Fleet Management of Distributed Multi-Robot Systems for Autonomous Data Collection

Andrej Orsula
Robotics
Aalborg University
Aalborg, Denmark
aorsul16@student.aau.dk

Eduardo Fonseca
Robotics
Aalborg University
Aalborg, Denmark
efonse19@student.aau.dk

***Abstract*—With the advent of machine learning, a need for large quantities of various data has emerged. Mobile robots provide improved effectiveness for data collection over static sensors due to their ability to autonomously navigate through large areas while continuously acquiring various data. Utilising multiple robots to collect data simultaneously can reduce the overall cost and accelerate the process even further. Despite all these benefits, the management of a large mobile robot fleet is a complex task where each robot must be monitored and eventually updated or reconfigured. Similar problems were solved in recent years by Internet of Things (IoT), which allows for a direct translation of this paradigm to distributed multi-robot systems. Advances in cloud computing and software containerisation are additional enabling factors that provide distributed applications with the flexibility they require. Lastly, the development of robotic middleware with adequate reliability and efficiency for distributed systems provides a foundation for multi-robot applications. This paper demonstrates a combination of these state-of-the-art technologies into a system capable of effective data collection. In this effort, *balena* was utilised for its fleet management capabilities, while all applications were containerised using *Docker* and developed on top of *Robot Operating System 2* (ROS 2). Finally, a proof of concept, comprising of a local server and three *TurtleBot3* robots, was implemented to verify the proposed system architecture. This experimental setup supports the hypothesis that the use of state-of-the-art frameworks can accelerate the development and provide better performance for distributed multi-robot systems with improved reliability.**

***Index Terms*—Distributed Robotics, Fleet Management, Data Collection, ROS 2, Balena, Docker**

I. Introduction

Machine learning techniques have increasingly been used to solve real-life problems in the last few years. Many of these algorithms require large training datasets to derive complex models. Moreover, training data must be representative enough to provide a generalised solution without overfitting. However, the acquisition of physical data is often expensive and time consuming, which proves especially problematic in complex environments such as densely populated areas.

Static sensors provide diminishing returns in large environments due to the increasing requirements for hardware and connectivity. This problem was addressed by [1], in which a mobile robot was employed to periodically collect data from static wireless sensor nodes. Automatic deployment of sensor nodes by the use of robots was suggested by [2]. However, these approaches do not fully utilise the potential of mobile robots and are limited to discrete vantage points.

Autonomous mobile robots with several sensors were employed in [3] and [4] for indoor and outdoor environments respectively. The major advantage of using mobile robots as data collection agents is their ability to automatically relocate to many positions while providing a continuous data flow. A strategy that balances the amount of data collected from high priority areas was proposed by [5], which further increases the efficiency of utilising mobile robots. A similar strategy was investigated by [6] for distributed multi-robot systems that are capable of accelerating the data collection by utilising multiple robots in parallel.

Distributed systems allow robots to collaborate while sharing common resources and are therefore considered to be superior to single robots in performing the same task [7]. In spite of these advantages, management of a large fleet of robots is complex. Each robot must be monitored and capable of reporting issues and incidents as soon as they occur. Eventually, software updates are required to apply security patches, add new features or reconfigure robots to a new behaviour, application or even environment. Updating each robot manually is tedious and prone to error, which makes it infeasible once a large fleet is deployed. Additional complications arise if a system comprises of robots that are heterogeneous in hardware, software or both [8].

Many of the complications introduced by multi-robot distributed systems are identical in nature to concepts present in Internet of Things (IoT), a paradigm developed independently from robotics in recent years. The concept of IoT is based around a variety of pervasive devices capable of communicating among each other through unique addressing schemes [9]. Intersections between robotics and IoT technologies are beginning to emerge, resulting in a combination termed Internet of Robotic Things (IoRT) [10] as shown in Figure 1. Cloud computing is another enabling technology for distributed robotics that provides networked access to hardware resources or services on demand. Among others, *Rapyuta*[1] [11] is an example of an open-source platform that specialises in cloud services for robotics, i.e. cloud robotics.

[1]https://rapyuta.io

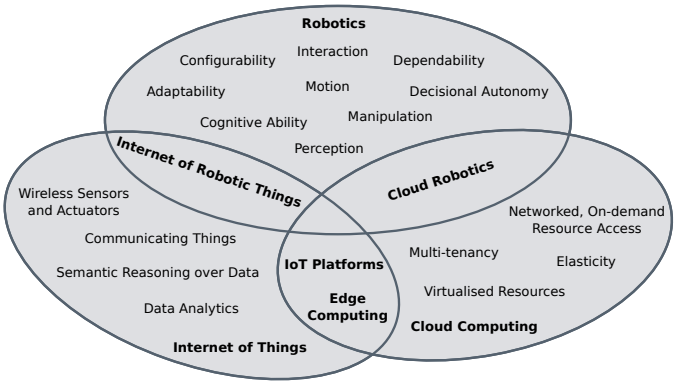

Fig. 1. The enabling technologies of multi-robot distributed systems. [10]

On the other hand, IoT platforms provide cloud-based fleet management services such as provisioning, deployment and management [8]. There is a large diversity of available IoT platforms, e.g. *Azure IoT Hub*[2], *Google Cloud IoT*[3] or the open-source *balena*[4].

One of the driving forces behind the rise of IoT platforms are the recent advances in software containerisation technology [8]. Containerisation allows the deployment and running of light-weight and standardised units of isolated software in a platform-agnostic manner [12]. Not only does this improve the portability of software onto robots utilising different architectures, but it also simplifies collaboration among developers. An additional requirement for multi-robot applications is a middleware capable of supporting distributed systems. *Robot Operating System* (ROS) has been a popular choice in the open-source robotics community for several years. However, the centralised network configuration of ROS limits its applicability in distributed systems, which is one of the factors that prompted the development of ROS 2 [13]. Although it is still under development, the potential of ROS 2 has attracted many developers in the last few months.

All of the aforementioned technologies support the development of multi-robot distributed systems. The contribution of this paper is to demonstrate a combination of open-source state-of-the-art technologies into an ensemble that provides appropriate fleet management capabilities with effective data collection.

II. Methods

Through this section, the proposed system is described. The first part provides background knowledge for the selected frameworks, namely *balena*, *Docker* and ROS 2, whereas the second part details the design of the system architecture.

A. Balena

Balena is an IoT platform designed for fleet management of connected *Linux* devices. Its core is based on utilising *Docker* containers to manage applications, while allowing multiple services within an application to be run together by the use of *Docker Compose*. All applications can be built for a specific architecture locally, or by the use of cloud-based builders provided by *balena*. To utilise *Docker* or *Docker Compose*, *balena* provides its own daemon for running the containers, which is claimed to be more light-weight and targeted towards IoT devices. This daemon is part of the host operating system *balenaOS* that is required for devices registered with *balena*. All of these services can either be hosted by *balena* using their paid *balenaCloud* services, or by self-hosting a custom open-source version called *openBalena*. Unlike *openBalena*, *balenaCloud* also handles security and contains additional features such as delta updates and web-based dashboard. [14]

B. Docker

Docker is used to containerise software and all its dependencies into a standardised unit that includes source code, runtime binaries, system tools and libraries in a platform-agnostic manner. This provides a guarantee that each container will always run in a repeatable manner, which makes collaboration simpler. Multi-container applications can be configured and started by the use of *Docker Compose* with a single command. As opposed to virtual machines, Docker virtualises separation of applications by the use of *Linux* containers, in which common resources can be shared among different applications. Containers have been shown to outperform virtual machines in both performance and scalability [12].

A *Docker* container is created using an image that is built from a series of layers, i.e. sequential instructions defining the construction process of these images. *Docker* leverages pointers to the existing image files in order to provide the ability of basing one image on top of another. Creation of new containers from images adds an extra layer that can be modified during run-time. This layer is removed once a container is deleted, while the underlying image stays the same. All containers can be given access to any of the connected hardware and network interfaces. [12]

C. Robot Operating System 2

Robot Operating System (ROS) is an open-source and widely used framework for development of robotic applications in both research and industry. ROS provides middleware that utilises a topic-based publish-subscribe messaging pattern, where topics are channels that allow processes, designated as nodes, to communicate. Despite its popularity, ROS has certain limitation that ROS 2 aims to improve. These improvements include better performance, reliability, integrating with real-time applications and more [13].

ROS 2 is built on top of *Data Distribution Service* (DDS), which is an industry-standard communication framework. It provides node discovery, message definition and serialisation along with a similar publish-subscribe transport. DDS also exposes several parameters under Quality of Service (QoS) settings, which provides further flexibility in controlling the behaviour of communication. Unlike the centralised master

[2]https://azure.microsoft.com/services/iot-hub
[3]https://cloud.google.com/solutions/iot
[4]https://balena.io

structure utilised within ROS 1, the use of DDS makes ROS 2 better suited for multi-robot applications in distributed systems. ROS 2 generally aims to hide the complexities of DDS implementation details to provide easier access to its features for robotic developers. Furthermore, ROS 2 also implements request-response services and request-feedback-response actions. [15]

D. System Architecture

The system proposed by this paper comprises of a fleet of mobile robots and a single server that provides overview and control over the fleet. The overall architecture is visualised in Figure 2. Each mobile robot is required to have a wireless connection and be equipped with a sensor that provides it with the perception of its surroundings whilst supporting autonomous navigation. The server can either be cloud-based, locally-hosted or somewhere in-between to resemble edge computing concepts. Access to *balenaCloud* is required by all robots as it is utilised for building and deployment of the applications. All services are developed as *Docker* images and grouped into *Docker Compose* applications, which makes them capable of running with either *Docker* or *balena* daemon.

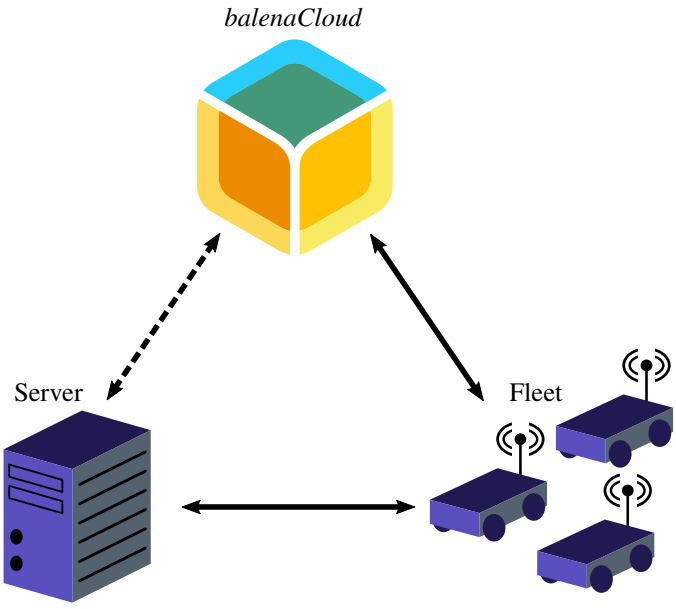

Fig. 2. An overview of the system architecture proposed by this paper.

All mobile robots are provisioned with *balenaOS* and registered under a *balenaCloud* application with unique names. Multiple applications can be utilised if the robots have conflicting computer architectures. If a fleet consists of multiple robot models that are heterogeneous in hardware, the robots can also be registered under separate *balenaCloud* applications based on their model.

The registration of a server to *balenaCloud* depends on its accessibility and should generally be performed if the utilised device is dedicated solely for this application. In case the server is cloud-based or a part of a system that has multiple assignments, installation of *balenaOS* under a virtual machine is recommended. In case the server is not registered with *balenaCloud*, the device must have *Docker* daemon with support for *Docker Compose*.

E. Base Robot Functionalities

Each robot utilises its own computer resources to run core processes that provide it with sensor readings, low- and high-level control of the actuators as well as keeping track of the relative transformations among its own coordinate frames. These core processes are customised to robot hardware in systems with heterogeneous fleet. All of these provide inter-process communication via ROS 2 topics that are accessible by other nodes for subscription and publishing, e.g. for collection of sensor readings or control of command velocities.

F. Navigation

The autonomous navigation of robots is provided by the use of *navigation2* [16] stack for ROS 2. This includes utilisation of *Adaptive Monte Carlo Localisation* [17], *Dijkstra's algorithm* [18] for global path planning and the local planning is accomplished by *Dynamic Window Approach* [19].

A centralised approach for providing navigation is utilised, where the server stores all maps and configuration parameters for all robot models in the fleet. Furthermore, all navigation processes are executed using the computational resources of the server by providing ROS 2 services for starting and stopping of the navigation stack. This centralised approach allows the server to control the motion of all robots in the fleet individually.

Once a robot is booted, it automatically calls the start service of the server with a request that contains a unique identifier of the robot, its model and the name of the map that should be loaded. Furthermore, each robot keeps track of its last known pose so that there is no need for global localisation in case a robot needs to reboot for various reasons, e.g. to receive a software update. Whenever a robot shuts down, a stop service is called to gracefully release the computational resources of the server that were required for running an instance of navigation stack for the particular robot.

G. Data Collection

While the entire fleet is autonomously navigating, the server can be utilised to monitor and record all data published on ROS 2 topics. If desired, only messages published to certain topics can be recorded. To avoid reinventing the wheel, all data is collected by the use of *rosbag2* [20], which provides a standardised storage of messages without their de-serialisation or additional serialisation. Moreover, by utilising this standardised data recording framework, redistribution of the collected data to other developers becomes simpler and further improves the community aspect of ROS 2.

III. Results

This section presents the experimental setup that was implemented[5], based on the concepts presented in the previous section, to serve as a proof of concept.

[5]Source code can be accessed on https://gitlab.com/rob768/project

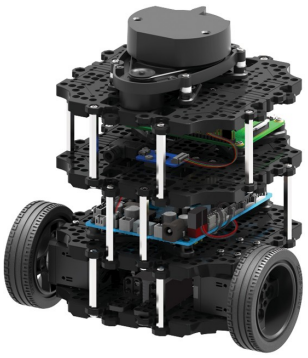

Fig. 3. The utilised *TurtleBot3 Burger* model. [21]

In this setup, three *TurtleBot3 Burger* [21], see Figure 3, were utilised to serve as a mobile robot fleet. Each robot contains a single board computer with *ARMv8* instruction set. *TurtleBot3* moves around by the use of differential drive, while providing odometry based on a sensor fusion between wheel encoders and inertial measurement unit (IMU) measurements. Each robot is also equipped with a 2D 360° laser scanner for perception of its surroundings. All *TurtleBot3* were provisioned with *balenaOS* and registered to a single application on *balenaCloud*.

For server, a generic laptop with *x86_64* architecture was utilised. This device was not provisioned with *balenaOS*, and *Docker* daemon was utilised instead to run containers that provide navigation services and collection of the data acquired by the fleet. The robots and the server were wirelessly connected to the same local area network. Moreover, the map of the environment that was utilised for the proof of concept was acquired by the use of ROS 2 *cartographer* [22] with one of the robots.

The system is initialised by running server containers for both navigation services and data collection, that were configured to record all available topics from each robot. Subsequently, all *TurtleBot3* are switched on. If any of the *TurtleBot3* had been relocated while switched off, its initial pose was explicitly assigned to its new pose within the map. After the navigation stack was initialised for each robot, navigation goals to the individual robots were manually provided.

The server was capable of successfully recording messages from all connected devices. As all tests were performed on a stable local network with relatively little traffic, no messages were lost during the course of testing. The distributed nature of ROS 2 provided better reliability of multi-robot system, whereas the use of *balena* with *Docker* accelerated the development and made it simple to update any or all of the robots with a specific configuration to repeat the tests.

IV. Discussion

The utilisation of state-of-the-art technologies for the development of multi-robot distributed systems is considered through this section. Furthermore, the shortcomings of the proposed system are analysed.

IoT platforms allow efficient management of robot fleets at scale, which introduces the notion of IoRT. These platforms significantly reduce the cost of device provisioning and deployment, while also providing the ability to apply software updates in an automated manner over the full life-cycle of a product. Moreover, the capability to easily reconfigure each robot for a new customer, application or environment opens these platforms to new use cases. Overall, IoT platforms are superior to manual fleet management techniques. However, the features provided by these platforms come at a price that is either connected with the cost of the paid services or the time-consuming manual configuration of self-hosted open-source alternatives. Fortunately, many of these platforms provide free trials that can be exploited for each scenario to evaluate their benefits.

Cloud computing provides the possibility of decreasing the requirements for computational resources of each individual robot, which in turn reduces the cost of each unit and potentially increases their battery life. A locally-hosted centralised server, such as the one proposed by this paper, shares similarities with the cloud computing paradigm, however, it resembles edge computing more closely as it brings the computational resources closer to the robots where they are needed. This in turn decreases the network latency, which might provide the required response time for certain real-time applications. On the other hand, the major drawback of utilising either of these approaches is the reduction of robot autonomy. Each robot is dependent on the connectivity with the server and would fail to navigate around as soon as it gets disconnected from the network. An ideal distributed system would allow each robot to be fully self-sufficient under any condition it might experience. Such distributed system architecture is already supported by robotic middlewares. Nonetheless, hardware requirements of individual devices still pose a limiting factor. It is believed that a trade-off between centralised and distributed approaches would provide the best results.

Containerisation of all applications is especially useful due to the benefits of portability. This has proven to be an important aspect during implementation of the proof of concept as a result of differences in computer architecture among the devices. Therefore, the creation of images for all robotic applications is recommended as it would significantly stimulate the code reuse among different applications and developers. Furthermore, the addition of containers does not prevent the use of more traditional methods, which could consequently coexist together. This presents a proposal for future development; an abstraction that integrates standardised container technology with an existing robotic middleware, as it might prove to be useful in many scenarios and further expand the community aspect of open-source robotics.

Overall, the investigated state-of-the-art technologies provide several advantages for multi-robot distributed systems. The applicability of such systems is far beyond data collection and might prove to be effective in multiple areas of research and development that benefit from sheer numbers of distributed robots. Active collaboration among the robots

in such systems is also an important aspect that should be considered, especially if these robots coexist in relatively close vicinity.

V. Conclusion and Future Work

In this paper, several open-source state-of-the-art technologies were combined into a multi-robot distributed system capable of providing fleet management functionalities, along with effective data collection. The implemented proof of concept indicates that a combination of these technologies provides numerous advantages for distributed systems, when compared to more traditional methods.

The major disadvantage of the proposed system architecture is the reduced autonomy of each individual robot, which shall be addressed in the future. Moreover, collaborative exploration with the use of simultaneous localisation and mapping shall be investigated to allow data collection in previously unmapped areas.

References

[1] T.-C. Chen, T.-S. Chen, and P.-W. Wu, "On Data Collection Using Mobile Robot in Wireless Sensor Networks," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 6, pp. 1213–1224, Nov. 2011.

[2] R. Soua, L. Saidane, and P. Minet, "Sensors deployment enhancement by a mobile robot in wireless sensor networks," in *2010 Ninth International Conference on Networks*, Apr. 2010, pp. 121–126.

[3] B. R. K. Mantha, C. C. Menassa, and V. R. Kamat, "Ambient Data Collection in Indoor Building Environments Using Mobile Robots," in *33th International Symposium on Automation and Robotics in Construction*, Jul. 2016.

[4] M. Zhou and W. Gao, "Multi-sensor Data Acquisition for an Autonomous Mobile Outdoor Robot," in *2011 Fourth International Symposium on Computational Intelligence and Design*, vol. 2, Oct. 2011, pp. 351–354.

[5] R. Wang, M. Veloso, and S. Seshan, "Active sensing data collection with autonomous mobile robots," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 2583–2588.

[6] O. Cheikhrouhou, A. Koubâa, and H. Bennaceur, "Move and improve: A distributed multi-robot coordination approach for multiple depots multiple travelling salesmen problem," in *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, May 2014, pp. 28–35.

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

[8] R. Dautov and H. Song, "Towards iot diversity via automated fleet management," in *Joint Proceedings of the Workshop on Model-Driven Engineering for the Internet of Things (MDE4IoT) & of the Workshop on Interplay of Model-Driven and Component-Based Software Engineering (ModComp) Co-located with the IEEE/ACM 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Sep. 2019, pp. 47–54.

[9] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[10] P. Simoens, M. Dragone, and A. Saffiotti, "The Internet of Robotic Things: A review of concept, added value and applications," *International Journal of Advanced Robotic Systems*, vol. 15, no. 1, Jan. 2018.

[11] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A Cloud Robotics Platform," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, Apr. 2015.

[12] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*, Mar. 2015, pp. 342–346.

[13] B. Gerkey, "Why ROS 2?" accessed 2019-11-23. [Online]. Available: https://design.ros2.org/articles/why_ros2.html

[14] A. Marinos, "Resin.io changes name to balena, releases open source edition," Oct. 2018. [Online]. Available: https://balena.io/blog/resin-io-changes-name-to-balena-releases-open-source-edition

[15] W. Woodall, "ROS on DDS," accessed 2019-11-23. [Online]. Available: https://design.ros2.org/articles/ros_on_dds.html

[16] "Navigation2," accessed 2019-11-23. [Online]. Available: https://github.com/ros-planning/navigation2

[17] D. Fox, "KLD-sampling: Adaptive particle filters," in *In Advances in Neural Information Processing Systems 14*. MIT Press, 2001, pp. 713–720.

[18] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[19] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics & Automation Magazine, IEEE*, vol. 4, pp. 23–33, Apr. 1997.

[20] K. Knese, "ROS 2.0 ROSbags," unpublished, accessed 2019-11-23. [Online]. Available: https://github.com/ros2/design/blob/ros2bags/articles/rosbags.md

[21] ROBOTIS, "PLATFORM - TurtleBot 3," accessed 2019-11-23. [Online]. Available: http://www.robotis.us/turtlebot-3

[22] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.