2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)
September 12-14, 2018, Madrid, Spain

Generic ROS-based Architecture for Heterogeneous Multi-Autonomous Systems Development

Mustafa Alberri, Sherif Hegazy, Mohamed Badra, Mohamed Nasr, Omar M. Shehata, Elsayed I. Morgan
Multi-Robot Systems (MRS) Research Group
German University in Cairo
Cairo, Egypt
mrs.lab.guc@gmail.com

***Abstract*—Autonomous systems have been considered a strong field of attraction to the research community in the last few decades. Accordingly, this paper proposes a design and implementation of a generalized Robot Operating System (ROS) based architecture applicable for multi-autonomous systems in heterogeneous environments. The architecture allows inter-robotic data exchange for Vehicle-to-Vehicle communication by integrating embedded systems into the architecture structure. Moreover, the proposed architecture is a hierarchical one composed of three layers connected through various communication protocols, and characterized by being affordable, powerful and expansible. The architecture has been tested on three different platforms; autonomous mobile robot, autonomous vehicle and autonomous quad-copter. Results of the conducted experiments showed stability and successful operation of the proposed architecture.**

***Index Terms*—Hierarchical architecture, ROS-based, Embedded Systems, Autonomous Systems, Multi-robotic systems**

I. Introduction

In the last few decades, the concept of autonomy has been introduced to the world of robotics, where an autonomous robotic system is a system that has the freedom to control and govern self affairs [1], which means that the robot relies on guidance devices to navigate through a predefined route. Navigation is one of the most fundamental functions of an adaptive autonomous system, which is also a method of classification between different robotic platforms. Autonomous robotic systems target a combination of complex applications to imitate artificial intelligence behaviors such as self localization, mapping, and path planning. To properly implement such applications with guaranteed optimum results, autonomous systems need a firm base which provides convenient environmental conditions for the operation process. In the world of robotics, this base is defined as the system architectural design.

Autonomous systems have been one the leading fields of interest in the reasearch community, due to the extensive range of applications offered by such systems. Autonomous Vehicles (AV) development can be used as an alternative means of road transportation instead of manually driven vehicles. The deployment of AVs in public roads guarantees more efficient energy consumption while freeing drivers from the burden of driving to complete other tasks [2]. In 2014, 7Starlake and Easymile introduced the EZ10 autonomous bus and since then it has been deployed in over 50 sites and 14 countries with varying routes [3]. Military institutions are globally attracted to Autonomous Surface Vessels (ASV) and Autonomous Aerial Vehicles (AAV) for surveillance and safety applications [4]. The US marine corps have been working with several autonomous helicopters for fire scouting such as the MQ-8B unmanned helicopter manufactured by Northrop Grumman [5]. In addition, the latter can be exploited in a wide range civilian applications, for example, AAVs are currently used for home deliveries and ASVs can be used for civilian transport over water bodies. Autonomous Mobile Robots (AMR) have also been used in industrial applications and factories to reduce manual labor and save expenses [4].

However, working with single robots have created limitations represented in short range of perception and high cost of manufacturing [6]. It has been established that working with multiple robots with a simple design to perform complicated tasks has much more superior advantages than working with a single powerful robot with a more complex design. The development of multi-robot systems (MRS) has opened up a whole new world of applications that could not have been feasible to execute using single-robotics. The deployment of MRS allows the possibility of inter-vehicular communication, as well as bidirectional infrastructure and pedestrian communication, this can be largely beneficial for vehicular and robotic safety.

In this research, we have designed and implemented a low-cost, high performance, generic ROS-based architecture for autonomous systems. The proposed architecture is consisted of multiple layers. Testing and validation of the proposed architecture was done on three different platforms: an Autonomous Mobile Robot (AMR), an Autonomous Vehicle (AV) and an Autonomous Quad-copter.

In this paper, section I presents a short introduction about autonomous sytems and some of their applications. Section II defines the problems faced by robotic systems developers, their corresponding solutions, and some related work. Section III highlights details about our proposed hierarchical architecture, and a description about its multi-layer structure [7]. Section IV presents the conducted experiments, testing conditions and their results, which are thoroughly explained. Finally, section V provides a conclusion to our work and some future objectives.

II. Background

Generally, a large portion of robotics platforms and frameworks lack the capability of being usable by several applications. This creates growing complications for robotics developers, which means that each individual researcher needs to arm himself with a wide range of complex scientific fields in order to equip the robot with sufficient software modules for a valid and successful operation, negatively affecting the attraction to robot development field for some individuals. In order to compensate for this, Willow Garage has come up with a solution by introducing a free open-source robotic software that supports the reuse of generalized packages and libraries requiring minor to no modification to the source code. The framework is called Robot Operating System (ROS) and was released in 2007 [8]. Supported by an extensive knowledgeable community of researchers, ROS guarantees to provide a robotic software base for interested developers covering almost all environments and architectures, including both homogeneous and heterogeneous applications.

Robot Operating System (ROS) is a software framework through which robotics researchers can control their robots using a personal computer. It is like a brain to the robot. Since ROS is an open-source robotic platform [9], interested individuals can download and use packages and pieces of code where other people have already implemented, preventing developers to reinvent the wheel for every separate application and saving tremendous amount of time. Before ROS, robot development was much more challenging and consumed a lot of resources because every system was unique and lacked standardization, but ROS encourages the reuse of good ideas by sharing and addressing the common problems robotics developers face. The main programming languages used are C++ and Python. ROS is currently supported and tested on Linux-based operating systems.

An implementation and simulation of a Multi-Robot Task Allocation (MRTA) approach based on ROS that emphasized on the communication between several heterogeneous robots was proposed in [10]. Another implementation of a detailed scheme for management of a fleet of Autonomous Mobile Robots (AMR) based on Raptuya Cloud Robotics Platform was presented in [11], while uncovering the limitations of such platform by implementing the same system on ROS. Moreover, the work in [12] integrates Robot Operating System (ROS), V-REP (Virtual Robot Experimentation Platform) simulation environment and JADE multi-agent system for the cooperation of a group of UAV (Unmanned Aerial Vehicle) and UGV (Unmanned Ground Vehicle) robots. The main conclusion that was agreed upon for the aforementioned ROS-based implementations was that using the ROS framework for deploying multi-robot systems provided several advantages in comparison to other robotics architectures.

Most robot software modules based on ROS are optimized to be running on powerful and high specification computer processors, generating other problems that needs to be tackled including the ability to operate remotely which requires a compressed hardware design with minimization to the robot's physical size. Another problem with currently available robot controllers in the market is that it mainly targets economically backed institutions such as companies and factories, making it unaffordable to students and individual researchers. Embedded systems and sub-systems provide an efficient solution to overcome this predicament, and integrating ROS into embedded systems opens up a whole new order for innovation in the field of robotic development by providing powerful, reliable and low-cost solutions. Working with ROS on Linux-based PCs causes real-time limitations for sensor information that needs to be transmitted with regard to timing regulations of the application, working with a blend of ROS-enabled systems and real-time sub-systems provides more possibilities for applications [13]. Another implementation of a communication structure between heterogeneous Multi-Robot Systems by utilizing a new ROS-based hybrid architecture was introduced in [14]. The architecture consists of a central server for complex processing joined with embedded boards as robot nodes for performing real-time processing. However, The main focus of this paper is based on non-real time computations.

In order to incorporate autonomous systems with ROS and embedded technology, the implementation of a proper ROS-based architectural design that integrates high level controllers with low level interfacing is an essential initial step towards accomplishing that. In this paper, we propose a generic ROS-based architecture for autonomous systems that is boosted with the following characteristics:

1) Modularity and separation: armed with the ability to be separated and combined easily, our architecture is ready for almost any type of control modules and applications.
2) Flexibility and ease of use: adding and removing sensors and actuators to our system requires simple to no editing in the code.
3) Expansibility: the architecture has the ability to be expanded for absorbing more components and software modules.
4) Affordability: our proposed architecture is comprised of low-cost high power hardware components that is economically convenient for all levels.
5) Usability and shareability: the robotic research community can benefit from the architecture by applying it to their own applications successfully [7].

III. Proposed Architecture

The proposed architecture for this paper is a hierarchal architecture. It is comprised of three main levels operating simultaneously but each has it's own functionality. The sophistication of this system relies on the division of objectives among the layers, and performing tasks simultaneously. But first, let us introduce some basic communication concepts in ROS.

A. Robot Operating System (ROS)

The basic understanding of data transfer in ROS, is that data is transferred in the form of messages. Messages can be in the

form of predefined data types, or they can be custom made. Messages are exchanged between nodes through topics. The Fig. 1 shows the process of data transfer [9]. In order for the

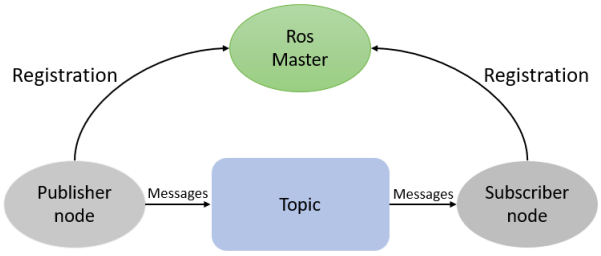

Fig. 1. Basic communication in ROS [9].

nodes to be ready for exchanging messages, they should register to the same ROS Master node for proper data exchange. A master is initialized by running the command 'roscore' in ubuntu terminal. The node which sends messages is a called a publisher node, and the node that receives messages is called a subscriber node. The publisher node sends messages to a topic, and for the subscriber node to receive the message, it should subscribe to the same topic. A topic can have more than one publisher and more than one subscriber. The previously explained process is the traditional communication method in ROS. However, the *multimaster_fkie* package offers communication with multiple running ROS Masters, which enables communication between several robotic entities for inter-vehicular communication.

B. Hardware Description

This section provides a description for the functionality and significance of each level in the proposed hierarchical architecture.

1) Higher Level: All software modules that requires high processing power such as control algorithms, sensor fusion, path planning, etc. are located in this level. This level receives filtered and simplified sensor data from the intermediate level for processing. Monitoring and assigning tasks and sub-tasks to active nodes in the system are the main functions of this level. A Linux-based computer handles all functionality in the top layer and the ROS version is ROS kinetic.

2) Intermediate Level: The connection between the lower level and the higher level is materialized through this level in the architecture. The lower level sends raw sensor data to this level for filtration and conversion into computable data which is by turn sent up to the higher level. Actuating signals for the motors are also recieved by this layer from the higher one. Communication between the higher and the intermediate level is over a common wireless LAN network and data is exchanged through ROS topics and the *multimaster_fkie* package. This level is comprised of Raspberry Pi 3 Model B running on Linux (Ubuntu Mate) with ROS kinetic installed.

3) Lower Level: This layer in the architecture is considered an interface for sensors and actuators. It 'reacts' to the hardware components by receiving raw sensor data and sending it up the hierarchy and receives actuating signals for moving the motors. Communication between the lower and the intermediate level happens through the I2C communication protocol. This level is implemented using the Atmega328p micro-controller, it can be more than one driver depending on the application and the number of sensors and motors, since using the I2C protocol with the Raspberry Pi as master have provided the opportunity of using up to 128 slave micro-controllers.

C. Software Description

This section provides a description to the functionality and significance of the implemented ROS packages in different levels of the proposed hierarchical architecture, as well as the implemented code in the low level drivers.

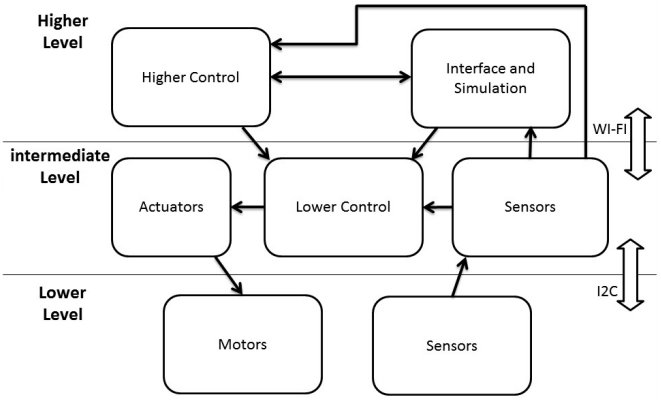

Fig. 2. Block diagram describing software architecture and packages.

1) Higher Control: This package is placed in the aforementioned high level computer, which makes it the perfect candidate for embracing nodes and modules that necessitates high computational power such as trajectory tracking and task allocation. This package mainly comprises of nodes implemented on Matlab and Simulink, although it can also contain python/C++ nodes for controlling the robot and logic computation. Nodes in this package receives its inputs from the sensors package in the intermediate level for feedback and observation, whereas the output of these nodes are sent to the intermediate level through Wi-fi for diagnosing.

2) Interface and Simulation: The proposed architecture allows controlling the actuation and movements of the robotic system for open loop control by the user, which is materialized by this package and visualized by a simple Graphical User Interface (GUI). The implemented interface further allows the monitoring and observation of available robot nodes in the common network by viewing all sensor readings and providing warnings in case of network crashes and hardware failures. In addition, any nodes that deploys simulation algorithms are located in this package.

3) Lower Control: Located at the center of the architecture, this package is responsible for handling most of the major communication and rate of data transfer between different

levels of the architecture. The package is composed of three main ROS nodes:

- Diagnose Node: checks the system status such as network connectivity and battery level and calls for immediate action in case of partial system failure.
- Control Node: represents a backup control algorithm to ensure overall system stability and safety, in case failure of communication between the intermediate and higher level due to reasons including breakdown of multimaster connection or network breakdown.
- Heartbeat Node: this node is responsible for handling the frequency of data transfer between all entities of the architecture, overall system feedback sampling rate can be changed by simply modifying a ROS parameter.

4) Actuators Package: This package primarily consists of one node for transporting actuating signals to the lower level drivers which in turn moves the motors.

5) Sensors Package: Everything concerning feedback and sensor information are handled by this node located in the intermediate level of the architecture. After receiving raw sensor data from the lower level, filtration and conversion into virtual processable data takes place in this package. According to the application, sensor fusion nodes can be executed in this package for further exploitation of the Raspberry Pi processing power.

6) Motors: This package is not a ROS package, it is made up of C/C++ code for programming the Atmega328p chips. It is responsible for reading rpm values and Pulse Width Modulation (PWM) signals from the I2C bus sent by the Actuators package in the mid-level and writing them directly to the motors for actuation.

7) Sensors: Since it is located in the low-level of the hierarchy, it is also comprised of C/C++ code for reading raw sensor information directly from the installed sensors. Both this package and the previous one are mainly considered as interfacing modules for hardware components.

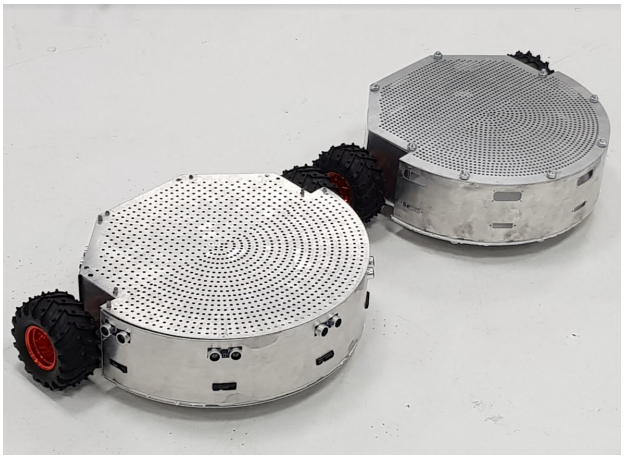

Fig. 3. Autonomous mobile robots hardware.

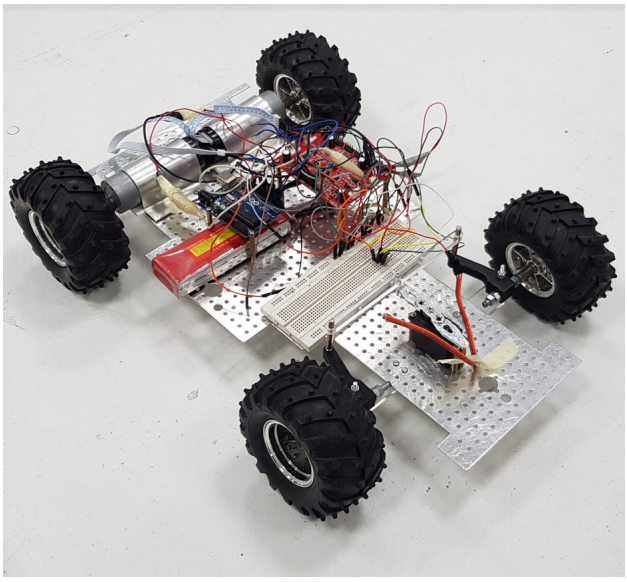

Fig. 4. Autonomous vehicle hardware.

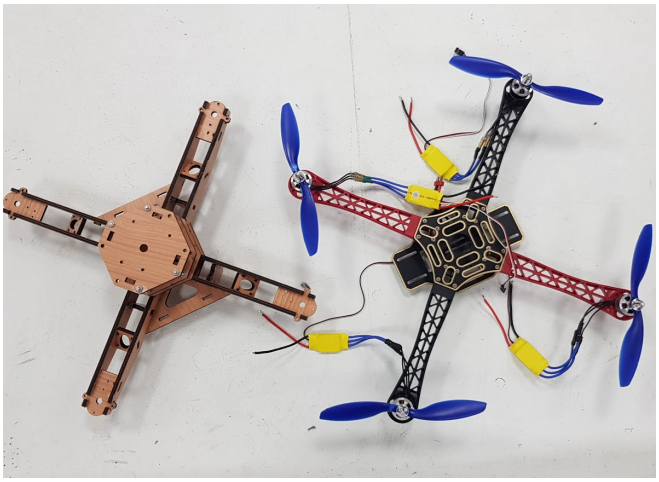

Fig. 5. Autonomous quad-copters hardware.

IV. Experiments and Results

In this section, the testing and validation of the proposed architecture is conducted and illustrated by a simple experiment in a laboratory setting. The architecture was tested on three platforms: an autonomous mobile robot, an autonomous vehicle and an autonomous quad-copter, which were designed and manufactured by students in the German University in Cairo, Egypt as shown in Fig.s 3, 4 and 5. The computer used in the experiment had an Intel® Core i7 2.40 GHz processor with 16 GB of RAM running a 64-bit ubuntu 16.04 operating system. Communication between the Raspberry Pi and the high level computer was over a wireless LAN network through the *multimaster_fkie* package, and connection between the Raspberry Pi and the Atmega328P lower level microcontrollers was through the pre-installed I2C interface. All the components installed on the mobile robot were supplied with

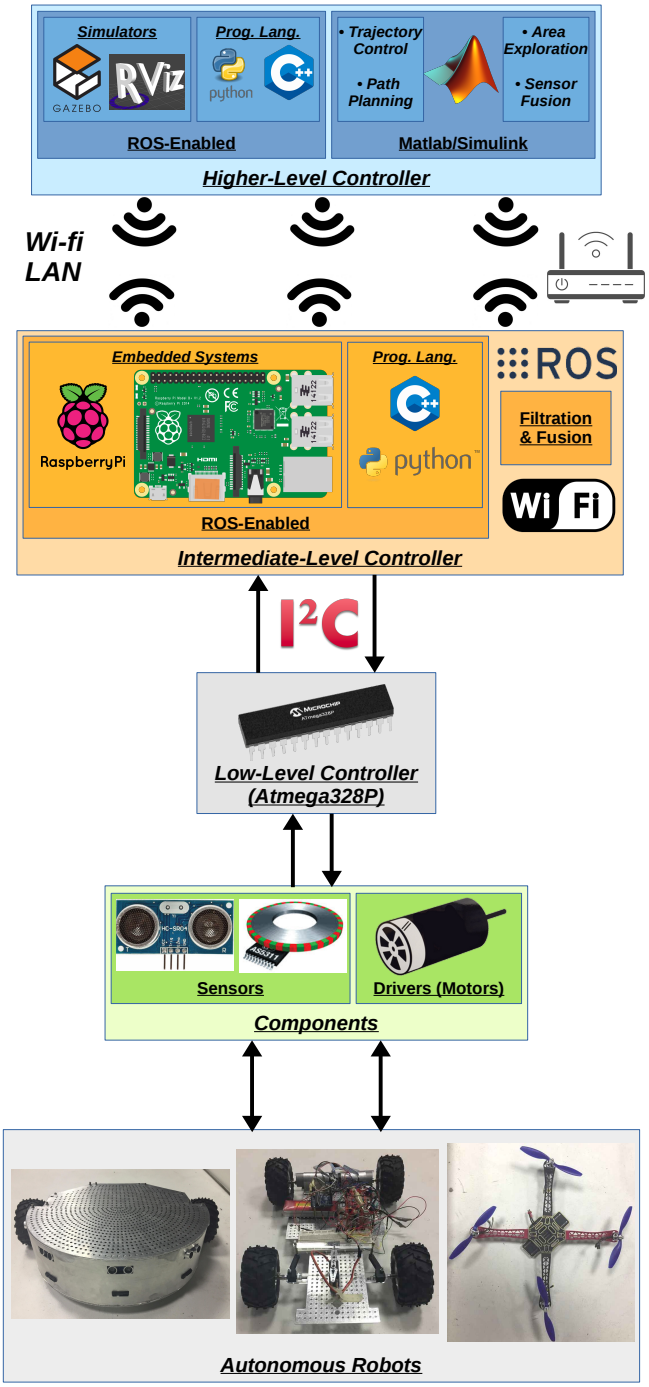

Fig. 6. Block diagram for an overview of the proposed ROS-based architecture.

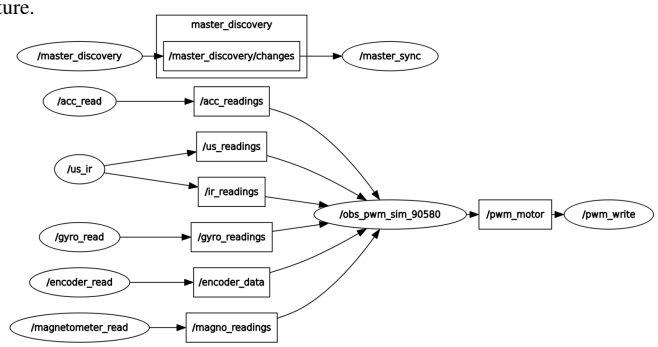

Fig. 7. rqt graph describing functional ROS nodes and topics of the experiment.

an 11.1V Lipo (Lithium Polymer) battery power supply, while utilizing voltage converters for some components in order to meet their voltage level regulations.

The aim of the experiment was to assess the stability of the whole architecture on actual hardware, while observing the speed, reliability and efficiency of data transfer between different levels of the hierarchical architecture. Another parameter that needed to be tested was the processing power of our embedded boards used which are Raspberry Pi 3 model B for the intermediate level and Atmega328P micro-controllers for lower level interfacing, on an actual application deployed on real hardware. A block diagram illustrating an overview about the sensors and actuators used for bidirectional data transfer and I2C bus connections through the whole hierarchy is shown in Fig. 8.

The implemented application that was used for evaluating the ROS-based architecture was a simple obstacle avoidance algorithm implemented on Matlab/Simulink in a confined space within a laboratory setting. An rqt graph generated from the rqt library is used for visualizing and illustrating all functional nodes and topics for the experiment as shown in Fig. 7. The sequence of operation can be described as follows: Raw data from sensors was recieved through the Atmega chips interface. A node on Raspberry Pi sends a 'read' command to the I2C bus with a specific slave addresses for data acquisition. The same node publishes that data over a ROS topic through the multimaster connection over Wi-Fi. A subscriber node on Matlab uses the data received as input to the algorithm. The output is shared on a ROS topic through a publisher node and PWM signals are sent down the same hierarchy for actuation.

The process was working with a frequency of 20 Hz as specified by ROS nodes publishers and subscribers. The same experiment was conducted for the AMR shown in Fig. 3 and the AV shown in Fig. 4 for validating the effectiveness and efficiency of the introduced architecture under several testing conditions with different hardware designs. Another experiment was done by controlling the speed of brushless DC motors for open loop control of the autonomous quadcopter shown in Fig. 5.

Another experiment was conducted using open loop control to traverse the AMR in a square trajectory with data transmission speed of 20 Hz. The main purpose of this experiment was to validate the accuracy of the architecture in exchanging more sensitive data. Fig. 9 represents angular feedback (theta) from an encoder during operation.

The results of the experiments were promising, showing that the network connection and I2C bus were stable and data was sent back and forth through the whole hierarchical architecture with minimal data lost. The experiment was running for more than ten minutes with negligible delays and errors. Observing the results of the validation experiments showed that the proposed architecture is relatively stable and has the potential for further testing with heavier applications that requires more speed and processing power. The architecture is ready to be deployed on several autonomous robots for inter-vehicular communication for multi-robotic applications.

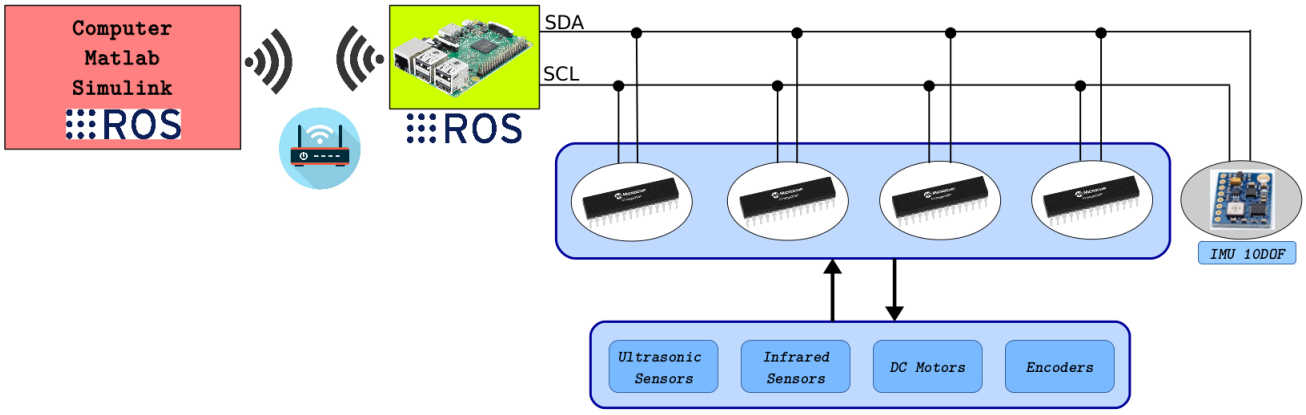

Fig. 8. Block diagram describing components used and communication between them.

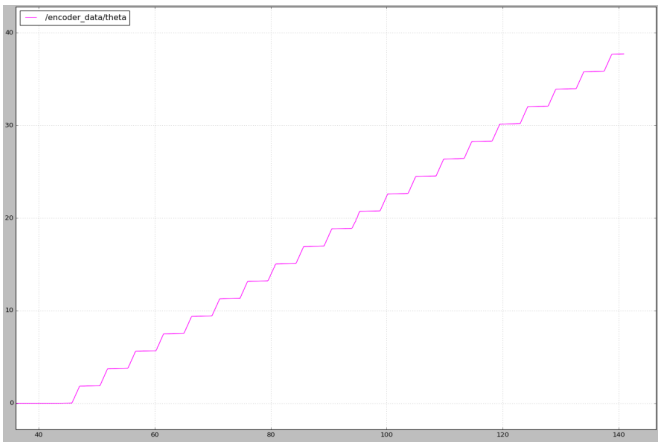

Fig. 9. Feedback from encoder generated using rqt library.

V. Conclusion

This paper presents an implementation of a generalized ROS-based architecture that is capable of inter-connecting multi-robotic heterogeneous systems for stable communication and data exchange. The proposed architecture consists of three layers arranged in a hierarchical manner. It comprises of a higher-level controller where complex computations are performed, and a lower-level controller which is used as an interface for sensors and motor drivers, as well as an intermediate-level embedded system which is considered as the main link between low-level data acquisition and high-level logic operations. The high and middle architecture layers are composed of several ROS packages with each package spanning a collection of ROS nodes each having its own purpose and functionality, whereas the lower layer involves a collection of C/C++ hardware interfacing packages. The validation of our architecture is in the form of experiments conducted on three different platforms which are; autonomous mobile robot, autonomous vehicle, and autonomous quad-copter. The results of the conducted experiments showed stability for an acceptable time interval during the experimentation process. We have concluded that the proposed architecture is reliable enough to bear more complex autonomous system applications, and provides a stable communication base for multi-robot systems.

The future work will focus on further testing of the architecture on heterogeneous multi-robot systems, applying the architecture on several autonomous robots for cooperative sensing and maneuvering, and simultaneous localization and mapping (SLAM). We will also focus on obtaining a more sophisticated architecture by increasing system reliability through including safety measures and encryption servers.

References

[1] D. K. Pratihar, *Intelligent Autonomous Systems: Foundations and Applications*. Springer, 2010, vol. 275.

[2] L. Hobert, A. Festag, I. Llatser, L. Altomare, F. Visintainer, and A. Kovacs, "Enhancements of v2x communication in support of cooperative autonomous driving," *IEEE communications magazine*, vol. 53, no. 12, pp. 64–70, 2015.

[3] "About ez10," http://7starlake.com/EZ10/EZ10.html.

[4] F. Fahimi, *Autonomous robots: modeling, path planning, and control*. Springer Science & Business Media, 2008, vol. 107.

[5] "Northrop grumman mq-8 fire scout," https://en.wikipedia.org/wiki/Northrop_Grumman_MQ-8_Fire_Scout.

[6] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *Industrial and Information Systems (ICIIS), 2012 7th IEEE International Conference on*. IEEE, 2012, pp. 1–5.

[7] D. Gomez, P. Marin-Plaza, A. Hussein, A. Escalera, and J. Armingol, "Ros-based architecture for autonomous intelligent campus automobile (icab)," *UNED Plasencia Revista de Investigacion Universitaria*, vol. 12, pp. 257–272, 2016.

[8] "Robot operating system (ros)," http://www.ros.org/.

[9] "Ros wiki: Documentation," http://wiki.ros.org/.

[10] W. P. N. dos Reis and G. S. Bastos, "Multi-robot task allocation approach using ros," in *Robotics Symposium (LARS) and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR), 2015 12th Latin American*. IEEE, 2015, pp. 163–168.

[11] A. Singhal, P. Pallav, N. Kejriwal, S. Choudhury, S. Kumar, and R. Sinha, "Managing a fleet of autonomous mobile robots (amr) using cloud robotics platform," in *Mobile Robots (ECMR), 2017 European Conference on*. IEEE, 2017, pp. 1–6.

[12] Z. Obdržálek, "Mobile agents and their use in a group of cooperating autonomous robots," in *Methods and Models in Automation and Robotics (MMAR), 2017 22nd International Conference on*. IEEE, 2017, pp. 125–130.

[13] P. Bouchier, "Embedded ros [ros topics]," *IEEE Robotics & Automation Magazine*, vol. 20, no. 2, pp. 17–19, 2013.

[14] C. Hu, C. Hu, D. He, and Q. Gu, "A new ros-based hybrid architecture for heterogeneous multi-robot systems," in *Control and Decision Conference (CCDC), 2015 27th Chinese*. IEEE, 2015, pp. 4721–4726.