


On the Applicability of Docker Containers and systemd Services for Search and Rescue Applications*

Georg Novotny¹ , Simon Schwaiger¹, Lucas Muster¹, Mohamed Aburaia¹  and Wilfried Wöber^{1,2} 

Abstract—The use of robotics in search and rescue operations has grown in recent years, as these systems have the potential to greatly improve the efficiency and effectiveness of rescue efforts. However, developing and maintaining robotics systems for search and rescue applications can be challenging. These systems often operate in harsh and unpredictable environments, which demands high autonomy and intelligent behavior of the robots. Such robots are developed based on a vast amount of (open source) software solutions, which force robotic engineers to focus on systems engineering instead of application implementation. One way to address robot maintenance is to use containerization and service management technologies, such as Docker containers and systemd services, to manage the software and resources on the robotics platform. Docker containers allow applications and their dependencies to be packaged in a portable, isolated environment, while systemd provides a reliable and flexible way to manage system processes. In addition, monitoring tools can be used to track system resources and alert human supervisors to potential issues, while deployment tools can streamline the process of deploying and maintaining the robotics platform.

This paper tackles the identification and presentation of useful tools for robotic system maintenance focusing on search and rescue robots. We aim to provide a guideline for the robotic system maintenance process, and explore the applicability of Docker containers, systemd services, and other tools for search and rescue robotics applications. We discuss the benefits and potential challenges of using these technologies, and provide examples for application. Through our analysis, we aim to provide insight into the potential of containerization, service management, monitoring, and deployment technologies to enhance the capabilities of search and rescue robotics systems. We hope that our study will simplify the development and maintenance procedure for robotic system development.

I. INTRODUCTION

Mobile robotics has the potential to greatly improve the efficiency and effectiveness of search and rescue operations, allowing teams to quickly assess and respond to emergencies in hazardous or inaccessible environments [2], [6]. Since the early 2000s, various competitions and trials have been held in order to support the development of rescue robots. These events, such as the European Land Robot Trial (ELROB) [15] and the European Robotics Hackathon

(EnRicH) [14], involve the completion of various tasks in diverse environments, including 2D and 3D mapping, detection and evacuation of individuals, and manipulation of objects. These trials aim to assess the capabilities and effectiveness of rescue robots in real-world situations. Here, researchers, organizations, and companies can test and evaluate their proposed robotics systems, under disaster area-like environments [3], [10], [11]. However, due to constant improvements of software solutions including major version updates (e.g. Robot Operating System (ROS) [7], [13]), the development and maintenance of these safety-related applications recently became a challenging task. Furthermore, development teams may need to deploy robotics systems on a variety of platforms, including ground vehicles, aerial drones, and underwater drones, each with their own unique hardware and software requirements [19]. Hence, the development and maintenance task is complicated by the need to support a wide range of hardware. These aforementioned challenges make it difficult to provide software that is compatible with a broad range of applications.

To address these challenges, search and rescue teams are turning to containerization technologies, such as Docker [8], to manage the software and resources on the robotics platform [20]. Such containers allow applications and their dependencies to be packaged in a portable, isolated environment that can be easily deployed on any hardware platform. This approach can make it easier to develop and maintain software for mobile robotics search and rescue applications, as the containers can be easily moved between different hardware platforms and operating systems.

These containers can be a valuable tool for addressing the practical challenges of different hardware in safety-related mobile robotics applications. By storing applications and their dependencies in a portable, isolated environment, containers can make it easier to develop and maintain software thus being compatible with a wide range of hardware platforms. Containerization technologies have the potential to improve the efficiency and effectiveness of search and rescue operations, allowing teams to quickly respond to emergencies and may save more lives.

In this paper, we present a guideline for the development and maintenance of a container-based robotic application with a focus on search and rescue robots. We hope, that our guideline will help developers to accelerate their development procedures significantly. The remainder of this paper is structured as follows: Chapter II provides an overview of similar approaches that utilize abstraction using containerization, monitoring and managing for mobile robotics. Chapter

*This work was supported by UGV-ABC-Probe (FORTE) project.

¹Georg Novotny, Simon Schwaiger, Lucas Muster, Mohamed Aburaia, and Wilfried Wöber are with the Department of Industrial Engineering, UAS Technikum Wien, Höchststaedtplatz 6, 1200 Vienna, Austria {georg.novotny, simon.schwaiger, lucas.muster, mohamed.aburaia, wilfried.woeber}@technikum-wien.at

²Wilfried Wöber is with the University of Natural Resources and Life Sciences, Department of Integrative Biology and Biodiversity Research, Institute for Integrative Nature Conservation Research, Gregor-Mendel-Straße 33, 1180 Vienna, Austria

III reflects the implementation of the proposed approach on a mobile Search and Rescue (SAR) robot. The results are presented in Chapter IV, the paper is summarized in Chapter V and an outlook on future research is provided.

II. RELATED LITERATURE

Mobile search and rescue operations often require the use of multiple specialized applications, such as mapping, localization, sensing, perception, and exploration [18]. These applications can be challenging to manage and coordinate, especially in dynamic and time-sensitive SAR scenarios. To address this challenge, researchers have proposed the use of modular cyber-physical systems (CPS) architectures based on ROS [7], [13] and Docker. These architectures allow for the decoupling of complex CPS into simpler subsystems, enabling independent teams to work concurrently on the development of these subsystems.

One such example is [5] where containerization is used as a way to wrap up software modules in the proposed architecture for CPS. Specifically, the architecture employs Docker lightweight virtualization containers to package software modules together with all of their dependencies, allowing them to be easily deployed and run on any system. This can be particularly useful in CPS, where multiple software modules may need to be integrated and run together on a single device or system. The proposed architecture has been implemented using inexpensive hardware, such as Raspberry Pi and Arduino boards, to demonstrate its potential for use in industrial automation and mobile robotics in production systems. This approach offers benefits such as improved organization of information, modularity, ease of operation, and security through the use of lightweight virtualization and mature firewall technologies.

In [4], the author discusses the use of containerization in advanced robotics applications. Containerization involves packaging software into self-contained units that can be easily transported and deployed on different systems. This allows for the deployment of robotics systems in a flexible and scalable manner. The author describes the use of containerization in conjunction with decentralized computing approaches, such as fog and mist computing, in order to improve the speed and efficiency of these systems. The author showcases this on a manufacturing application, where a robotic arm equipped with sensors and cameras (mist computing) utilizes fog computing, via a high-performance industrial PC, to process data in real-time, while containerization allows the flexible and scalable deployment of the system. Similarly, in SAR operations, containerization can allow the rapid deployment of resources, while fog and mist computing can enable the decentralized processing of data and improve the efficiency of the system.

Another approach to managing a fleet of autonomous mobile robots (AMR) in SAR operations is the use of a cloud robotics platform [16]. A cloud-based system can provide a centralized and scalable method for managing multiple AMR, while also allowing communication with an Enterprise Resource Planning (ERP) system. A real-world experiment

with physical robots in a laboratory setting demonstrated the benefits of such a system, including the ability to handle large amounts of data, support for multiple masters, and the potential for future expansion. However, the limitations of the current cloud framework were also identified, including the need for improved security and interoperability.

The authors in [1] present a toolchain for enabling the containerization and orchestration of ROS-based robotic software applications on heterogeneous and hierarchical hardware architectures. The toolchain was demonstrated through the deployment of a real case study involving a mobile robot navigation system, which included an ORB-SLAM application [9] combined with local/global planners and obstacle avoidance. The toolchain includes a containerization step for each software component, allowing for the abstraction of the components from the target hardware and software environment. This containerization is achieved using Docker, and is extended to support CPU/GPU architectures and real-time constraints. The containerization step enables the creation of lightweight and portable containers for the software components, which can be easily deployed and executed on a variety of hardware architectures. Additionally, the containerization allows for the verification of functional and real-time constraints through hardware-in-the-loop simulation, and enables the exploration of automatic mapping across Cloud-Server-Edge architectures. The toolchain allows for the efficient and reliable deployment of the mobile robot navigation system on a variety of hardware architectures, while also ensuring that the functional and real-time constraints are met.

Overall, the use of modular containerized architectures based on ROS and Docker, as well as cloud robotics platforms, has shown potential for improving the management and coordination of multiple specialized applications, which can also be applied in mobile SAR operations. These approaches offer benefits such as improved organization and communication, modularity, ease of operation, and scalability.

III. METHODS

The selection of Docker as the containerization tool for this study was based on its simplicity, portability, and strong community support, as well as its dominant market share in the field [17]. systemd was chosen for its ease of use, versatility, and compatibility with other tools.

In this section, we will present the necessary steps to implement a systemd-managed, Docker application stack, which can be monitored by a supervisor for mobile SAR applications¹.

We will demonstrate this on the SAR robot [3], [10], [11], that should be capable of autonomously exploring and mapping its environment. Figure 1 depicts the therefore proposed software stack containing of multiple docker containers, running on different machines, and the managing and monitoring software.

¹Note that we assume that all commands are run on a UNIX system booted by systemd with a Bash shell.

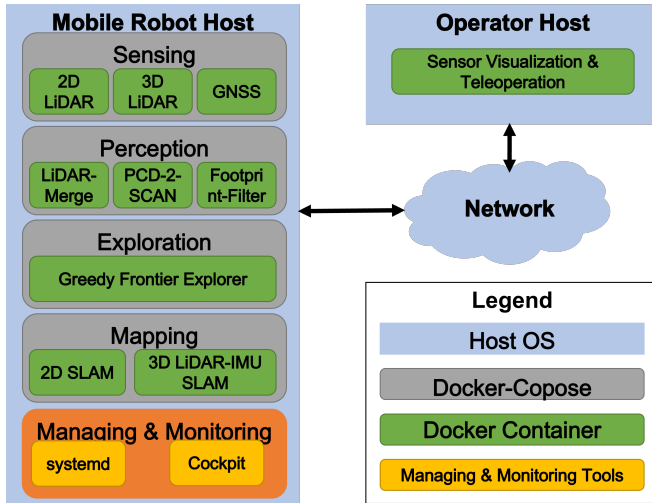


Fig. 1: Overview of proposed software stack

A. Installation

The first necessary step is to install the required programs as shown in listing 1.

Listing 1: Component installation

```
$ curl https://get.docker.com | sh &&
  sudo systemctl --now enable docker
$ apt update && apt install -y cockpit
  && sudo systemctl enable
  cockpit.socket
$ git clone
  https://github.com/mrevjd/cockpit-docker
  && bash cockpit-docker/install
```

B. Preliminary

After completing the installation of the necessary software and hardware components, we will now proceed to the fundamentals chapter, where we will delve into the basic concepts and principles that underlie the functioning of our system.

1) *Containerization*: There are many different containerization platforms available, including LXC, Kubernetes, rkt, Packer, and Docker [17].

In the context of SAR operations, Docker can be used to package the software and hardware components required for deploying mobile robots into a single container. This includes the robot's operating system, control software, and any necessary hardware drivers. By packaging all of these components into a single container, it becomes much easier to deploy and manage robots in the field, as all necessary components are included in the container and can be easily transferred between different environments.

To create a Docker container, a file referred to as a *Dockerfile* is created, which specifies the instructions for building the container. These instructions can include commands to install dependencies, copy files, and configure the container.

Listing 2: Example Dockerfile

```
FROM ros:noetic-ros-base

SHELL ["/bin/bash", "-o", "pipefail",
"-c"]
RUN apt-get update && rosdep update &&
  rm -rf /var/lib/apt/lists/*
ENV
  ROS_MASTER_URI="http://10.0.0.100:11311"
ENV ROS_IP="10.0.0.100"
ENV ROS_HOSTNAME="10.0.0.100"

CMD ["roscore"]
```

The listing 2 starts the ROSMASTER required for ROS 1 [13] applications.

Once the *Dockerfile* is created, users can use the `$ docker build` command to build the container and `$ docker run`. This creates a container image, which can be used to create and run containers.

In addition to building containers individually, Docker also provides a tool referred to as *Docker Compose*, which allows users to define and run multiple containers as a single application. Docker Compose uses a configuration file, named a *docker-compose.yml* file, to define the containers and their relationships to one another. Therefore it is easier to deploy and manage multiple containers as a single application.

Listing 3: Example docker-compose.yml

```
version: '3'
services:
  talker:
    image: osrf/ros:humble-desktop
    container_name: talker
    hostname: talker
    command: ros2 run demo_nodes_cpp
      talker
  listener:
    image: osrf/ros:humble-desktop
    container_name: listener
    hostname: listener
    command: ros2 run demo_nodes_cpp
      listener
    depends_on:
      - talker
```

The listing 3 defines two individual containers that communicate together utilizing ROS2 [7]. As shown in the example listing 3 multiple applications can be run simultaneously by *docker-compose*, each in its own container. This provides benefits such as:

- **Isolation**: By running each application in its own container, you can ensure that it is isolated from other applications and the host system. This can be useful for security and stability reasons, as it helps to prevent one application from affecting other applications or the host system.
- **Resource allocation**: Containers allow the allocation of specific resources (e.g., CPU, memory, disk space)

System Services				compose	All
Name	Description	State		Automatic Startup	
compose@	compose@.service Template				
compose@mapping	mapping service with docker compose	active (exited)		Enabled	
compose@perception	perception service with docker compose	active (exited)		Enabled	
compose@sensing	sensing service with docker compose	active (exited)		Enabled	

Fig. 2: Visualization interface displaying the systemd services and corresponding docker containers.

to each application. This can be useful for ensuring that each application has the resources it needs to run efficiently.

- **Version control:** Utilizing containers can simplify the management of different versions of an application, such as working with different Robot Operating System (ROS) distributions.
- **Maintenance:** Running each application in a separate container can make it easier to update or maintain each application independently.

2) *Monitoring and Managing:* In addition to containerization, it is important to monitor and manage the deployed robots in the field. *Cockpit* is an open-source, web-based system management tool that can be used to monitor and manage Docker containers, including those containing mobile robots for SAR operations.

systemd is a system and service manager for Linux [12] that can be used to control and manage Docker containers, including those managed by Docker Compose. It uses a declarative language called unit files to describe services, sockets, devices, and other system processes. These unit files (as seen in listing 4) are more structured than traditional init scripts, which enables systemd to automatically derive additional functionality, such as monitoring requirements, from them. Additionally, unit files are compatible with all distributions that use systemd, whereas init scripts must be adapted to specific distributions. This makes unit files a more flexible and portable option for describing and managing system processes.

Listing 4: Example systemd unit file

```
[Unit]
Description=My Docker Service

[Service]
Type=simple
ExecStart=/usr/bin/docker run --rm -d
    --name ros-noetic ros:noetic-ros-base
    roscore

[Install]
WantedBy=multi-user.target
```

systemd further supports the use of so-called *template unit files*. systemd template unit files are a powerful tool for automating the generation of multiple instances of the same unit type. By using the @ symbol in the unit file name, a template unit can be used as a base for generating multiple instances of the same unit type, each with a slightly different configuration. This can be particularly useful for generating multiple instances of a service unit, as it allows you to easily start, stop, and manage each instance individually. To use a template unit, you must specify the instance name as an argument to the *systemctl* command. Moreover, they help to manage multiple instances of the same unit type, template units can also help to improve the organization and maintenance of a system.

Listing 5: Example systemd template unit file *docker-ros@.service*

```
[Unit]
Description=My Docker Service

[Service]
Type=simple
ExecStart=/usr/bin/docker run --rm -d
    --name ros-%i ros:%i-ros-base roscore

[Install]
WantedBy=multi-user.target
```

To utilize this template unit file (listing 5), the instance name must be provided as an argument to the *systemctl* command. For instance, to initiate the container specified in the *ros:noetic* image, the command *\$ systemctl start docker-ros@noetic.service* should be executed. This will initiate the specified container using the docker run command, and the %i placeholder in the ExecStart line will be replaced with the provided instance name. This template unit file allows for the efficient creation of multiple instances of the same container type, each with a slightly varied configuration.

By leveraging systemd to control Docker Compose applications, it becomes possible to effectively manage and monitor the containers containing various applications used in search and rescue operations. Cockpit provides an interface for viewing and managing these containers, including the ability to view logs and access the console for each container.

This capability is particularly useful in search and rescue scenarios, as it allows responders to remotely monitor and control the deployed robots from a central location, without exposing themselves to a potentially hazardous environment.

C. Docker-Compose Setup

For our use-case we defined four different main applications (as seen in listing 6): (i) sensing (ii) perception (iii) exploration (iv) mapping, which should each be started by one separate *docker-compose* file. Here it is worth noting that each of these applications consists of a number of sub-applications that each is started in a separate docker container.

Listing 6: Four main applications folder structure

```
apps/
├── exploration
├── mapping
├── perception
└── sensing
```

D. systemd Template

In this work, we utilize a main systemd template unit file (see listing 7) to efficiently start and manage the various applications required for SAR robotics operations. This main template unit is configured to start, depending on the argument, multiple docker containers via *docker-compose*. Each *docker-compose* corresponds to a specific application such as mapping, localization, sensing, perception, and exploration.

Listing 7: Example systemd template unit file *docker-ros@.service*

```
[Unit]
Description=%i service via docker-compose
PartOf=docker.service
After=docker.service
After=network-online.target

[Service]
Type=oneshot
RemainAfterExit=true
WorkingDirectory=/taurob_tracker/3dparty/%i
ExecStartPre=/bin/bash check_ros.sh %i
ExecStart=/usr/bin/docker-compose -p %i
    up -d --remove-orphans
ExecStop=/usr/bin/docker-compose down

[Install]
WantedBy=multi-user.target
```

Note that the systemd template unit utilizes a shell script *check_ros.sh* (see listing 8) as *ExecStartPre* that is utilized to check the status of the ROSMASTER and restart the compose services depending on it.

Listing 8: *check_ros.sh* file

```
#!/bin/bash
/opt/ros/noetic/env.sh /bin/bash -c
"until rostopic list; do sleep 0.1;
done; sleep 5; until ! rostopic list;
do sleep 0.1; done; sudo service
compose@"$1" restart" &
PID="$!"
echo -e "\e[32mROS Check running at
$PID\e[0m"
```

By utilizing a template unit file for each application, such as mapping, localization, sensing, perception, and exploration, we are able to start multiple instances of the same application type, each with a slightly different configuration. Additionally, we utilize these template unit files to start a *docker-compose* file for each application, containing multiple sub-applications required for the respective tasks. This allows efficient deployment and management of all necessary applications from a central location, rather than individually starting and managing each application. The template unit files can also be configured to automatically restart any stopped containers, ensuring that the necessary applications are always running and available for use. This approach has proven to be particularly useful in SAR robotics, as it allows us to efficiently manage the various applications required for the operation.

IV. RESULTS

The proposed approach for containerization and managing and monitoring of applications in mobile search and rescue robotics was demonstrated using a show-case robot [3], [10], [11].

The robot was equipped with various sensors and software applications for tasks such as mapping, localization, and exploration, which were all run in separate docker containers. These containers were managed using systemd services, which allowed for easy start, stop, and management of the containers.

To further enhance the functionality and usability of the proposed approach, the containers were also monitored using cockpit. Figure 2 shows the cockpit interface, which displays all of the systemd services that started the corresponding *docker-compose* applications. By using Cockpit, it was possible to easily view the status of each container, as well as access the logs and console for each container. This was particularly useful for remotely monitoring and managing the deployed robots from a central location, without exposing them to the environment. By using docker containers and systemd services, it was possible to easily add or remove applications as needed, without affecting the overall operation of the system.

Overall, the results demonstrated the feasibility and effectiveness of using containerization and managing and monitoring applications in mobile search and rescue robotics. The proposed approach provided a simple and flexible way to manage and monitor the various software applications and

sensors on the robot, while also enabling modularity and scalability. While it is true that monitoring and managing robots remotely is possible without containers, the proposed approach provides several advantages. By using containers, each application can be isolated, and specific resources can be allocated to each application. The use of systemd as a system and service manager allows for easy start, stop, and management of the containers from a central location. Cockpit provides an intuitive interface for viewing and managing the containers, including the ability to view logs and access a console for each container. This capability is particularly useful in search and rescue scenarios, as it allows responders to remotely monitor and control the deployed robots from a central location, without exposing themselves to a potentially hazardous environment.

V. SUMMARY AND OUTLOOK

In summary, this paper presented a method for containerizing and managing applications required for mobile search and rescue operations using Docker, systemd, and Cockpit. The proposed approach allows for the easy creation and management of multiple containers, each containing a different application, such as mapping, localization, sensing, perception, and exploration. By using systemd to control the Docker Compose applications, it is possible to easily start, stop, and manage the containers from a central location. Cockpit provides an intuitive interface for viewing and managing the containers, including the ability to view logs and access a console for each container.

As a case study, we demonstrated the functionalities of the proposed approach using a mobile search and rescue robot [3], [10], [11].

In terms of future research, one potential direction would be to leverage the cloud to host the containers. This would allow for even greater scalability and ease of management, as the containers could be easily accessed and controlled from anywhere with an internet connection. Another area for future research could be the implementation of a fleet management system for a group of mobile search and rescue robots, similar to the work described in the related literature. This could allow for the coordinated operation of multiple robots, each with its own set of applications, for enhanced search and rescue capabilities.

REFERENCES

- [1] S. Aldegheri, N. Bombieri, F. Fummi, S. Girardi, R. Muradore, and N. Piccinelli, "Late breaking results: Enabling containerized computing and orchestration of ros-based robotic sw applications on cloud-server-edge architectures," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, July 2020, pp. 1–2.
- [2] S. Cebollada, L. Payá, M. Flores, A. Peidró, and O. Reinoso, "A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data," *Expert Systems with Applications*, vol. 167, p. 114195, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S095741742030926X>
- [3] S. Emsenhuber, K. Philipp, C. Pöschko, F. Voglsinger, G. Novotny, and W. Kubinger, "Robbie – a tele-operated robot with autonomous capabilities forenrich-2019 robotics trial," in *Proceedings of the ARW & OAGM Workshop 2019*. Universit"at Graz, 4 2019, pp. 117–118. [Online]. Available: https://workshops.aapr.at/wp-content/uploads/2019/05/ARW-OAGM19_22.pdf
- [4] P. Galambos, "Cloud, fog, and mist computing: Advanced robot applications," *IEEE Systems, Man, and Cybernetics Magazine*, vol. 6, no. 1, pp. 41–45, Jan 2020.
- [5] P. González-Nalda, I. Etxeberria-Agiriano, I. Calvo, and M. C. Otero, "A modular cps architecture design based on ros and docker," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 11, no. 4, pp. 949–955, Nov 2017. [Online]. Available: <https://doi.org/10.1007/s12008-016-0313-8>
- [6] M. N. Kiyani and M. U. M. Khan, "A prototype of search and rescue robot," in *2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI)*, 2016, pp. 208–213.
- [7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>
- [8] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.
- [9] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.
- [10] G. Novotny, S. Emsenhuber, P. Klammer, C. Poschko, F. Voglsinger, and W. Kubinger, "A mobile robot platform for search and rescue applications," in *Annals of DAAAM and Proceedings of the International DAAAM Symposium*, vol. 30, 2019, pp. 0945–0954. [Online]. Available: http://www.daaam.info/Downloads/Pdfs/proceedings/proceedings_2019/131.pdf
- [11] G. A. Novotny and W. Kubinger, "Design and implementation of a mobile search and rescue robot," in *Proceedings of the Joint Austrian Computer Vision and Robotics Workshop 2020*, P. M. Roth, G. Steinbauer, F. Fraundorfer, M. Brandstötter, and R. Perko, Eds. Verlag der Technischen Universität Graz, 2020, pp. 21–26.
- [12] L. Poettering, "Rethinking PID 1," Aug. 2010. [Online]. Available: <http://0pointer.de/blog/projects/systemd.html>
- [13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al., "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
- [14] F. E. Schneider and D. Wildermuth, "The european robotics hackathon (enrich)," in *International Conference on Robotics in Education (RiE)*. Springer, 2021, pp. 174–185.
- [15] F. E. Schneider, D. Wildermuth, B. Brüggemann, and T. Röhling, "European land-robot trial elrob."
- [16] 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 *2017 European Conference on Mobile Robots (ECMR)*, Sep. 2017, pp. 1–6.
- [17] Statista, "Leading containerization technologies market share worldwide in 2022," Sept. 2022. [Online]. Available: <https://www.statista.com/statistics/1256245/containerization-technologies-software-market-share/>
- [18] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005.
- [19] I. Vasilyev, A. Kashourina, M. Krashenninnikov, and E. Smirnova, "Use of mobile robots groups for rescue missions in extreme climatic conditions," *Procedia Engineering*, vol. 100, pp. 1242–1246, 2015, 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877705815005160>
- [20] C. Xia, Y. Zhang, L. Wang, S. Coleman, and Y. Liu, "Microservice-based cloud robotics system for intelligent space," *Robotics and Autonomous Systems*, vol. 110, pp. 139–150, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S092188901830040X>