**074-x-22**

**Sensor-based Intelligence for Smart Environments using Internet of Things (IoT): A Case Study**

Sangho Park

Central Connecticut State University

[spark@ccsu.edu](mailto:spark@ccsu.edu)

**Abstract**

The development of educational hardware/software systems has been considered increasingly important in experiential learning for technology education and in hands-on experiments for engineering research. The Internet of Things (IoT) implemented from the embedded systems approaches is a useful domain for the purpose. This paper presents a case study that shows the design, implementation, and testing of an IoT system using low-cost off-the-shelf components. The system is composed of multiple sensor nodes dispatchable to remote locations and a server computer that a user locally controls. The sensor nodes and the server computer are connected via wireless internet for the transmission of sensory data. The developed IoT system can serve as a distributed sensor network with heterogeneous sensors that provide environmental monitoring capabilities for temperature, humidity, barometric pressure, geo-localization and video streaming from dispatched sites. The experimental evaluation shows the system’s reliable performance and accuracy. The system can be used as an example for capstone projects in educational setting as well as for actual deployment in smart environments with the addition of intelligent analytics applications on the server.

**Introduction**

The development of educational hardware/software systems has been considered increasingly important in experiential learning for technology education and in hands-on experiments for engineering research. The Internet of Things (IoT) implemented from the embedded systems approaches is a useful domain for the purpose.

The Internet of things describes physical objects (or groups of such objects) with sensors, processing ability, software, and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks (Wikipedia, 2022.) The IoT systems can vary in scale from personal health-monitoring systems to city-scale smart transportation systems. The complexity of involved technologies is also various. Health monitoring systems, for example, can use near-field communication (NFC) or Bluetooth (IEEE 802.15) communication between sensors and monitoring devices. The sensors in such systems typically consume small energy supplied by battery or energy harvesting technologies. In contrast, city-scale smart transportation systems involve wide-area networks (WAN or cellular) with heterogeneous communication protocols and plethora of sensors. Sensors in such systems can use high-power electricity for wide sensing coverage and reliability. The survey of Ang et al. ((Ang et al., 2022)) categorizes the research and development domains of IoT into two phases. The first phase focuses on developing the building blocks and enabling technologies, and the second phase focuses on the addition of values to the application domain such as smart environments and intelligent analytics. Oftentimes, the two phases are intertwined as new technologies emerge. For a specific research group, both the two phases are necessary to go through.

Even if the general trend of IoT system developments is to combine more and more technologies, it is also true that specific application domains need a specific set of technologies optimized to the application use cases. Liu (2016), for example, reports a case study of designing and implementing an intelligent environmental-control system specified for the maintenance of a large scale (8 meter x 50 meter footprint) greenhouse. In this study multiple heterogeneous sensors are hard-wired and installed across the greenhouse and the sensor readings are wirelessly transmitted by a data collection node to a remote web server for user access. The sensors and the data collection node are specially packaged for waterproofing.

It is important to consider the best choices among available technologies and options in the specific application use cases. From the embedded systems perspective, it is imperative to balance between the system performance and the cost of overall system operations. The study of the current paper presents an embedded-systems approach to Internet of Things, and shows a case study of designing, implementing, and evaluating a portable low-cost intelligent system for smart environment.

Majority of the works in IoT reviewed in the survey papers (Ang et al. 2022; Nauman et al. 2020) are large-scale research and encompass various application domains. Many new approaches and methods have been proposed and evaluated. Some of them became standard technologies and others are in experimental phases.

The case study presented in this paper mainly adopts standard technologies available to open public and uses off-the-shelf devices for implementation of the proposed system.

**System Structure Design**

Figure 1 shows the use case diagram of the proposed systems for smart environment represented as a UML (universal markup language) Use Case diagram. Smart environment (Cook & Das, 2004) has been envisioned as "a small world where different kinds of smart device are continuously working to make inhabitants' lives more comfortable." Smart environment is a physical world that is richly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the environment and connected through a continuous network.

The overall system structure is a server-client system with wireless network connection. The ‘Analyzer’ in Figure 1 is the human user of the smart environment who accesses the server computer to get awareness of the environmental status. The Analyzer wants to get aware of the situation, which involves interpreting, visualizing, analyzing, and accumulating sensor data transmitted from remote sensor nodes. The server computer local to the user is depicted by the dotted round rectangle on top, and the group of remote sensor nodes are depicted by the dotted round rectangle at bottom. Inside the group of the sensor nodes is a single embedded system enclosed by the dotted small rectangle, which corresponds to a sensor node. The sensor node runs various sensors, packages sensor data, back up the data, and send it to the remote server via wireless communication. The server computer communicates with multiple sensor nodes simultaneously.

Sensor nodes (‘Node\_1’, ‘Node\_2’, etc. in Figure 1) are independent embedded systems with their own OS, and work as clients in the network. The communication among the server and the clients in the current study uses WiFi (IEEE 802.11) wireless network with a router. The router provides a private network for the communication. It is possible to configure the router to support a public network, but it would need consideration of network security issues. Given that the sensor node clients are small-scale embedded computers, it is safer to configure the router for private network.

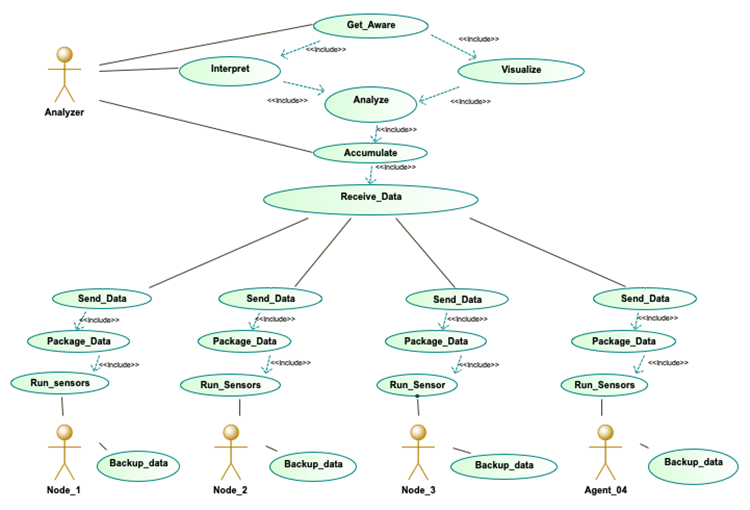


Figure 1: Use cases of the system

**Hardware Design**

The implementation of the current study uses four Raspberry Pi embedded computers as the sensor node clients and an Apple MacBook Pro computer as the server. Any general-purpose computer (e.g., Windows PC, Apple Mac computer, Linux computer, etc.) can work as the server in the system.

The sensor node uses ARM Quad core Cortex-A72 chip as its CPU with 1.5 GHz clock speed. It supports hardware interfaces via its general-purpose input and output (GPIO) pins, which can be connected to various sensor devices. Each sensor device has its own communication interface such as inter-integrated circuit (I2C) interface bus, serial peripheral interface (SPI) bus, or universal asynchronous receiver/transmitter (UART) bus, etc.

Figure 2 shows the overall system hardware diagram of the sensor node.

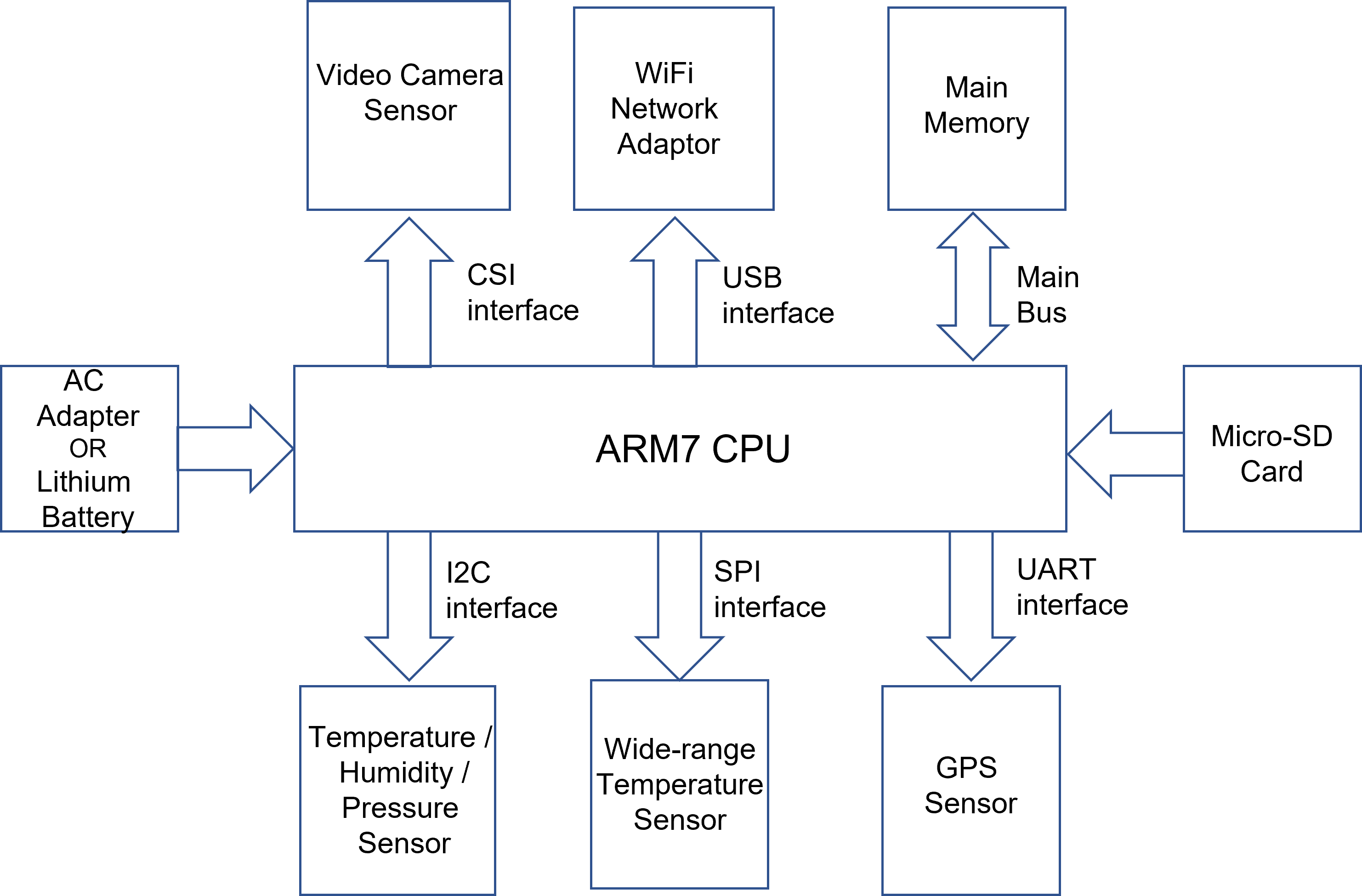


Figure 2: System Diagram

The sensor node includes BME280 combo-sensors for temperature, humidity, and barometric pressure, MAX31855 thermo-coupled temperature sensor, MT3330 GPS sensor, and a camera with Camera Serial Interface (CSI).

Figure 3 shows the prototype implementation of the sensor node.

A picture containing text

Description automatically generated

Figure 3: Prototype implementation of the sensor node

BME280 chip is manufactured by Bosch Inc., and MAX31855 chip is manufactured by Analog Devices Inc. MT3330 chip is manufactured by Mediatek Inc., and the camera is equipped with Sony IMX477R CCD sensor.

BME280 supports I2C bus, and MAX31855 supports SPI bus. MT3330 supports UART bus, and the camera is connected to the ARM7 CPU via CSI interface bus.

These heterogeneous sensors and their diverse interfaces are supported by Raspberry Pi GPIO interface driven by either 5V or 3.3V voltage. More sensors can be integrated if the communication bus interface is supported by the sensor node.

The hardware specifications of the sensors are as follows.

BME280 has accuracy of ±3% for humidity, ±1°C for temperature, and ±1 hPa for barometric pressure.

MAX31855 can measure temperature range between -200C and +1350C with proper K type thermocouple. The accuracy is about ±2 °C to ± 6 °C depending on the thermocouple. Therefore, MAX31855 has different use cases in temperature sensing compared to BME280 of which temperature sensing is mainly for atmospheric ambient temperature.

MT3330 can search 66 satellites and track 22 of them. Its GPS position accuracy is within 3 meters, and the update rate is 1 to 10 Hz.

Sony IMX477R CCD (charge-coupled device) sensor supports 12.3 Mega pixels of image resolution.

The camera sensor provides video data as a series of image frames, whereas all the other sensors provide scalar data (i.e., values.)

The sampling rate of the scalar-data capture in this study is 1 second for all the sensors (except camera) for evaluation purpose. In usual settings, it would not be necessary to sample temperature and humidity every second.

The camera sensor supports video streaming as well as snapshot image capture. Image and video processing is highly CPU-intensive and causes CPU temperature to rise. The current implementation uses a passive heat dissipation method (i.e., aluminum diecasting case plus heat sink) for steady video streaming, but active cooling systems (e.g., cooling fan) may be needed for more CPU-intensive jobs such as image processing.

**Software Design**

Figure 4 shows the overall structure of software design for data processing and communication between a single sensor node and the server computer. Note that the server computer communicates with four sensor nodes in the same fashion simultaneously. The dotted rectangle on the left represents the sensor-node processes, and the dotted round rectangle on the right represents the server processes.

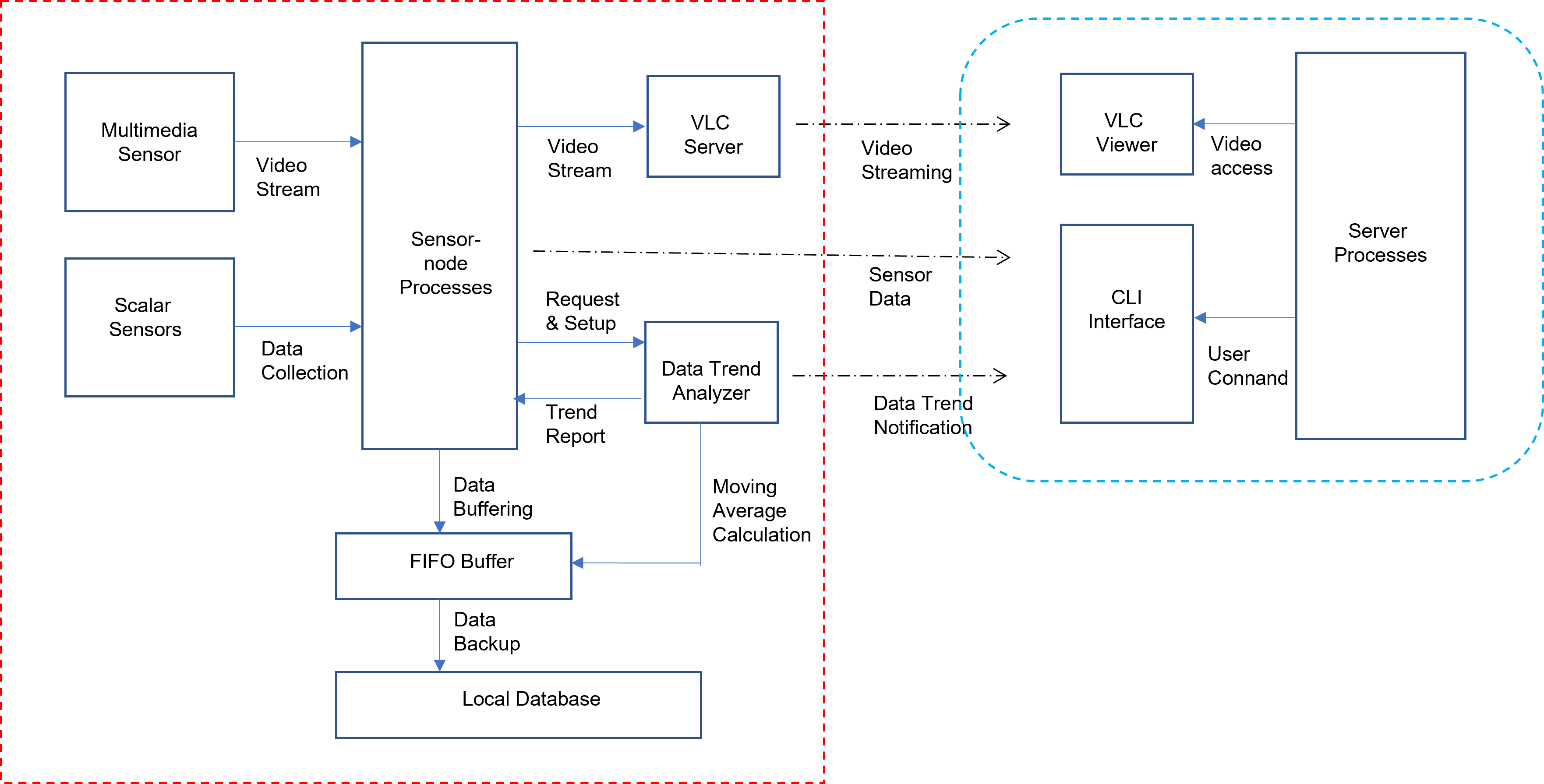


Figure 4: Software structure for data processing and communication

The CCD sensor in the camera generates a live-video stream which needs fast refresh rate (i.e., typically 30 frames per second) and each frame occupies a huge amount of data compared to scalar sensor data. Therefore, the video and scalar data are processed differently in the sensor node.

Video data provides the user with rich information about the monitored scene for context awareness in smart environments (Park & Trivedi, 2008). Context awareness is one of the most salient features in intelligent environment (Roy et al., 2006) and the realization of automatic context awareness requires advanced image processing algorithms, which are typically CPU-intensive. In the current study, the video data is mainly processed for the purpose of video streaming from the sensor node to the remote server computer via a wireless network for the user’s context awareness in manual manner.

The scalar data from the scalar sensors are time-series data, and they are processed mainly for data logging and data trend analysis. The sensor-node processes keep sending the scalar data to the remote server via the wireless communication while backing up the data locally. The scalar data is buffered into a first-in first-out (FIFO) queue implemented on the main memory and backed up to flash SD card sporadically. The FIFO queue also serves as a data buffer for transmission.

The sensor-node processes also include the data trend analysis to detect a rising or falling pattern of environmental variables (e.g., rising and falling of ambient temperature, humidity, etc.) Sensory data in general is noisy due to hardware characteristics and electromagnetic influences, and it is not practical to compare two adjacent values to tell if the sensory value is rising or fall due to the noise. It is necessary to filter out the noise from the signal. Moving average plot has been used to filter the spiky noises from data and to obtain a reliable smooth signal patterns as shown in (Fang et al. 2014.)

The moving average value ***mt*** at the *t-th* timestamp is defined by summing the raw sample value ***rt*** and N values around the raw value along the time as follows.

The even number *N* determines how many nearby values are averaged. For example, if *N*=4, total five sample points including *rt* are averaged. The value *N* needs to be configured depending on the characteristics of the sampled data pattern.

The study in the current paper extends the moving average method to develop the Data Trend Analyzer in Figure 4. The Data Trend Analyzer accesses the FIFO buffer on the main memory to calculate the moving average values at every sampling moment and evaluates if the data stream is in rising or falling trend. The basic idea is similar with the time-series analysis in statistics (Hansun, 2013) or the technical analysis in stock market trading (Raudys et al., 2013.) It uses two moving averages of different scale as follows. A long-term moving average with a big value of N is compared to a short-term moving average with a small value of N over time. The short-term moving average is more sensitive to the values adjacent to the current value, while the long-term moving average indicates overall value in longer time duration. The data trend analyzer detects the moment of crossing between the two moving average plots and reports if it is a rising trend or a falling trend.

Each sensor node runs its own software for sensor reading and data backing up, while all sensor nodes should be communicating with the server in a corporative manner. In the current system, the server-client communication is handled by multi-socket connections approach, where the server opens / closes multiple sockets on demand from the clients and processes the communications via event driven interrupts.

Each sensor node has the following major jobs. It reads data from hard-wired multiple sensors, converts the data format to byte streams, stores the data temporarily on local flash SD card, and send the data to server wirelessly. The data communication is guaranteed by TCP/IP protocol which exchanges acknowledgement packets at every transaction.

The execution of programs on the sensor node can be autonomously run according to a planned schedule via crontab utility on Linux OS on the client. It also can be initiated and controlled by user on the server side. Secure connection between the client and the server is maintained through the WiFi network via SSH or SFTP protocols.

The software paradigm is well-suited with the system design in Figure 1. The current implementation of the software for the system is based on object-oriented design and uses Python and C++ programming languages. For fast processing, C/C++ library modules are called from Python using C/Python binding mechanism and running behind the Python code.

**Experiment and Data Analysis**

One of the motivations in the current study is to prototype a working system for distributed mobile sensor network by using off-the-shelf components and to evaluate the reliability of such systems. An experiment was conducted for basic data analysis, which includes reliability test in data collection process, accuracy estimation of sensory data, and exploration of data profiling for descriptive statistics (i.e., mean, standard deviation, and curve fitting.)

The experiment was conducted over two consecutive days using four sensor nodes and one server computer. The sensor nodes were placed strategically on different sites within a two-story building. Node\_1 was placed inside the basement near to its walkout entry. Node\_2 was placed inside a room on the ground floor near window. Node\_3 was placed inside a second-floor room with lots of windows. Node\_4 was placed near to attic area. All windows and doors were closed during the experiment, and no HVAC systems were on. Each node has collected sensory data every second for two days, producing about a 10 Mbytes of scalar data per node.

Figure 5 shows the temperature graphs (y-axes are common) in Celsius scale from the four sensor nodes. The x-axis is sampling index in time. The temperature values of [average ± standard deviation] of the four sensors are [23.2 ± 0.15] °C, [24.29 ± 0.5] °C, [28.27 ± 0.56] °C, and [28.56 ± 1.56] °C for sensors 1 through 4, respectively. As expected, temperature variation becomes bigger from the lower to higher levels in the building. The temperature rises from the lowest at basement to the highest at near the attic area. The two dips in the temperature curves correspond to nighttime, while the peaks correspond to daytime. The temperature variations between day and night become wider from the basement near to the attic area. This implies that the upper-level temperature near to the attic is more affected by the day/night ambient temperature influence from outside.

Graphical user interface, chart, histogram

Description automatically generated

Figure 5: Temperature graphs in Celsius scale

Figure 6 shows the relative humidity graphs (y-axes are common) in percentage scale. The relative humidity values of [average ± standard deviation] of the four sensors are [51.50 ± 2.19] %, [56.39 ± 1.57] %, [45.39 ± 1.25] %, and [44.39 ± 2.83] % for sensors 1 through 4, respectively. The relative humidity is inversely correlated with ambient temperature. The ambient temperature profiles and the relative humidity profiles also show distinct patterns depending on the location of the sensor node. The humidity is highest on the ground-floor data and lowest on the second floor near the attic.

Graphical user interface

Description automatically generated

Figure 6: Relative humidity graphs in percentage scale

Figure 7 shows the barometric pressure graphs in hPa scale. The barometric pressure values of [average ± standard deviation] of the four sensors are [1003.45 ± 1.63] hPa, [1002.84 ± 1.59] hPa, [1003.30 ± 1.61] hPa, and [1002.87 ± 1.62] hPa for sensors 1 through 4, respectively. The four graphs show very similar patterns of variation. The barometric pressure patterns of the four sensor nodes closely resemble one another, which implies the sensor reading is reliable.

Chart

Description automatically generated

Figure 7: The barometric pressure graphs in hPa scale

Figure 8 shows the GPS latitude coordinate values in degrees. The values of [average ± standard deviation] of the four sensors are [41.7939 ± 0.00023] °, [41.7942 ± 0.00014] °, [41.7939 ± 0.00007] °, and [41.7940 ± 0.00006]° for sensors 1 through 4, respectively. Note that the actual GPS coordinate values of the latitude and the longitude were obfuscated by adding a random constant integer for privacy reasons. However, adding the constant number does not affect the standard deviation or the temporal patterns. The high spikes in the graph of Node\_1 shows signal fluctuation due to the insufficient number of tracked satellites.

Graphical user interface, chart

Description automatically generated with medium confidence

Figure 8: GPS latitude coordinate values in degrees

Figure 9 shows the longitude coordinate graphs in degrees. The values of [average ± standard deviation] of the four sensors are [-72.8859 ± 0.00024] °, [-72.8858 ± 0.00012] °, [-72.8859 ± 0.00009] °, and [-72.8857 ± 0.00005] ° for sensors 1 through 4, respectively. The high spikes in the graph of Node\_1 shows signal fluctuation due to the insufficient number of tracked satellites.

Graphical user interface, application

Description automatically generated

Figure 9: GPS longitude coordinate values in degrees

Figure 10 shows the temporal graphs of the number of tracked satellites in integer count. The number of tracked satellites are integer numbers. For validation purpose, the values of [average ± standard deviation] of the four sensors are [5.5 ± 1.43], [7.8 ± 1.64], [9.1 ± 1.65], and [10.4 ± 1.54] satellites for sensors 1 through 4, respectively. The number of tracked satellites directly influence on the reliability of latitude/longitude calculations, as shown in Figures 8 and 9. It is also noticed that indoor environment can receive satellite signals, although it depends on the material and structure of the building. The GPS coordinates of latitude and longitude are similar among the four sensor nodes. The random peak noises apparent on the Node-1 at the basement implies certain unreliability of the GPS readings. This is due to the location of the MT3330 GPS sensor at the basement. Even if it is at the basement, the sensor receives GPS signals from 2 to 10 tracked satellites. This is in contrast to the more reliable GPS readings from the other sensor nodes, which are located on over the ground. As the locations of the sensor nodes go from the basement, to the ground level, to the second floor, and to near the attach area, the number of tracked satellites increases from 5, 7, 9, to 10 (See Figure 10). The fluctuation pattern in the number of tracked satellites is normal for GPS receivers in general situations. The MT3330 GPS sensor supports simultaneous searching and tracking of numerous satellites.

Chart, bar chart

Description automatically generated

Figure 10: The number of tracked satellites in integer count

Figures 11 and 12 show the moving average plots overlaid on the raw data plot. ‘Raw’ in the legend means the raw data plot, ‘MovAvg1’ means short-duration moving average plot, and ‘MovAvg2’ means long-duration moving average plot. Figure 11 shows that long-duration moving average (MovAvg2) provides better profile of the data trend than short-duration moving average (MovAvg1), while Figure 12 shows that short-duration moving average performs better.

*~~Chart, line chart

Description automatically generated~~*

Figure 11: Temperature curves in raw vs. moving averages.

*~~Chart, histogram

Description automatically generated~~*

Figure 12: Humidity curves in raw vs. moving averages.

Figures 11 and 12 also show the comparison of the short-term and long-term moving average plots. The short-term and the long-term average patterns represent the trend of temperature change in the short term and the long term, respectively. It is likely that the next temperature in near future will rise when the short-term moving average goes above the long-term moving average, and vise-versa. The curve-crossing moment is detected by the data trend analyzer on the sensor node and notified to the user on the server computer as trend report.

This trend report provides useful information to the user for decision making regarding the environment.

Figure 13 shows the snapshots of streamed videos from the cameras on two sensor nodes (node-2 and -3), respectively. The other sensor nodes (node-1 and -4) have blocked camera views due to the installed locations in the experiment (i.e., basement and attic, respectively.) Figure 13 shows the view through the window glass and the entrance viewed from inside, respectively.



Figure 13: Snapshots of streamed videos from the sensor-node cameras

The video streaming by the sensor nodes can be accessed from the remote server using the Virtual Network Computing (VNC) application. VNC supports multiple simultaneous access to the sensor nodes.

Table 1 shows the CPU die temperature changes over time with specific processes. The temperature while idling is the baseline. The CPU temperature stays at the baseline level while processing scalar sensors. The CPU temperature rises significantly while streaming a live video, but it does not go over 70 °C over time. The heat dissipation in the current implementation uses passive method, i.e., aluminum diecast case with heat sink. The case internally contacts the CPU and the memory chip via a heat-transferring tape. This passive heat dissipation method is enough to maintain the CPU temperature within a reasonable bound. For reference, the CPU temperature of the server computer (Apple MacBook Pro) was 64 °C during the experiment.

Table 1: CPU die temperatures in Celsius during various tasks



Active cooling system (e.g., cooling fan) would significantly reduce the CPU temperature of the sensor, but it would consume significant amount of electric power of the sensor node. Given the limited hardware resources of the embedded systems in general, it is not desirable to use active cooling methods.

**Conclusions**

The development of hardware/software systems is important in experiential learning and hands-on experiments in engineering education. This paper presented a case study of the design, implementation, and testing of an IoT system for environmental monitoring from the embedded systems approach.

The results from the study show that development of a distribute sensor network is achievable using off-the-shelf components and low-cost generic embedded system computers. Multiple sensor nodes, each of which is an independent embedded system with heterogeneous sensors, are simultaneously connected to a general-purpose notebook computer to provide environmental monitoring capabilities including temperature, humidity, barometric pressure, geo-localization and video streaming. Experimental evaluation shows that the developed IoT system is reliable to use in extended time and accurate in data sampling and transmission. The sensor nodes also realize the locally embedded intelligence (e.g., data trend analysis) and autonomously trigger notifications to the server computer. The system can be used as an exemplar capstone project as a whole or as term projects in pieces in educational setting. The developed system can also be used for actual deployment in smart environments with extra intelligent analytics applications on the server.

Future study will include weather-proof housing of the sensor node for outdoor deployment.

It is also part of the future study to investigate light-weight image processing on board on the sensor node side for locally embedded intelligence.

**Acknowledgments**

This research was supported and funded by AAUP-CSCU University Research Grant from Central Connecticut State University.

**References**

Ang, L, Seng, K, & Wachowicz, M. (2022). Embedded intelligence and the data-driven future of application-specific Internet of Things for smart environments, *International Journal of Distributed Sensor Networks*, 2022, Vol. 18(6)

Cook, D. & Das, S. (Ed.) (2004). Smart Environments: Technology, Protocols and Applications. Hoboken, NJ: J. Wiley & Sons.

Fang, S.F., Xu, L.D., Zhu, Y.Q., (2014). An integrated system for regional environmental monitoring and management based on internet of things, *IEEE Transactions on Industrial Informatics*, Vol. 10(2).

Hansun, S. (2013). A new approach of moving average method in time series analysis, *Conference on New Media Studies* (CoNMedia), pp. 1-4, doi: 10.1109/CoNMedia.2013.6708545.

Liu, J. (2016). Design and implementation of an intelligent environmental-control system: perception, network, and application with fused data collected from multiple sensors in a greenhouse at Jiangsu, China, *International Journal of Distributed Sensor Networks*.

Nauman, A., Qadri, Y.A., Amjad, M., Zikria, Y. B., Afzal, M. K., & Kim, S. W. (2020). Multimedia Internet of Things: a comprehensive survey. *IEEE Access*, Vol.8.

Park, S. and Trivedi, M. (2008). Understanding Human Interactions with Track and Body Synergies (TBS) Captured from Multiple Views, Computer Vision and Image Understanding: Special Issue on Intelligent Visual Surveillance, 111(1), pp. 2-20, Elsevier Inc.

Raudys, A., Lenčiauskas, V., Malčius, E. (2013). Moving Averages for Financial Data Smoothing. In: Skersys, T., Butleris, R., Butkiene, R. (eds) *Information and Software Technologies. ICIST 2013. Communications in Computer and Information Science*, vol 403. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-642-41947-8_4>

Roy, N., Roy, A., & Das, S. (2006). Context-Aware Resource Management in Multi-Inhabitant Smart Homes: A Nash H-Learning based Approach. In Proc. of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications.

Wikipedia. (2022). Internet of Thing, https://en.wikipedia.org/wiki/Internet\_of\_things , retrieved on July, 28th, 2022.

**Biographies**

SANGHO PARK is currently an Associate Professor of Computer Electronics and Graphics Technology at Central Connecticut State University, where he has been teaching courses including Circuit analysis, Analog electronics, Embedded systems, and Capstone projects. His research interest includes Sensor-based intelligent systems, Computer vision, Image processing, and Machine learning. Dr. Park may be reached at [spark@ccsu.edu](mailto:spark@ccsu.edu).