bytes in client2. Thus, the average data sent by the client is 86.62 bytes. The time measurement occurs at the beginning of sending data on both clients. The maximum time of sending sensor data to client1 was 508.63 ms, while the value on client2 was 691.809 ms. The minimum value of sending sensor data to client1 was 0.97 ms, while for client2 was 5.53 ms. So, the average value for sending sensor data from both clients was 58.26 ms. The next experiment is the sending time of the push button state to the host. Based on the overall results, the maximum value on client1 was 548.28 ms and for client2 was 246.18 ms. For the minimum value on client1, it was 44.62 ms while for client2 was 239.78 ms. Thus, the average time needed for sending the push button data in both client is 255.7 ms.

Table 3
Overall Feature Testing Result

Testing	Min	Max	Average
Client1Sensor (ms)	0.97	508.63	74.25
Client1 Push button (ms)	44.62	548.28	267.86
Client1 Led (ms)	3809.27	15346.77	9054.65
Client1 Data Sizes (bytes)	76.00	90.00	86.30
Client2 Sensor (ms)	5.53	691.81	36.57
Client2 Push button (ms)	239.78	246.18	243.51
Client2 Led (ms)	480.63	3003.92	2115.63
Client2 Data Sizes (bytes)	84.00	89.00	86.93

The maximum delivery time for LED data from client1 was 15346.77 ms, while for client2 was 3003.918 ms. The minimum delivery time from client1 was 3809.27 ms, while for client2 was 480.63 ms. These results show that the required delivery time was longer for each subsequent time. It happens because there is a difference in speed between sending and receiving caused by the process in buffer on the receiver. For both clients, there was a very large time difference, because client1 is no longer connected to the host before doing the experiment, causing the buffering of the accumulated data, compared to the client2 that is not connected in longer time. The required time for data delivering has an average value of 5350.93 ms for both clients.

All of those time measurement results have a lower value compared to other researches which have a value in the range of 200 s or about 3 mins [25].

IV. Conclusion and Future Works

It can be concluded that the device succedded in recognizing the surrounding devices without requiring complex configuration by human, particularly using the pervasive computing method. In this method, humans do not have to bother with configuration process for each devices. The devices are programmed to disseminate information automatically, and they can store information from other devices when receiving a reply. In this research, the used protocol is UDP protocol, which contains data about 86.62

bytes in average. Thus, the message is light and does not overload the device's memory. The system in this study was implemented on Raspberry Pi 3 devices as the host and two MyRIO NI as the clients. These devices are programmed by LabVIEW. The discovery and re-discovery experiments were used to measure the quality of the system. The result of discovery and re-discovery experiments showed that it is implemented without error, but it had unstable time for both scenarios.

The program is made by implementing a state machine method. Each device can make discoveries so that they know each other, including its information without manual complex configuration. The system is also successfully implemented and tested without any error in the feature owned by MyRIO such as a sensor, push button, and LED. Each of those services has an average time in sending data for both clients, which is 58.26 ms, 255.7 ms, and 5350.93 ms respectively. Based on this research, the system is acceptable for being implemented on smart home environment. MyRIO can be implemented as a server or slave or Host Node and Raspberry PI can be implemented as an end node, slave or Client Node. More client node can be attached on the implementation. The number on clients is not tested on this research, and it can be implemented and analyzed for further research. The measurement of the availability, suitability, portability, etc will be done at that time after some several requirements have been added. Other research can be done by adding more sensors to fulfill the user requirements.

ACKNOWLEDGMENT

The team thanks to the Laboratory of Computer System and Robotics, Computer Engineering Department, Faculty of Computer Science. This paper is an extension of work originally published and reported in "IAES 2016: International Conference on Electrical Engineering, Computer Science and Informatics (EECSI 2016)" at Semarang, East Java, Indonesia 22-25 November 2016, with the title "Lightweight UDP Pervasive Protocol in Smart Home Environment based on LabView". The other works that related are "UDP Pervasive Protocol Implementation for Smart Home Environment on MyRIO using LabVIEW", Vol 8, No 1 February 2018 and "UDP Pervasive Protocol Integration with IoT for Smart Home Environment using LabVIEW", Vol 8, No 6, December 2018 (Part II).

REFERENCES

[1] S. D. T. Kelly, N. K. Suryadevara, and S. C. Mukhopadhyay, "Towards the Implementation of IoT for Environmental Condition Monitoring in Home," IEEE Sensors Journal, vol. 13, no. 10, pp. 3846-3853, Oct. 2013.

[2] Arda Editya; Surya Sumpeno; Istas Pratomo, "Performance IEEE 802.14.5 and zigbee protocol on realtime monitoring augmented reality based wireless sensor network system," International Journal of Advances in Intelligent Informatics, vol. 3, no. 2, pp. 90-97, 2017.

[3] L. Atzori, A. Iera, and G. Morabito, "Making things socialize in the Internet — Does it help our lives?," in Proceedings of ITU Kaleidoscope 2011: The Fully Networked Human? - Innovations for Future Networks and Services (K-2011), Cape Town, 2011.

[4] B. Zhou et al., "Smart home energy management systems: Concept, configurations, and scheduling strategies," Renewable and Sustainable Energy Reviews, vol. 61, pp. 30-40, 2016.

[5] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," Business Horizons, vol. 58, no. 4, pp. 431-440, 2015.

[6] S. Dey, A. Roy, and S. Das, "Home automation using Internet of Thing," in 2016 IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, 2016, pp. 1-6.

[7] A.Jacobsson, M.Boldt, and B.Carlsson, "A risk analysis of a smart home automation system," Future Generation Computer Systems, vol. 56, pp. 719-733, 2016.

[8] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context Aware Computing for The Internet of Things: A Survey," IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 414-454, 2014.

[9] Kurniawan, W., Ichsan, M.H.H., Akbar, S.R., and Arwani, I., "Lightweight UDP Pervasive Protocol in Smart Home Environment Based on Labview," in IAES International Conference on Electrical Engineering, Computer Science and Informatics (EECSI 2016), Semarang, Indonesia, 2016.

[10] M. Duke, R. Braden, W. Eddy, E. Blanton, and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents," , February 2015.

[11] M. Iglesias-Urkia et al., "Towards a lightweight protocol for Industry 4.0: An implementation based benchmark," in IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), Donostia-San Sebastian, 2017.

[12] R. Morabito, R. Petrolo, V. Loscrí, and N. Mitton, "Enabling a lightweight Edge Gateway-as-a-Service for the Internet of Things," in 7th International Conference on the Network of the Future (NOF), Buzios, 2016.

[13] Z. Sheng, D. Tian, and V. C. M. Leung, "Toward an Energy and Resource Efficient Internet of Things: A Design Principle Combining Computation, Communications, and Protocols," IEEE Communications Magazine, vol. 56, no. 7, pp. 89-95, 2018.

[14] Pallavi Sethi and Smruti R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," Journal of Electrical and Computer Engineering, vol. 2017, p. 25, 2017.

[15] P. Dalal, M. Sarkar, N. Kothari, and K. Dasgupta, "efining TCP's RTT dependent mechanism by utilizing link retransmission delay measurement in Wireless LAN," International Journal of Communication Systems, vol. 30, no. 5, 2015.

[16] W. Kurniawan, M. H. H. Ichsan, and S. R. Akbar, "UDP Pervasive Protocol Implementation for Smart Home Environment on MyRIO using LabVIEW," International Journal of Electrical and Computer Engineering (IJECE), vol. 8, no. 1, pp. 113-123, February 2018.

[17] M.H.H. Ichsan, W. Kurniawan, and S.R. Akbar, "UDP Pervasive Protocol Integration with IoT for Smart Home Environment using LabVIEW," International Journal of Electrical and Computer Engineering (IJECE), vol. 8, no. 6 Part II, December 2018.

[18] P. P. Merino, E. S. Ruiz, G. C. Fernandez, and M. C. Gil, "A Wireless robotic educational platform approach," in 13th International Conference on Remote Engineering and Virtual Instrumentation (REV), Madrid, 2016.

[19] L. Zimmermann, R. Weigel, and G. Fischer, "Fusion of Nonintrusive Environmental Sensors for Occupancy Detection in Smart Homes," IEEE Internet of Things Journal, vol. 5, no. 4, pp. 2343-2352, 2018.

[20] R. Raj, R. K. Sahu, B. Chaudhary, B. R. Prasad, and S. Agarwal, "Real time complex event processing and analytics for smart building," in Conference on Information and Communication Technology (CICT), Gwalior, 2017, pp. 1-6.

[21] V. Bhanumathi and K. Kalaivanan, The Role of Geospatial Technology with IoT for Precision Agriculture.: Springer, Cham, 2018, vol. 49.

[22] V. Vujović and M. Maksimović, "Raspberry Pi as a Sensor Web node for home automation," Computers & Electrical Engineering, vol. 44, pp. 153-171, 2015.

[23] Ioannis Giachos, Evangelos Papakitsos, and Georgios Chorozoglou, "Exploring natural language understanding in robotic interfaces," International Journal of Advances in Intelligent Informatics, vol. 3, no. 1, pp. 10-19, 2017.

[24] M. H. H. Ichsan, W. Kurniawan, and M. Huda, "Water Quality Monitoring with Fuzzy Logic Control Based on Graphical Programming," TELKOMNIKA, vol. 14, no. 4, pp. 1446-1453, 2016.

[25] O. A. Mohamad, R. T. Hameed, N. Tapus, "Smart Home System Based on Comparative Analysis Among AODV and DSDV Protocols in MANET," in 19th International Conference on System Theory, Control, and Computing (ICSTCC), Cheile Gradistei, Romania, 2015.