IOT FRAMEWORK FOR MANAGEMENT OF ROS BASED AUTONOMOUS MOBILE ROBOT

Hasna Parveen K Z[1], Tasneem Fathima K Z[2], Anjana Anil[3], Shilpa John[4], Jayadas C K[5], Hijas E W[6], Anurag R[7], Saju Nampoothiri[8]

[1,2,3,4,5]*Department of Electronics Engineering, Govt. Model Engineering College, APJ Abdul Kalam Technological University, India*
Email: Hasnaparveen1999@gmail.com, Tasneemfathima1999@gmail.com, anjanaanil.mec@gmail.com, shilpajohn.mec@gmail.com, jck@mec.ac.in
[6]*TATA ELXSI, Neyyar Building, Technopark, Trivandrum*
Email: hijasew@gmail.com
[7,8]*Trizlabz, AITRON Labs Pvt. Ltd., Kochi, India*
Email: ar@trizlabz.com, sn@trizlabz.com

Abstract:
This paper presents the design of an Autonomous Mobile Robot (AMR) developed on Robot Operating System (ROS) taking care of the architecture and communication protocols for remote management. The objective is to communicate with the robot using the IoT platform and to have a detailed analysis of the communication architecture between the IoT platform and the ROS framework in the robot. The system uses a Kaa IoT platform based central server. Remote management of AMR is achieved through integration of frameworks such as ROS, MQTT and Cloud Platform. The performance of the proposed system is assessed through simulation and the requisite tests are then run on the platform. Satisfactory results have been attained and the insights gained in the platform design will pave the way for future works. The end-to-end communication between a user and an AMR can be extended to fleet management of AMRs by making use of the scalability of IoT platform.

Keywords:
Autonomous Mobile Robot (AMR), Robot Operating System (ROS), Message Queuing Telemetry Transport (MQTT), Internet of Things (IoT), Cloud Platform.

1. INTRODUCTION

Robots are generally used to assist humanity in diverse fields, from daily life works to applications in various industries like logistics, production floors for the movement of raw materials, tools, and finished products. The recent demands of people and industries have led to significant development in robot science. Mobile robots are one of the most prominent areas in this field.

Mobile robots are capable of navigating and interacting within an environment using sensors and actuators. This requires the collection of data from the robot along with analysis and understanding to improve the performance of the robot. The robot's task scheduling and configuration changes are also managed at a centralized level. This calls for the deployment of an IoT framework on the cloud to effectively maintain and improve the performance of the robot. Such a collaboration of an IoT-based platform will allow the robots to communicate with humans over the network, making use of the cloud. This paper demonstrates one such use case where such IoT-based platforms is used to communicate with autonomous mobile robots (AMRs) to carry out tasks and functions.

There are mainly two classes of mobile robots, namely autonomous mobile robots (AMR) and autonomous guided vehicles (AGV). The autonomy mechanisms of AMR and AGV cause the main difference between these robots. Physical guidance forms the foundation for an AGV and it generally navigates along a predefined path in a predetermined environment. AGVs are typically used for applications that involve repetitive tasks and they are often designed for specific tasks. The programmer needs to predefine the tasks to be performed by the AGV since it doesn't have an artificial intelligence-based decision mechanism. The main drawback of AGVs is the execution of dynamically changing tasks.

An AMR is capable of navigating through an unpredictable environment and locate itself in the model by sensing the environment parameters. Specialized planning algorithms may be used to enable AMR to make a navigation plan and optimize it. AMR does not have a predefined navigation plan initially. AMR localizes itself on the map while creating a map of the environment using the data from the associated sensors. This is referred to as simultaneous localization and mapping (SLAM). SLAM facilitates the dynamic revision and improvisation of the programmer's created navigation plan. AMRs have been widely used in different fields due to their pioneering features. AMRs have more complex hardware and software design than AGVs due to a large number of sensors and other components. The complicated control systems in AMRs work in a coordinated manner in compliance with the data stream [1].

Future applications will make use of a fleet of AMRs working together to fulfill a common goal set by the enterprise system with minimum human intervention. Such an association can be obtained by enabling an IoT platform that allows communication of robots with humans over networks including the internet. A cloud infrastructure permits offloading computationally expensive tasks onto a remote server. This reduces the onboard power and computational requirements.

This paper proposes an architecture for communication of Autonomous Mobile Robots (AMR) with the Cloud. The work focuses on the software implementation of communication between AMR and cloud. Section II gives an overview of the robotics middleware Robot Operating System (ROS) and Message Queuing Telemetry Transport (MQTT) protocol which is integrated with ROS for secured data sharing on ROS-enabled

robots. Section III describes the main components of the proposed system architecture. Section IV presents the design of the proposed system. Section V demonstrates the implementation and simulation results. Finally, Section VI concludes the paper with the future scope and research directions.

2. OVERVIEW ON ROS AND MQTT

Robot Operating System (ROS) is a "thin, message-based, peer-to-peer" robotics middleware designed for mobile applications. The robotic applications, which are generally designed as message-based applications, are connected with ROS. Thus, the user can easily monitor and dynamically control all the tasks under execution on the robot. The software and hardware interaction among ROS, AMR application and user is shown in Fig.1.

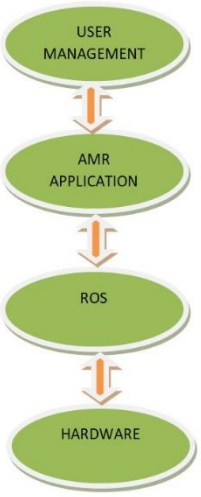

Fig.1. Software and Hardware interaction

An open-source Robot Operating System (ROS), which is one of the most widely used development platforms, provides low-level device control, high-end capabilities and good community support [2]. ROS is a collection of libraries and tools that aim to make the tasks of creating robust and complex robot behavior. It provides a communication layer, a data and programming structure, and many useful tools for visualization and debugging. ROS with its unprecedented direct access to state-of-the-art developments in the robotics field, is widely accepted by the scientific world and is popular among research enthusiasts from various domains of robotics. A publish/subscribe system with a Remote Procedure Call Protocol (RCP) is used for communication in ROS. Here, the data is sent using TCP/IP or UDP/IP in clear-text. With the increased use of ROS in various robotic applications, it has been exposed that the applications that are built in this way are vulnerable to cyber-attacks due to the lack of security mechanisms. The protection of data and the authentication of entities during communication should be a prime concern [3].

From a security point of view, an eavesdropper could easily attain unencrypted data and potentially damage the system by taking control over the established insecure communication. For example, the acquisition of surveillance of robots by an attacker or eavesdropper could damage the assets and the associated humans in the environment [4]. Cyber-attacks on the AMRs used for surveillance could also enable the penetration of intruders into proctored spaces and buildings, which could lead to catastrophic privacy issues [5]. Therefore, for a secured data sharing on the ROS-enabled robot, keeping the direct access of remote users to ROS remote, and for secured communication between the robot and the clients, a lightweight Message Queuing Telemetry Transport (MQTT) protocol is integrated with ROS. MQTT's built-in capabilities help in providing authentication, confidentiality and secured data sharing while establishing secure communication between robots and remote clients [6].

MQTT is a lightweight, open, simple and easily implementable ideal communication protocol for Machine to Machine (M2M) communications and IoT. It experiences lower message overheads than other protocols like Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), and Hypertext Transfer Protocol (HTTP) due to its fixed-sized header of 2 bytes. MQTT can also utilize SSL/TLS which ensures standard security and privacy. Although protocols like AMQP and HTTP also provide security using *Secure Sockets Layer*/Transport Layer Security (SSL/TLS), Simple Authentication and Security Layer (SASL), and IPSec, they require higher bandwidth and other resources and have low reliability [7]. Even though CoAP needs less bandwidth and resources than MQTT, it does not offer Quality of Service (QoS) and only ensures unreliable communication using User Datagram Protocol (UDP). Mosquitto, Mosca, and Paho are some commonly used open-source software that uses the MQTT protocol.

Among the simple and complex authentication mechanisms offered by MQTT, the commonly used basic security mechanisms include username, password, and digital certificates, which can be radically implemented. Public Key Infrastructure (PKI), X.509 digital certificates and SSL/TLS (Transport Layer Security) are some of the sophisticated mechanisms used to offer authentication and data privacy in communication networks. MQTT also offers authorization mechanisms that control the client's access rights to resources. With regard to ROS, all the clients can publish and subscribe to all the topics by default access right. Access Control Lists (ACL) can be used to specify the client's access rights when they are connected to some IoT platform. It is a list of permissions that allows users to access various topics and decides the actions that can be carried out with these topics [6].

Several IoT platforms such as Kaa, DeviceHive, OpenIoT, and ThingSpeak are available which help programmers to connect devices to the Internet. In this paper, the Kaa IoT platform is used since it provides all the functionalities needed for operating on large-scale data. Implementing the gateway and server on the Kaa IoT platform guarantees load balancing and high availability [8]. The scope of this paper is to define the system architecture of AMRs and identify a cloud infrastructure where the IoT functions for the management of indoor robots will be deployed and implemented.

3. COMPONENTS OF THE ARCHITECTURE

3.1 ROBOT OPERATING SYSTEM

ROS is the most commonly used framework for the software development of robots. ROS performs computation as nodes based on XML-RPC [9] using a publish/subscribe model. *catkin* is the official build system of ROS. With catkin, the *catkin_make* command is used which will build all packages in the workspace [10].

ROSCORE runs on the machine which is called the master. The nodes run on the system and all the nodes need to connect to the master. Based on the publish/subscribe messaging model, the data from the nodes created by builder tools are collected by the roscore, and are published as "/rosout" of a topic identifier. It is possible to collect data from the roscore with the help of the states of all programs and nodes which are available [11]. The nodes may be launched by a single launch file as well. The system can be set up by the following steps: (1) Design a secure Cloud package file required for communication (2) Establish the topics and messages for each node (3) Run roscore on master (4) Launch the nodes.

3.2 MQTT Broker – PAHO

The MQTT broker participates as a mediator in the communication process. The connected device sends messages in the broker direction, and hence removes the need to specify the address of the member in the network that requires the information. The incoming data from devices are published in a representative topic in the broker and gives access to any member subscribed to the same topic [12]. Since programs are running in python, MQTT support is provided by a Python package called PAHO. PAHO provides open-source client implementation of MQTT messaging protocols for IoT in Python.

3.3 IoT PLATFORM

IoT platform facilitates end-to-end development experience for IoT product development. It can create fully featured scalable, compatible and secure IoT applications on a single platform. It provides customizable microservice architecture and open protocols.

It ensures extraordinary flexibility of IoT devices and application management. IoT platform allows the developers to choose the technology, type of deployment, and third-party integrations. It provides various technological advantages like freedom of deployment, open IoT protocols, gateway support, maintaining records of application versions etc. The platform provides various business advantages like cost efficiency, simple and straightforward licensing plans, embedded license, white labeling, own custom-based solutions, flexible support options from vendors.

3.4 WEB INTERFACE

Web interface permits users to control the IoT system remotely. User interface (UI) elements are created using basic HTML/CSS/JS code. This HTML code incorporates a state machine, which presents the user with the login screen or options to access their account and fleet details. REST APIs provided by the IoT Platform is implemented in the JavaScript (JS) backend to allow communication between the client and cloud server [13].

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The proposed architecture shown in Fig. 2 has two main components that communicate with the IoT Cloud Applications: (1) AMR System (2) Web Application. The AMR system has two zones, namely trusted zone and untrusted zone. The trusted zone includes the Secure Remote Manager. This module ensures the security of other modules in the untrusted zone. The AMR system establishes communication with the Cloud Platform using MQTT Protocol, and it sends and receives data regarding telemetry, command execution software updates etc. The information from Cloud is handled by the IoT custom web dashboard using REST APIs. The gateway is implemented using the ROS framework on a Linux system. The gateway receives the commands when the data is sent to the server. The main server is implemented on the IoT platform.

4.2 ROS - KAA IoT PLATFORM

Packages are the most fundamental units of build and release. To implement communication between ROS nodes and the IoT Platform, a package called Secure Remote Mnager is needed. This shall allow two processes, Data Aggregator and Remote Data Manager to run simultaneously. The function of the Data Aggregator is to collect AMR data from other AMR subsystems and time synchronizes the messages and sends them to the Remote Data Manager. The Remote Data Manager performs two functions: (1) it uses the MQTT protocol to send the data to the Cloud Platform (2) it receives and processes the commands issued by the user at the Cloud Platform.

The AMR subsystem nodes shall include the perception node, planning node, control node, and system node. These nodes publish messages on different topics, which are then collected by the data aggregator. The data aggregator node time synchronizes the incoming messages from other subsystem nodes. This time-synchronized integrated message is received by the Remote Data Manager. Remote Data Manager then sends this data to the IoT platform using the MQTT protocol. The received data is visualized and monitored. Using the IoT platform, command execution may be performed for assigning tasks and controlling the movements of AMR. The commands to create/delete tasks and handle/change movements are issued by the user and are received by the Remote Data Manager. The commands and actions to be executed are then sent to the planning node. Depending on the task or movement, commands can be assigned to various other nodes in ROS. Similarly, Over the Air (OTA) updates can be handled through the Cloud Platform.

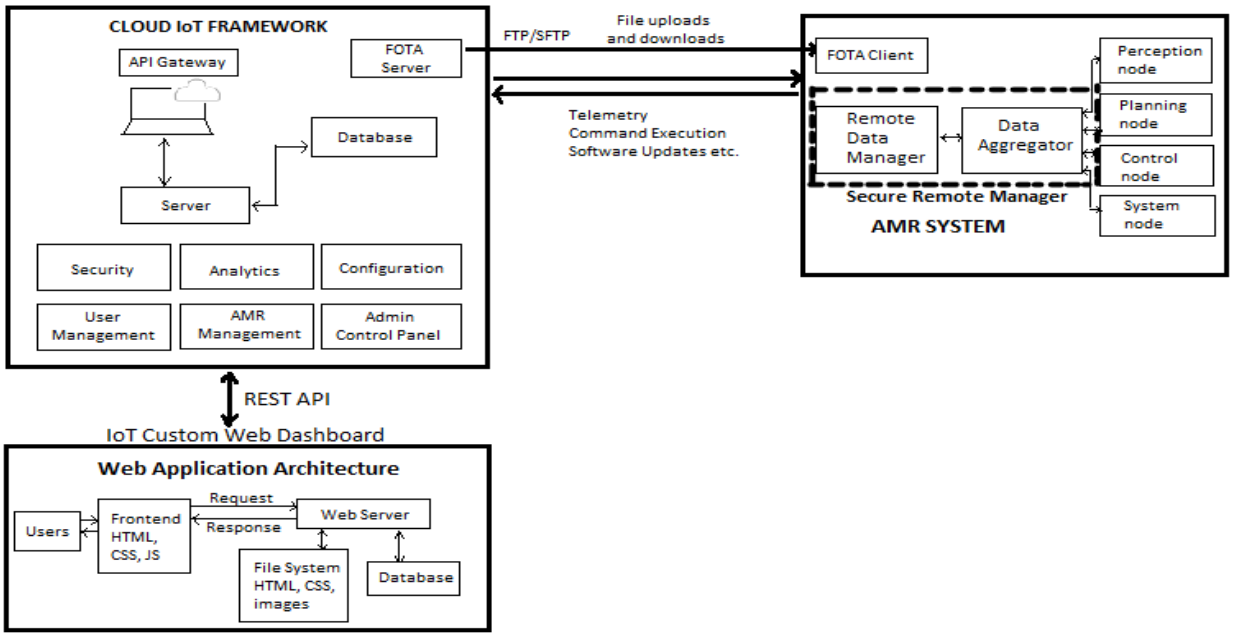

Fig.2. System Architecture

4.3 IoT PLATFORM AND USER COMMAND EXECUTION USING REST API

The platform provides several REST APIs for data collection, command invocation, management of endpoints etc. A web interface for the user is developed to visualize the AMR data. To get the recent time series data, IoT platform provides the following REST API request: /applications /{applicationName} /time-series /last. Its response would be in the form of arrays of the latest data points which are mapped by time series names and endpoint IDs [7]. The command execution on the user's web interface is also accomplished using the REST API of the IoT Platform. To invoke commands on an endpoint, provides the following REST API request: /endpoints /{endpointId} /commands /{commandType}. If the commands are executed within the specified 'awaitTimeout', appropriate command execution results are obtained. Else, the resource URL of the command present in the 'location' header results. The REST API request to get the command information is given by /endpoints /{endpointId} /commands /{commandType} /{commandId}. Its response will be a JSON object with properties of the command like commandId, status command type [14]. The interface also indicates the status of the command under execution as executed/pending/not received according to the status code of the request-response.

5. IMPLEMENTATION AND SIMULATION RESULTS

This section evaluates the cloud communication between AMR and web applications. Based on the proposed design, simulation was carried out and the obtained results are as follows. "Melodic" version of ROS was installed on the virtual machine, following which a ROS workspace was created with catkin. The workspace contains the package called "Secure Remote Manager", which provides the application with the classes of Remote Data Manager and Data Aggregator. The roscore and the applications are then launched. The data on the topics published by the planner and system nodes are shown in Fig. 3. The Rqt_plot in Fig.3. helps to plot any numeric value published by ROS topics.

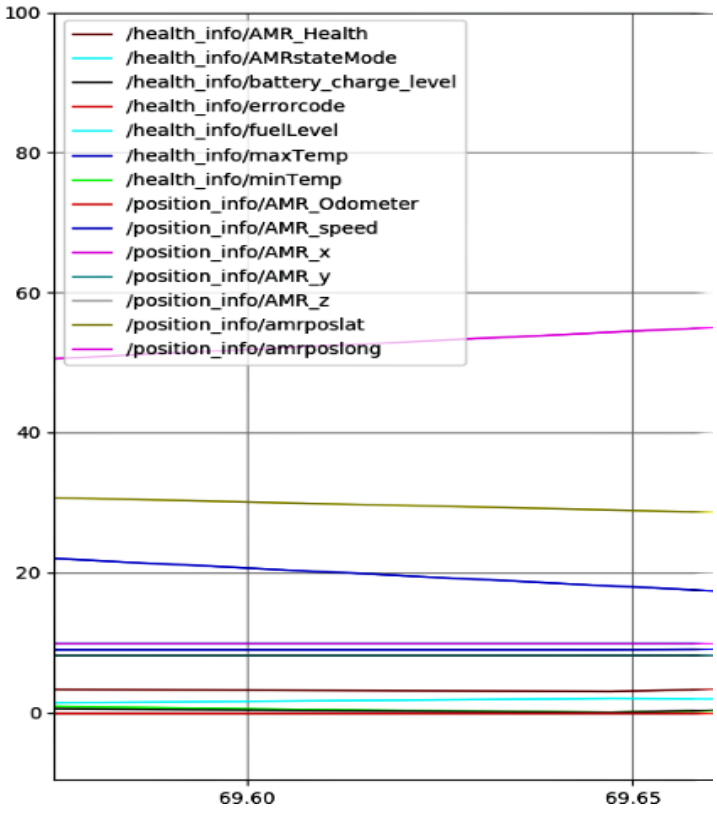

Fig.3. Rqt_plot for topic data from system and planner node.

The gateway was implemented on Linux using the ROS framework. The connections between the IoT gateway and server were programmed using the Python programming language. While sending the data collected from the system to the server, the gateway also receives commands from the server. The main server is implemented on the IoT platform, which is a highly reliable open-source middleware platform used for implementing numerous end-to-end IoT solutions. It manages data in its backend infrastructure through a server and uses Apache Zookeeper to coordinate the services.

Interconnected nodes form a IoT cluster which is related to a specific Kaa instance. Kaa cluster needs NoSQL and structured query language (SQL) database instances for storing endpoint data and metadata respectively. Maria DB is used as the SQL database. MongoDB is used as the NoSQL database, where the gathered data may be stored. MongoDB is an operational NoSQL database with the scalability and flexibility that one wants with querying and indexing. It allows for operational scalability, greater querying flexibility, faster read/write operations, higher availability, and simplicity. It also provides an active load balancing and hybrid encryption system for security that is based on RSA with a 2048-bit key pair and AES with a 256 (512)-bit key.

A web portal is drafted to visualize the obtained data. It gathers the data from the MongoDB database and performs visualization. Fig. 4 shows the received data on the Kaa IoT platform.

Hence, the users can perceive real-time information from sensors and the tasks carried out by the AMR. Algorithms are designed on the web portal to send control commands. The RESTful Application Programming Interface (API) is used for sending unicast notifications from a web portal to the server. The server passes the control command to the selected endpoint (gateway) when the notification is received. The gateway also carries out the dispatching of control commands to the sensors to be applied. SSL/TLS certificates were created to ensure secure communication and the device was authenticated using X.509 certificate.

The dashboard displaying the device orientation and controls for command execution is shown in Fig. 5. Commands like MOVE FORWARD, MOVE BACKWARD, MOVE RIGHT, MOVE LEFT, STOP, CHANGE SPEED, etc. have been included. Fig. 6 and Fig. 7 shows the response to the commands MOVE FORWARD and MOVE RIGHT that are received at the system. In a similar manner, software updates (OTA) are received and the system is notified.

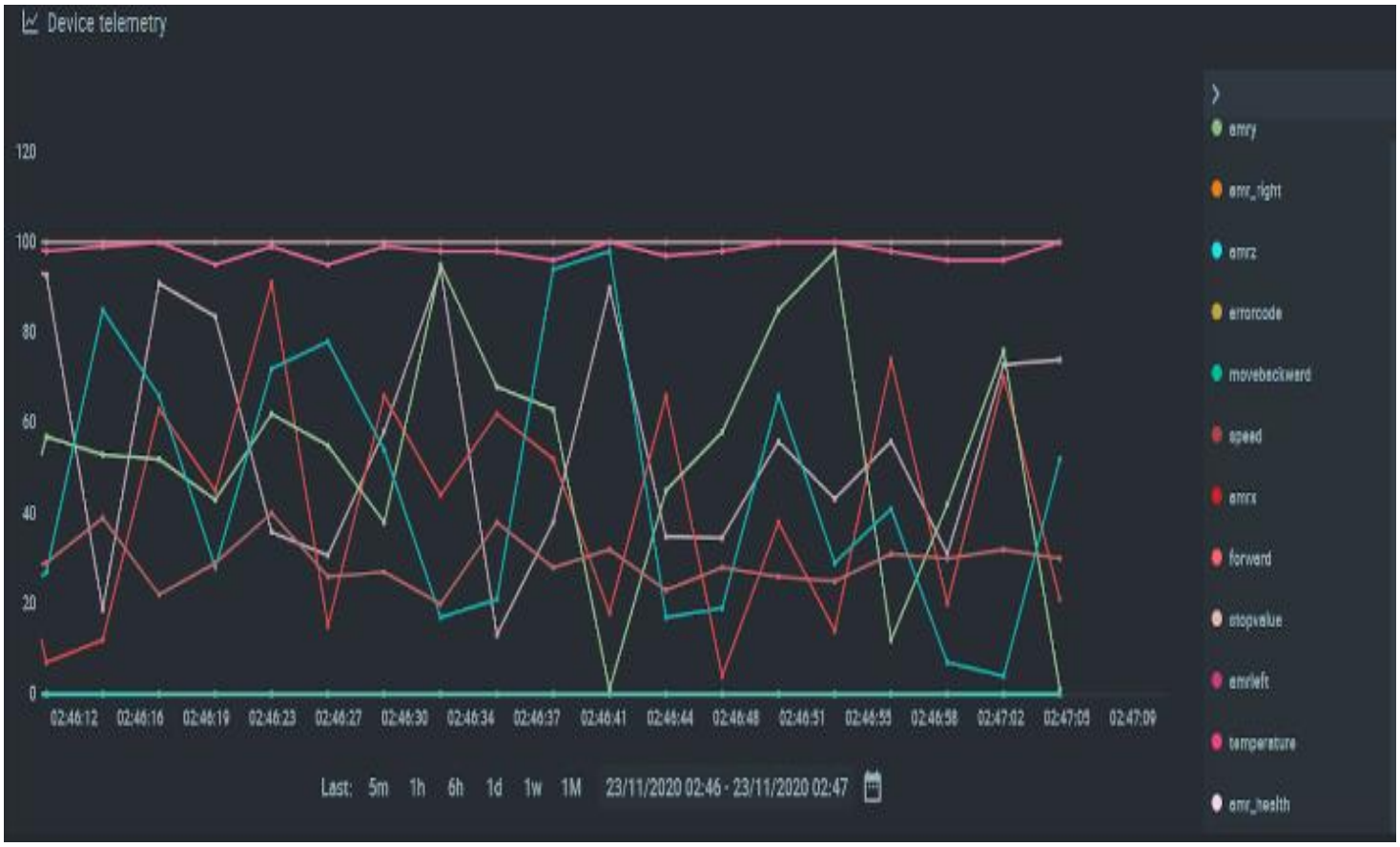

Fig.4. AMR Telemetry data

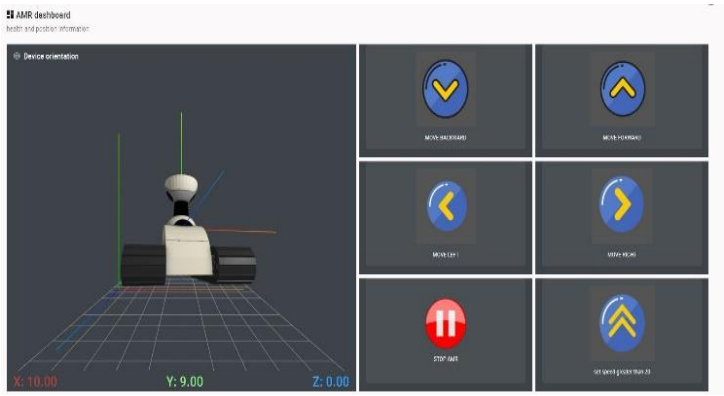

Fig.5. AMR Dashboard

```
<--- Received "MVFRWD" command on topic kp1/bv6uk05bhnjfabjeknvg-b1/cex/my_id/command/MVFRWD/status

Move Forward Command Received

Payload :[{"id":505584554,"payload":{"distance":10}}]
Move the AMR 10 distance in forward direction
```

Fig.6. Received Command on executing MOVE FORWARD

```
<--- Received "MVRHT" command on topic kp1/bv6uk05bhnjfabjeknvg-b1/cex/my_id/command/MVRHT/status

Move Right Command Received
Payload :[{"id":269355899,"payload":{"distance":20,"angle":60}}]
Move AMR right with turn angle 60 and distance 20
```

Fig.7. Received Command on executing MOVE RIGHT

Using the implementation and simulation results, functionality, and communication between ROS nodes and the IoT platform in scenarios like sending telemetry data to the cloud, implementing security protocols, software updates, sending commands to perform various tasks and functions from the cloud etc are checked. The necessary programs are written for the appropriate working of the robot by taking into consideration the sensors and actuators used.

The proposed architecture in this paper is a Device-to-Gateway model, where IoT devices access cloud services via an in-between device. In this model, the application software working on a local gateway like a smartphone acts as the intermediary or the hub. Unlike this model, most of the existing communications are based on Device to Device and Device to Cloud communications alone. Device to Device communication allows two or more devices to connect and communicate among them over IP networks or the internet. Device to Cloud communication helps to directly connect IoT devices to cloud services. However, the Device-to-Gateway model gives added advantages of security and data or protocol translation apart from other features in most of the use cases [15].

6. CONCLUSION

In this paper, the design of an Autonomous Mobile Robot (AMR) modeled for the Robot Operating System (ROS) is presented. With the proposed architecture, communication has been established with the robot using the IoT platform. The system is implemented with a IoT platform-based central server. The communication architecture between the IoT platform and the ROS framework in the robot has been analyzed. Telemetry collected from the robot is collected and visualized on the web portal. The data from the AMR is used for analytics purposes which can help in optimizing decision making and carrying out the tasks. Based on the implementation results, Kaa is a highly reliable and flexible platform, with good scalability in terms of data transfer, availability, transaction and concurrency. The simulation was carried out and satisfactory results were obtained. The paper illustrates practical end-to-end communication between a single user and an AMR. This methodology can be further scaled and used for fleet management of AMRs. As the IoT Platform is highly scalable, the insights from this work can be considered for future works as well. The IoT platform can be extended for industrial and commercial purposes by including analytics, fleet management etc.

REFERENCES

[1] M. Köseoğlu, O. M. Çelik and Ö. Pektaş, "Design of an autonomous mobile robot based on ROS," *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, Malatya, Turkey, pp. 1-5, 2017. doi: 10.1109/IDAP.2017.8090199.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, 2009.

[3] S. Morante, J. G. Victores, and C. Balaguer, "Cryptobotics: Why robots need cyber safety," *Frontiers in Robotics and AI*, Vol.2, Article 23, pp. 1-4, 2015. https://doi.org/10.3389/frobt.2015.00023

[4] D. Portugal, S. Pereira, and M. Couceiro, "The role of security in human-robot shared environments: A case study in ros-based surveillance robots," in *26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pp. 981–986, 2017.

[5] D. Portugal, M. A. Santos, S. Pereira, and M. S. Couceiro, On the security of robotic applications using ROS, ch. In *Artificial Intelligence Safety and Security*. CRC Press, Taylor & Francis, 2018.

[6] M. Mukhandi, D. Portugal, S. Pereira and M. S. Couceiro, "A novel solution for securing robot communications based on the MQTT protocol and ROS," *2019 IEEE/SICE International Symposium on System Integration (SII)*, Paris, France, 2019, pp. 608-613, doi: 10.1109/SII.2019.8700390.

[7] "Endpoint Time Series service API documentation" Available:https://docs.kaaiot.io/KAA/docs/v1.1.0/Features/Data-collection/EPTS/REST-API/. Accessed on 1 October 2020.

[8] M. Mehr Nezhad, M. H. Y. Moghaddam and M. Asadi, "Joint Peak Clipping and Load Scheduling Based on User Behavior Monitoring in an IoT Platform," in *IEEE Systems Journal*, vol. 15, no. 1, pp. 1202-1213, March 2021, doi: 10.1109/JSYST.2020.3009699.

[9] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," *ICRA Workshop on Open Source Software*, 2009.

[10] G. Hu, W. P. Tay and Y. Wen, "Cloud robotics: architecture, challenges and applications," in *IEEE Network*, Vol. 26, No. 3, pp. 21-28, May-June 2012, doi: 10.1109/MNET.2012.6201212.

[11] P. Amontamavut and E. Hayakawa, "ROS Extension of Blue-Sky web based development environment for IoT," *2016 Fifth ICT International Student Project Conference (ICT-ISPC)*, Nakhonpathom, Thailand, 2016, pp. 45-48, doi: 10.1109/ICT-ISPC.2016.7519232.

[12] Efrain, G. B. D., del Carmen, V. C. A., Fernando, E. C. L., & Cesar, L. M. D. (2018). Implementation of an IoT Architecture based on MQTT for a Multi-Robot System. *2018 IEEE Third Ecuador Technical Chapters Meeting (ETCM)*.

[13] K. Kumar, J. Bose and S. Tripathi, "A unified web interface for the internet of things," *2016 IEEE Annual India Conference (INDICON)*, Bangalore, India, 2016, pp. 1-6, doi: 10.1109/INDICON.2016.7839142.

[14] "Command Invocation REST API documentation"Available at:https://docs.kaaiot.io/KAA/docs/v1.3.0/Features/Command-invocation/CEX/REST-API. Accessed on 1 October 2020.

[15] "The Four Internet of Things Connectivity Models Explained" Available at: http:// www.inetservicescloud.com/ the-four-internet-of-things-connectivity-models-explained/. Accessed on 1 May, 2021.