UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology

Dāvis Krūmiņš

Web-based learning and software development environment for remote access of ROS robots

Bachelor's Thesis (12 ECTS)

Curriculum Science and Technology

Supervisors:
Associate professor of robotics engineering Karl Kruusamäe
Lecturer of robotics technology Veiko Vunder

Tartu 2022

Web-based learning and software development environment for remote access of ROS robots

Abstract:

Remotely accessible robots with a web browser interface are a convenient way to introduce robotics to new learners. Although different implementations exist, none of them can be easily adapted to a custom ROS (Robot Operating System) robot. This thesis aims to develop a generalised solution that can connect any ROS-based robot to the internet. A system is proposed, where VNC (Virtual Network Computing) technologies are used as the primary user interface, and Docker is used for deployment of multiple desktop environments. The resulting solution can accommodate multiple simultaneous clients, each with their own robot.

Keywords: Remote access to robots, web application, Virtual Network Computing, Docker, ROS

CERCS: T120 - Systems engineering, computer technology, T125 Automation, robotics, control engineering

Veebipõhine õppe- ja tarkvaraarenduse keskkond ROS robotite juurdepääsuks kaugteel

Lühikokkuvõte:

Veebibrauseri liidese kaudu kaugjuurdepääsetavad robotid on mugav viis robootika tutvustamiseks uutele õppijatele. Kuigi leiduvad erinevad implementatsioonid, ei saa ühtki neist lihtsasti kõigi ROS-i (Robot Operating System) robotitega ühendada. Selle lõputöö eesmärk on välja töötada üldistatud lahendus, mis suudab ühendada mistahes ROS-põhise roboti internetti. Välja on pakutud süsteem, kus peamise kasutajaliidesena kasutatakse VNC (Virtual Network Computing) tehnoloogiaid ja Dockeri abil saab kasutada mitut töölauakeskkonda. Lõplik lahendus suudab pakkuda mitut samaaegselt kasutuses olevat klienti, millest igaüks on seotud eraldi robotiga.

Võtmesõnad: Kaugjuurdepääs robotitele, veebirakendus, Virtual Network Computing, Docker, ROS

CERCS: T120 - Süsteemitehnoloogia, arvutitehnoloogia, T121 - Signaalitöötlus , T125 - Automatiseerimine, robootika, juhtimistehnika

TABLE OF CONTENTS

ABBREVIATIONS 4

INTRODUCTION 5

1 LITERATURE REVIEW AND BACKGROUND 6
- 1.1 ROS (Robot Operating System) 7
  - 1.1.1 Educational ROS robots 7
  - 1.1.2 ROS networking 8
- 1.2 Different approaches in User Interface design 9
  - 1.2.1 Custom web GUI for remote robot interaction 9
  - 1.2.2 Code submission 13
  - 1.2.3 CLI and IDE 16
  - 1.2.4 Full desktop experience using Virtual Network Computing (VNC) 17
- 1.3 Security 19
  - 1.3.1 Virtualization versus containerization 19
  - 1.3.2 Review of a good practice example 21
- 1.4 Cloud robotics 23

2 REQUIREMENTS 25
- 2.1 Functional requirements 25
- 2.2 System requirements 25

3 DESIGN 26
- 3.1 Creating virtual environments 27
- 3.2 Remote desktop in-browser display 30
- 3.3 Server architecture 32
  - 3.3.1 VNC client reverse proxies for multiple connections 32
  - 3.3.2 Remote container management 33
  - 3.3.3 Final system overview 35
  - 3.3.4 Server hardware 36

4 DISCUSSION AND FUTURE WORK 38
- 4.1 Limitations 38
- 4.2 Future work 39

REFERENCES 40

APPENDIX 42
- Source code 42
- Video demonstration 42
- Non-exclusive licence to reproduce thesis and make thesis public 42

ABBREVIATIONS

API - Application Programming Interface

CLI - Command Line Interface

CPU - Central Processing Unit

GNOME - GNU Network Object Model Environment

GUI - Graphical User Interface

GPU - Graphics Processing Unit

HTML - Hypertext Markup Language

HTTP - Hypertext Transfer Protocol

IDE - Integrated Development Environment

IP - Internet Protocol

IR - Infrared

JSON - JavaScript Object Notation

LAN - Local Area Network

MAC - Media Access Control

NIC - Network Interface Card

NPM - Node Package Manager

OS - Operating System

OSRF - Open Source Robotics Foundation

OWASP - Open Web Application Security Project

RAM - Random Access Memory

RFB - Remote Framebuffer

ROS - Robot Operating System

SDK - Software Development Kit

SLAM - Simultaneous Localization and Mapping

TCP - Transmission Control Protocol

TLS - Transport Layer Security

UI - User Interface

URL - Uniform Resource Locator

VLAN - Virtual Local Area Network

VM - Virtual Machine

VNC - Virtual Network Computing

VPN - Virtual Private Network

YAML - YAML Ain't Markup Language

INTRODUCTION

The nature of how education is distributed and received has changed significantly due to the COVID-19 pandemic, with remote learning implementations being adapted [1]. This has enabled actors in the education system to target wider audiences, with the physical location not being a limitation anymore.

Although a lot of theoretical knowledge can be obtained using e-learning, it is difficult to acquire the more significant practical experiences, especially in the more technical sciences [2]. Regarding robotics, it has been reported that students feel more present, self-aware and expressive when working with an actual physical entity instead of using other types of distance learning tools [3].

Robotics in its nature is a multidisciplinary field, where electrical and mechanical engineering is combined with computer science. While it is harder to incorporate the practical aspects of the former into a remote classroom, off-site programming of robots is certainly possible. Aspiring roboticists usually do not have a personal robot they can experiment with due to the high costs associated with purchasing one, so it is important to democratise the learning process. Remotely accessible robots would do just that by allowing anyone with internet access to study how computer code gets translated into real-world actuation and sensing.

The goal of this thesis is to develop a prototype for a web application that would allow learners to access and control a robot remotely over the internet. The user interface should be easily understandable, but not too limiting in its offered features, i.e. it should leave room for exploration. A virtual classroom with such a possibility would provide an engaging hands-on experience that is usually missing in an e-learning environment.

1 LITERATURE REVIEW AND BACKGROUND

Robotics standards are important in consolidating the heterogeneous field of robotics programming. With a common standard, developers can stop reinventing the wheel, re-use code more often, and have a common benchmarking reference [4]. One such standard is the open-source ROS (Robot Operating System) framework that is known for its extensibility, and which is becoming increasingly popular in the robotics community [5]. One of the reasons for the widespread adoption of ROS is the comprehensive learning tutorials written for it [6]. Moreover, due to the fact that good programming practices need to be observed in order to work with ROS successfully, there are good grounds for its adoption in education [7]. However, learning ROS can get easily tangled as the initial software setup is relatively complicated and, thus, intimidating to potential learners, especially when starting from a non-technical background. A web-browser environment that mimics the look and feel of an actual ROS development system while requiring minimal setup from the learner could potentially offer the much needed soft landing to learning software development for robots. Furthermore, such a learning and development environment can be used to provide access to physical robots, thus enhancing the learning experience beyond the limitations of robotic simulations.

Thus, this chapter reviews solutions that offer a way to connect a ROS robot to the wider network, and analyses the different paths that can be taken in implementing a web application that would allow it. The existing approaches are mostly distinguished by who the corresponding target audience is, meaning that the features available vary in their assumptions about the user's knowledge base.

Section 1.1 gives a brief overview of the essential ROS principles and terms that are used throughout this work. Section 1.2 analyses different remote ROS learning and development solutions that have already been implemented. Section 1.3 explores some of the security measures that should be taken when exposing robots to the internet, while section 1.4 examines how cloud computing services could potentially be employed in building the web application.

1.1 ROS (Robot Operating System)

One of the reasons why ROS is so lauded is that it offers ready-made tools for commonly encountered problems in robotics, and since ROS can be easily integrated within an existing software framework it is often the developer's top choice. ROS provides features such as hardware abstraction, low-level device control and the primary goal of it is to facilitate code reuse in the field of robotics [8]. The modular ROS infrastructure allows disparate processes called nodes (potentially not even sharing a common programming language) to communicate using a common message format. This means that even programs that were not initially meant to be used within the ROS context can be adopted to it.

ROS nodes communicate with each other anonymously via named buses called topics, which internally use unidirectional channels (Figure 1). A node publishing to a topic means that it acts as the "talker" and is sending information to it using a predefined message format. If a node subscribes to a topic it acts as the "listener" and reads the messages that other nodes have published. The overseer that handles all the communication between the nodes in a ROS application is called the ROS master, it enables nodes to locate each other and communicate peer-to-peer [9]. It also holds the global program parameters in an accessible API (Application Programming Interface).

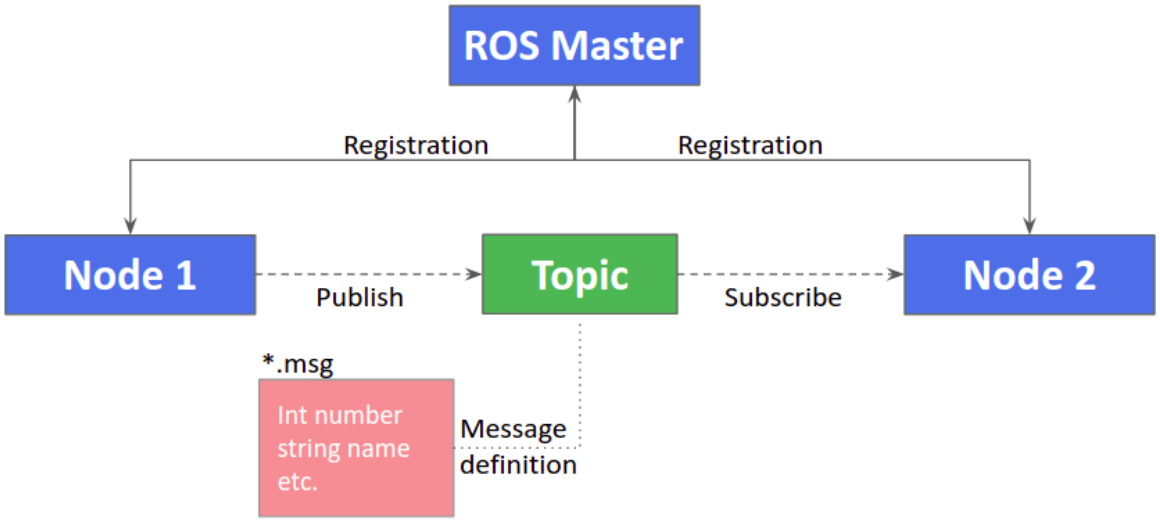
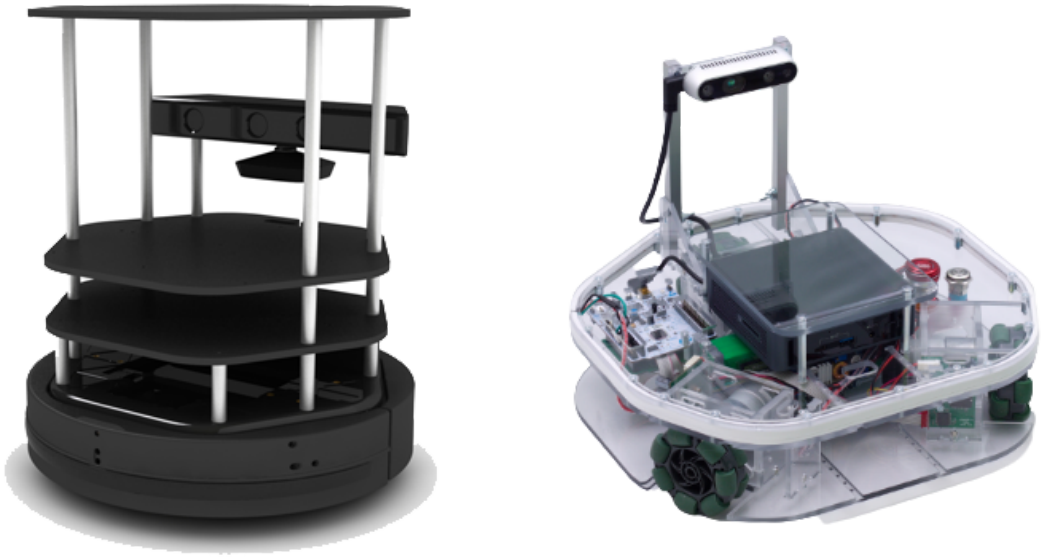


Figure 1. *The basic structure of how ROS handles communication between nodes* [10]

1.1.1 Educational ROS robots

For the purpose of learning robotics and ROS, different entry-level robots have been developed. A popular choice is the low-cost TurtleBot (Figure 2a) [11], a mobile robotics

platform with open-source software that facilitates learning of concepts such as robot kinematics, obstacle avoidance and navigation. A robot with similar capabilities is Robotont (Figure 2b) - an omni-directional mobile robot created for robotics research and teaching by the IMS (Intelligent Materials and Systems) lab at the University of Tartu [12].



(a) (b)

Figure 2. *(a) TurtleBot2, (b) Robotont*

1.1.2 ROS networking

The communication between the ROS nodes happens over the standard TCP/IP protocol, which means that it is possible to connect to multiple robots and control them all using a single computer that is within the same network [13]. For example, the Robotont is usually configured to host the ROS master and the user points to its IP address or hostname on their own computer, which then grants them access to the nodes running on the Robotont [14]. By default ROS master will communicate with anyone on the same local network, which makes it comfortable to use, but needs to be taken into account when the goal is to open up the said network to the wider area, while provisioning only a single robot [15].

1.2 Different approaches in User Interface design

In order to offer remote robot access to learners, there are multiple potential solutions that differ depending on the complexity of the setup and ease-of-use. The web applications that have been built can be broadly categorized into the following:

- Simple Graphical User Interfaces (GUI) that limit the user to specific pre-built functionality.
- Platforms that allow the user to submit custom code.
- Environments that try to simulate on-site working conditions with more advanced IDE (Integrated Development Environment) and CLI (Command Line Interface) integrations.
- Full access to the remote machine's desktop environment.

Some of the qualities to be examined in these existing applications are ease-of-use, potential learning outcomes, and vulnerability to malicious or error-prone users.

1.2.1 Custom web GUI for remote robot interaction

A website connected to ROS infrastructure can provide great remote robot control or monitoring capabilities. Plenty of options exist for this specific purpose - Robot Web Tools gives an overview of the open-source packages available [16]. Employing them makes it possible to create custom user interfaces and pick which ROS nodes and topics the user will be able to interact with. Platforms utilising these libraries usually aim to accommodate users who have no prior experience in robotics or programming; learners are provided with either a custom Visual Programming Language interface or a user-friendly GUI that appropriately guides them through the learning process.

The two building blocks for ROS web integration are *rosbridge* [17] and *roslibjs* [18]. Rosbridge provides a JSON interface to ROS and gives the client a way to subscribe or publish to topics and call services in an indirect manner. Roslibjs is a Javascript API that abstracts the interaction with the rosbridge and provides a simple path to ROS functionality. Specific examples of their usage in GUI-based ROS web applications will be examined further.

Karaca & Yayan [19] created an algorithm-focused learning interface (Figure 3) for ROS with the Blockly library [20]. Some of the programs that the users of this application can interact with on a simulated robot include teleoperation, SLAM (Simultaneous Localization and

Mapping) and wandering. Concepts such as publishing and subscribing can be more easily imparted to the learner, because the logical expressions are already present and no distracting syntax errors are encountered.

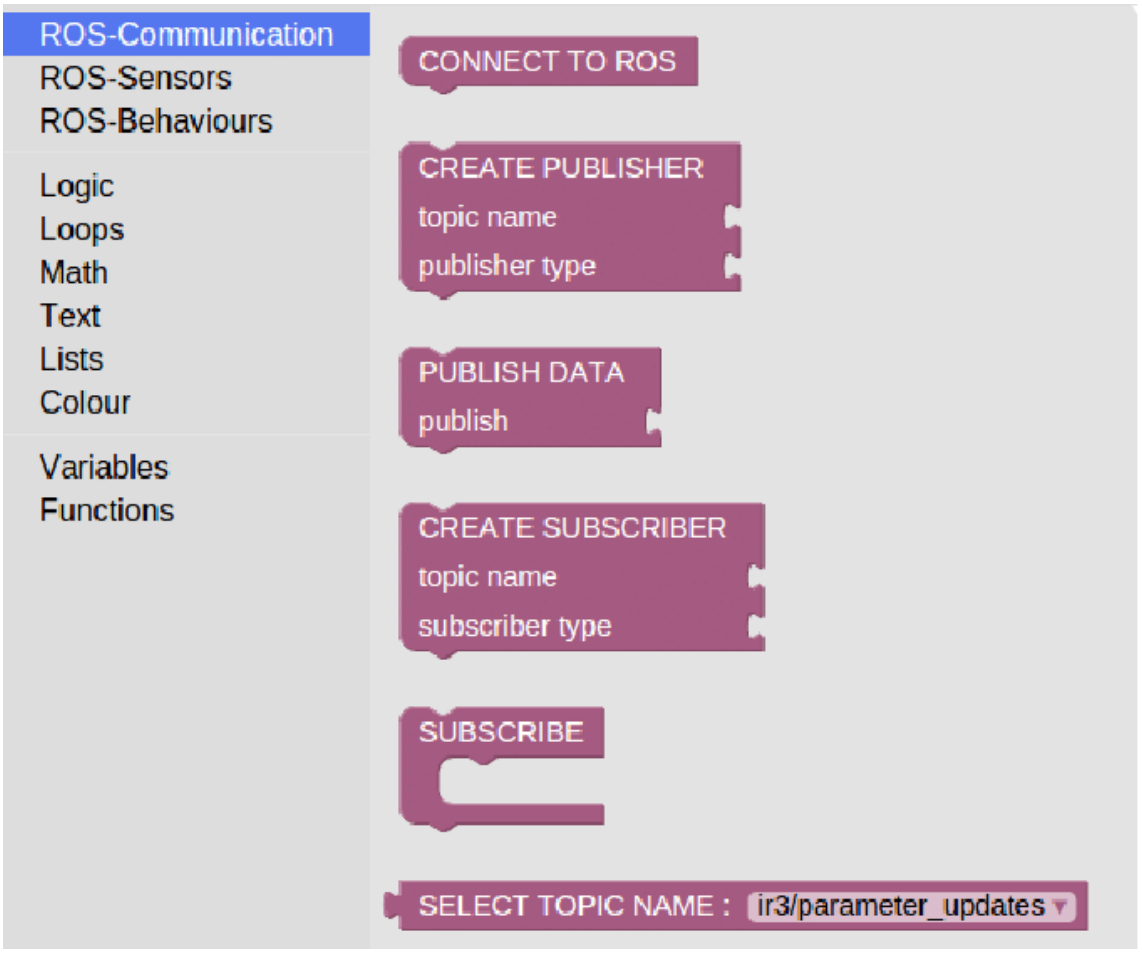


Figure 3. *Visual programming language (VPL) interface [19]*

Another example comes from Rajapaksha *et al.* [21], where through a user-friendly GUI students can run programs like mapping and object identification, with written explanations providing information on what is occurring on the screen (Figure 4). The view of the simulation is provided by *gzweb* - a graphics rendering client for the ROS simulation environment Gazebo [22].

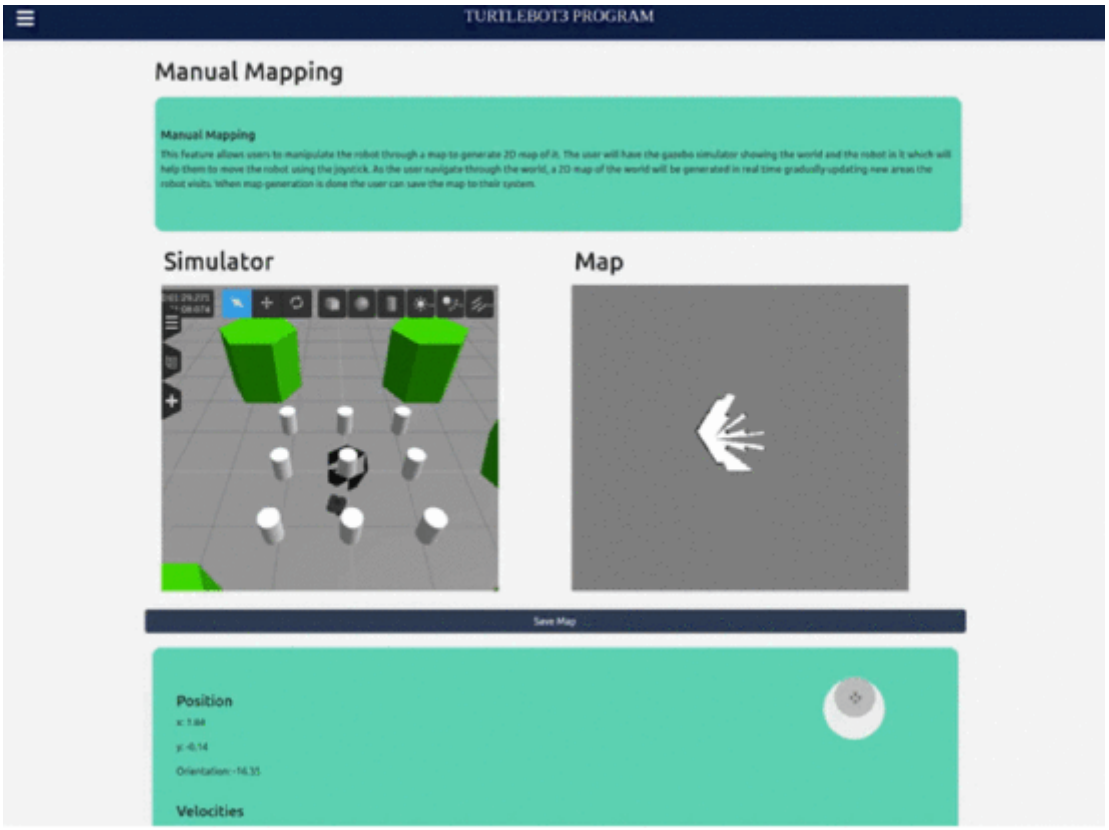


Figure 4. *Learning ROS by interactive demonstrations [21]*

Pitzer *et al.* developed a remote lab for a physical PR2 robot that provides control capabilities for teleoperation and pick-and-place object manipulation programs [23]. 3D visualisation of robot pose combined with camera streams from different perspectives allow users to see their commands being executed in real time (Figure 5). Additionally, there is a scripting interface in the form of a text input field, where more advanced users can write custom Javascript code utilising the *roslibjs* API, which gives them access to the active ROS topics and services exposed by the *rosbridge* server. There is also a way to test custom code by linking an *svn* repository, but very little details are provided on how it is achieved.

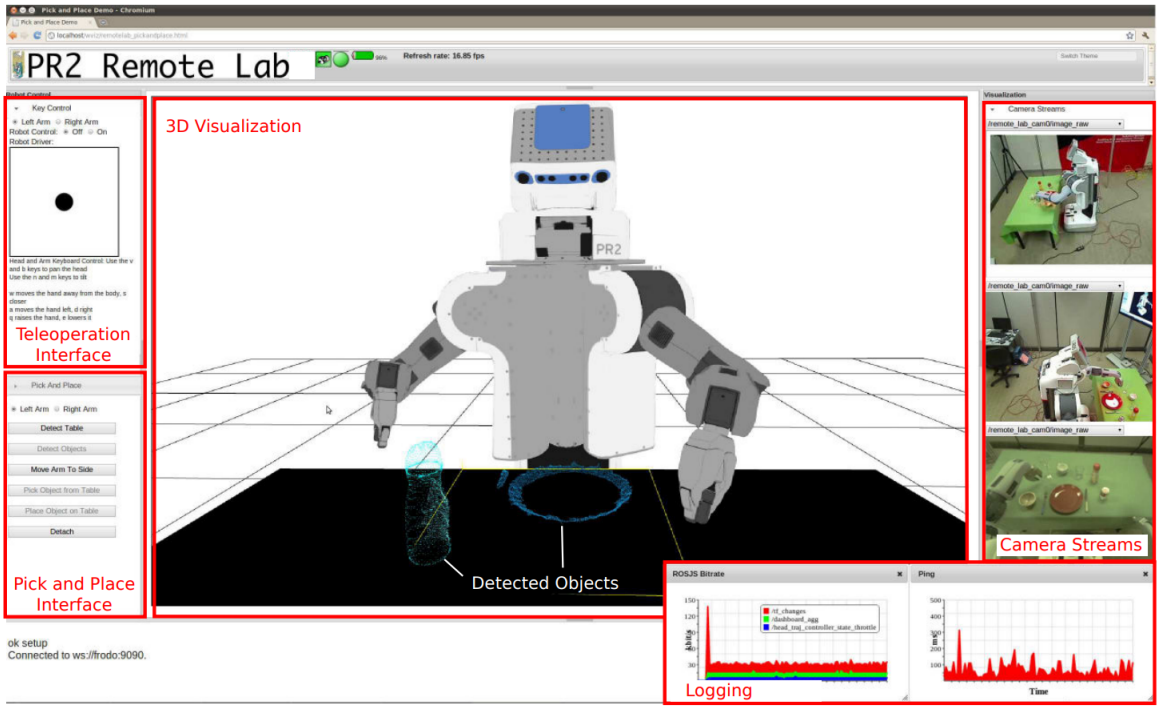


Figure 5. *PR2 robot remote lab web interface [23]*

Custom code input fields are provided by Lee for code written in the *Processing* visual programming language [24]. The input is converted into Javascript code and has access to the remote robot via *roslibjs* and *rosbridge.* With simple helper functions such as *connect()*, *publish()* and *subscribe()* that simplify the syntax the user can indirectly interact with ROS (Figures 6 and 7).

```
void move(x,z) {
    publish('/cmd_vel','geometry_msgs/Twist','{"linear
        ":{"x":' + x + ',"y":0,"z":0}, "angular":{"x
        ":0,"y":0,"z":' + z + '}}');
}
```

Figure 6. *Example of a user defined Processing VPL function [24]*

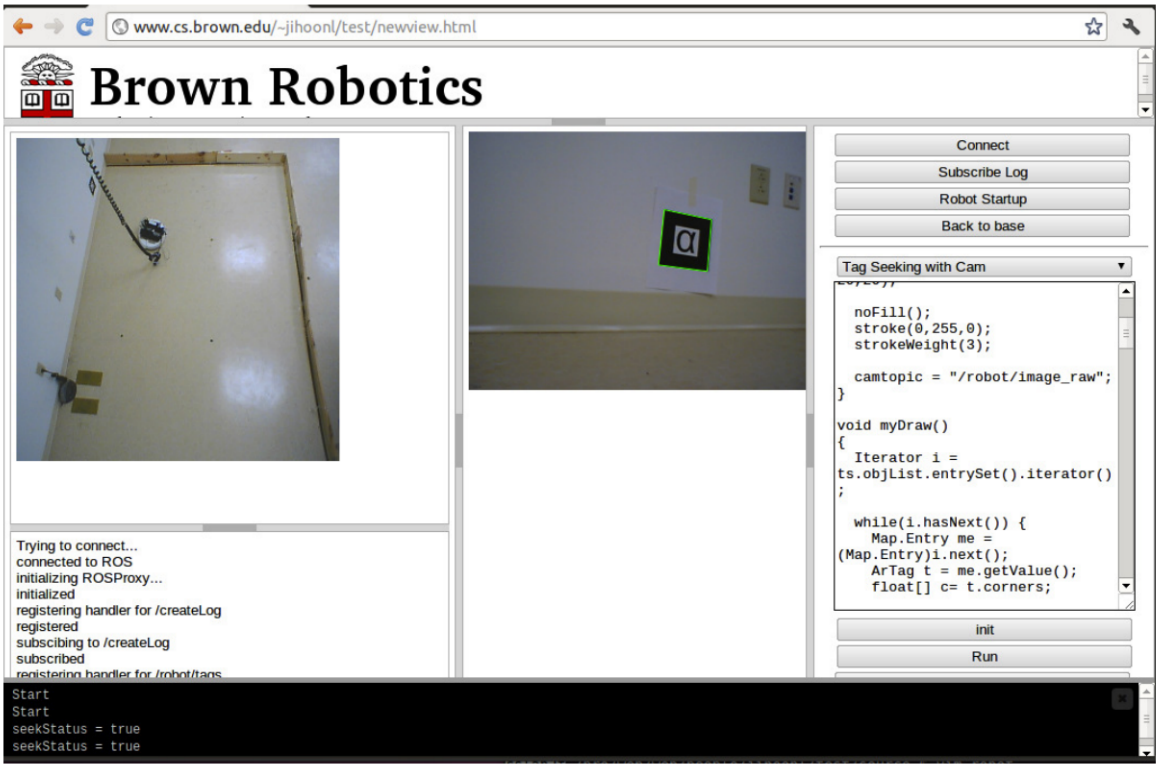


Figure 7. *The GUI of a web application for learning ROS through Processing VPL [24]*

All of the aforementioned GUI-based learning environments are easy to use for beginners being introduced to robotics and the usage of ROS. The last two examples are also advantageous in that it is easy to ensure that no potentially-dangerous custom code is being run on the remote endpoint, since the functionality is limited to select ROS topics and services that the administrator has chosen to open to the web via *rosbridge*.

1.2.2 Code submission

An option to test custom code written in a programming language that is regularly employed in robotics would give students a chance to develop more elaborate and applicable skills. Four examples [25-28] of platforms providing such remote programming capabilities through code submission will be examined in this section, but it has to be noted that only the last one [28] is ROS-based. The usual workflow when using one of these applications involves reserving a time slot during which testing and debugging of previously written code can take place in a continuous manner (feedback is provided in the form of program/sensor data logs and remote camera streams). It also has to be noted that allowing the execution of arbitrary code on the server can create cybersecurity risks and the first three examples of this section [25-27] do not address this issue.

Almeida *et al.* sets up to accept Python programs for the Lego NXT robot [25]. The text code is sent to the server (Figure 8), where before its execution only rudimentary syntax checks are completed.

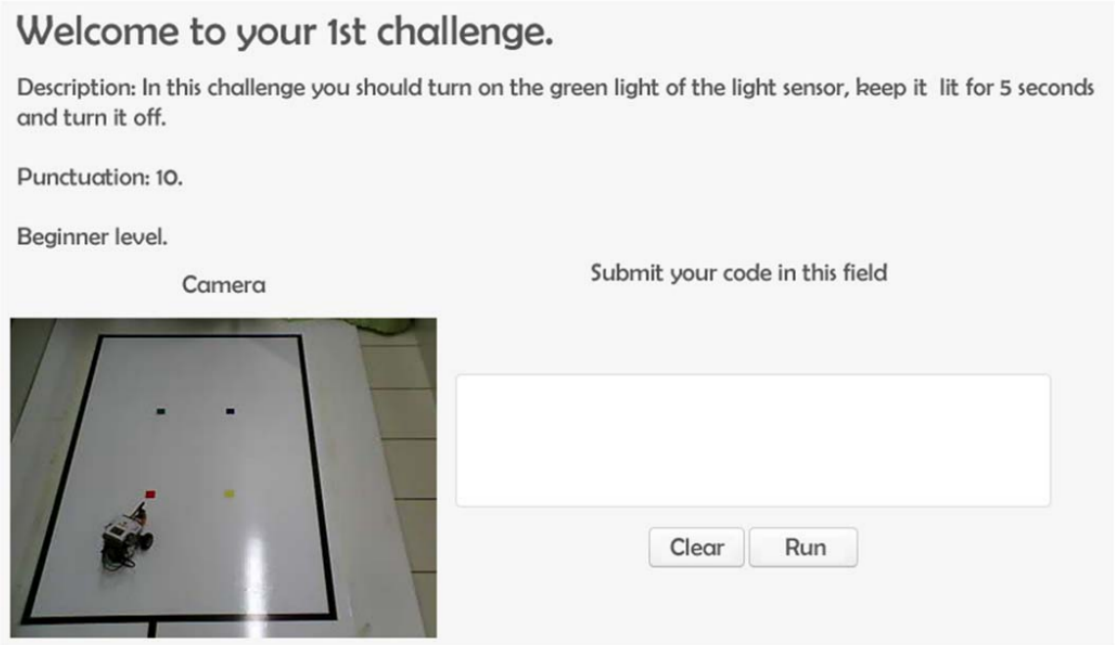


Figure 8. *Raw text code submission [25]*

In Grandi *et al.* a scalable framework incorporating session time booking is presented [26], it also includes a Java based stand-alone simulation environment that can display a robot arena in simple graphics (using tracking markers) if the user does not have enough bandwidth to utilise the video stream (Figure 9). The scalability stems from the fact that the communication with robots occurs directly with the robot's firmware through byte code, where some bits are reserved for addressing multiple board extensions.

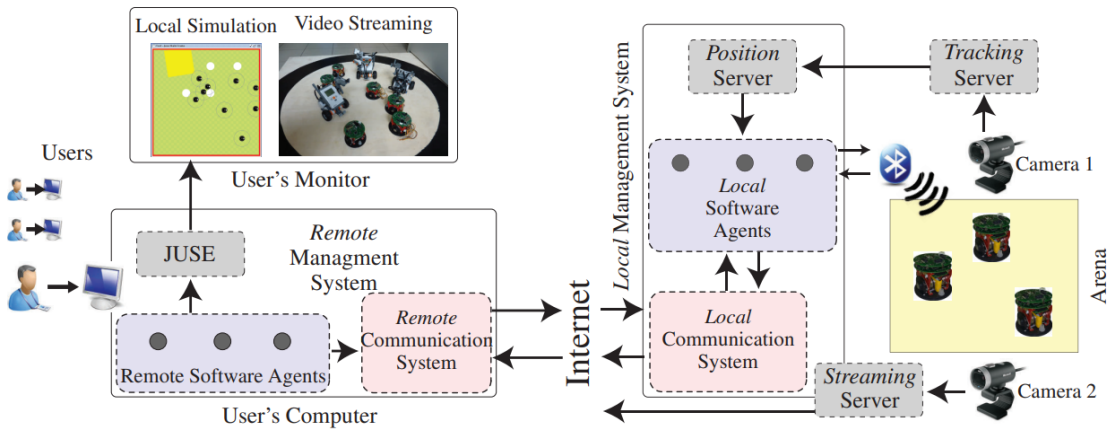


Figure 9. *The architecture of Grandi et al. framework [26]*

The submitted control code is transformed into bytes by the Local Software Agents and sent to the robots over Bluetooth to the robot's mainboard, where the firmware is configured to respond to a set of amount of instructions (such as reading IR sensors and setting motor speeds), which means that the user's freedom is "locked" to those actions. How the submitted code is processed is not described by Grandi *et al.*

Pickem *et al.* introduces a multi-robot, research-focused remote testbed in which the physical safety of the robots in terms of collision avoidance is described as the top priority [27]. The server employs a custom Monte Carlo algorithm to assess the frequency and severity of collisions that can possibly occur between the swarm robots when the user submitted code gets executed. The users can also access the same Matlab and Python simulation software for testing purposes in an offline environment by way of local installation. Although not described in detail, an option to add virtual robots that interface with the physical robots is also available (Figure 10). Cybersecurity concerns were not addressed in this work, relying on trusting the authorised personnel using this system.

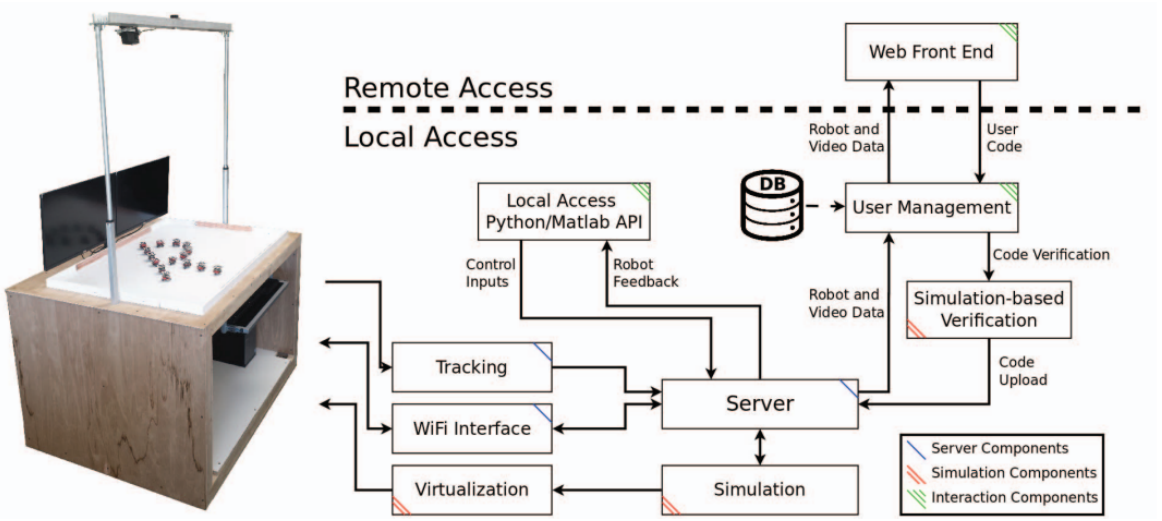


Figure 10. *The architecture of Pickem et al. framework [27]*

Casañ *et al.* comes forth with a web-enabled ROS system called RPN (Robot Programming Network), which provides a reliable way to execute a program created by a remote user directly on the server [28]. The users of RPN can write their Python code in a syntax-highlighting text field (Figure 11). Upon pressing "run" the written code is sent to the server as a string, where it is put into a file inside a sandbox ROS package. The file is then

executed inside a VM (Virtual Machine) using *rosrun*, which strengthens the security of the server's computer. The feedback the user receives about their submission comes in the form program's log messages and optional *rosbag* recordings. Any error messages encountered are displayed back to the user, and if the script hangs an option to stop the execution with a button click is also available.

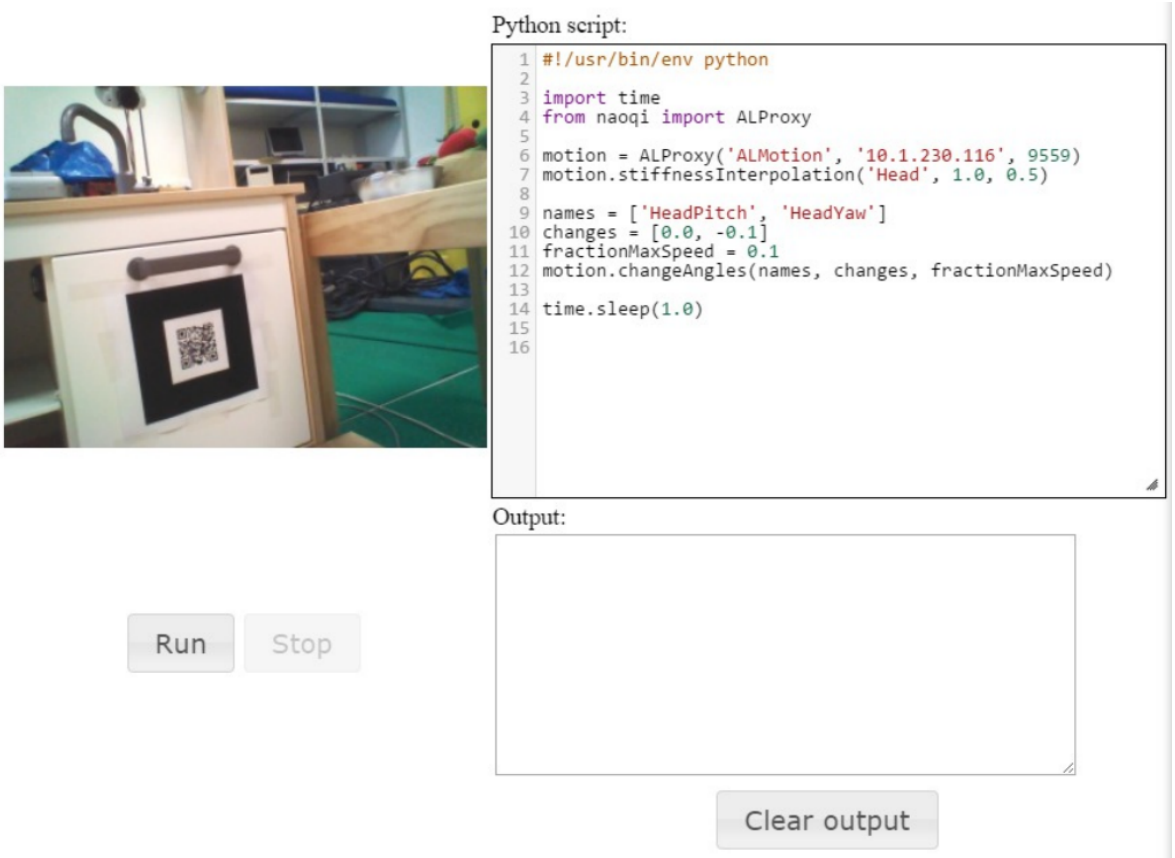


Figure 11. *Code submission interface [28]*

1.2.3 CLI and IDE

Some projects that enable remote robotics programming have chosen to incorporate modules for direct interaction with the host machine on the premises, e.g. a terminal or a code editor. This gives the learners a more realistic development environment for solving different tasks, but requires more elaborate design and can consume a lot of time before the minimal viable product (MVP) can be evaluated.

Kulich *et al.* offers students a NetBeans IDE extended with plug-ins for remote development, it includes integrated visualisations of the remote robot and custom project templates [29]. A simulation environment and a Unix CLI are also available. However, the process of how the user written programs are executed is not divulged by the authors.

A commercial example of a web application for remote robotics programming is The Construct Sim [30], which gives the possibility to use either simulation environments, enabled by WebGL [31], or access and program a physical robot in real time. The Construct gives users a custom UI (Figure 12) that has plenty of features to satisfy a learning developer's needs (terminal, IDE, feed of the simulation or real robot, educational materials).

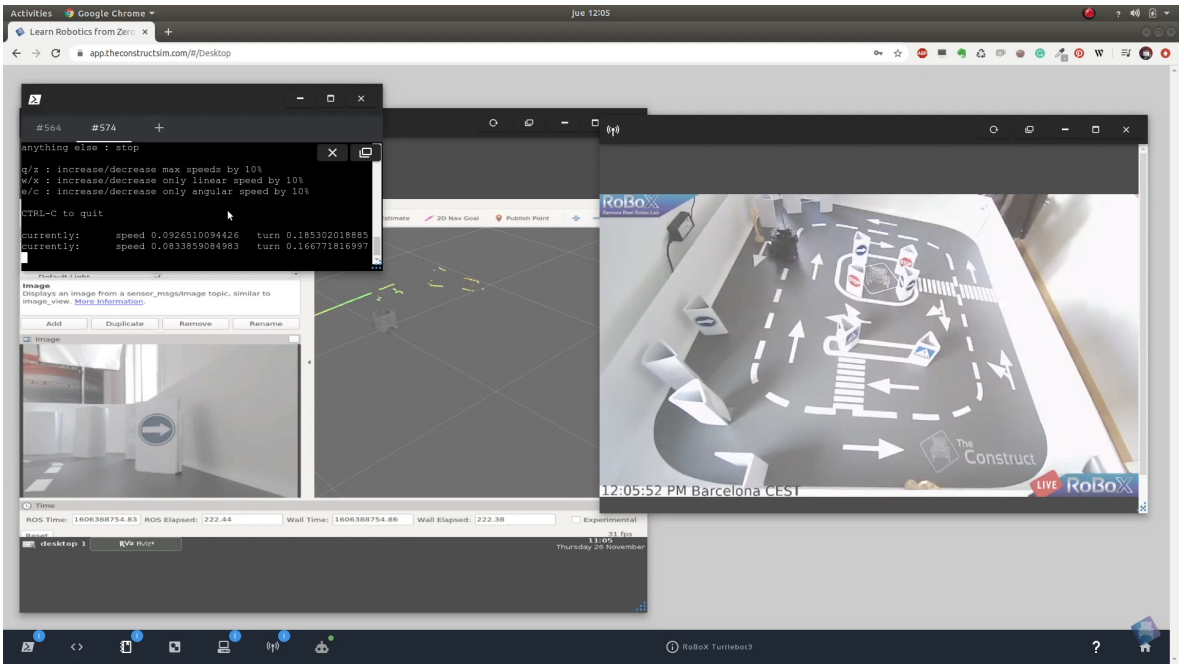


Figure 12. *The Construct Sim widget-based web interface* [32]

1.2.4 Full desktop experience using Virtual Network Computing (VNC)

The examples described in sections 1.2.1 and 1.2.2 encapsulate some specific learning outcome, whether it be the programming language used or robotics programming paradigms learned. Scaling these systems can be done, but might require some notable overhauls or additions.

In the case of ROS, which is used in conjunction with GNU/Linux, students should also acquire proficiency in working with the command-line shell. Moreover, to better use the acquired skills in an offline setting, the student should also become familiar with the specifics of the operating system of the computer in use (file-management, processes, device management etc.). The examples in section 1.2.3 do provide some of these features, but the user is still required to learn the ins and outs of those specific graphical interfaces. A simulation of the environment a student would use when working with a robot in an offline

environment would make the learned skills substantially more useful. This can be achieved by integrating the remote desktop view into a web-browser via VNC technologies.

OpenUAV developed by Anand *et al.* is an example of a remote robotics programming platform that uses the VNC approach [33]. The whole system is simulation-based and cloud infrastructure is used to achieve the required computational power for it. Figure 13 shows an example of a Lubuntu desktop served by this platform - an environment where programs like Gazebo can be accessed directly. Furthermore, photorealistic rendering using Unity is also described in this work, which means it is especially critical that users are not required to carry the computational burden - a valuable benefit of the web application.

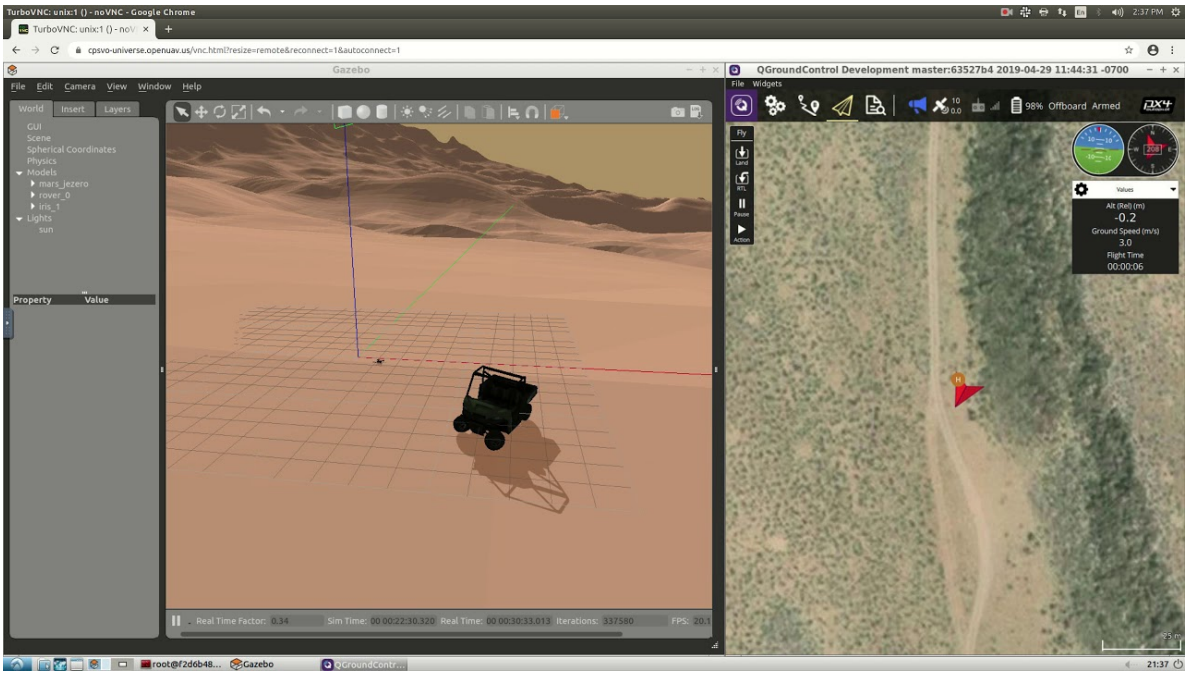


Figure 13. *Simulation of an autonomous probe deployment mission using OpenUAV* [34]

1.3 Security

In remote programming, giving users the capabilities to execute arbitrary commands on the server can result in misuse (e.g. corruption of the OS filesystem), so certain measures need to be taken to repel malicious actors. Isolating the session system in a virtual environment can protect the main server. This section examines the possibilities of virtualization and looks at how it has been applied in a remote lab context.

1.3.1 Virtualization versus containerization

Traditionally virtual machines (VMs) have been used in server consolidation, hosting, software development and testing, more importantly they are secure and provide excellent sandbox environments for students [35]. In order to use a virtual machine the host machine is required to have a hypervisor installed - software or firmware that will share and manage the host's hardware for the VMs.

However, in recent years lightweight operating-system-level virtualization using the open-source container technology Docker has been growing in popularity [36]. Containers allow developers to create isolated applications without worrying about compatibility, because the whole OS software can be reliably defined. Moreover, the resources are allocated directly by the host operating system's kernel which makes them more resource effective [37]. In Docker the base application and all its dependencies are stored in an image, which is a read-only template that serves as the starting point for a container. Docker makes it possible to run many instances of an image without quickly running into hardware limitations, which is crucial when multiple clients connect to the same server simultaneously and require an allocation of their own container environments. Figure 14 provides an overview of the differences between VMs and containers.

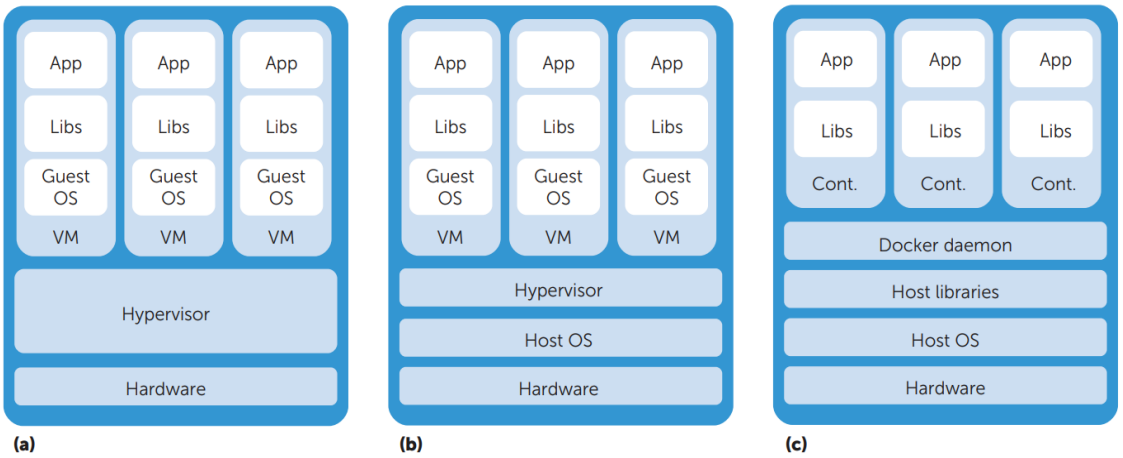


Figure 14. Differences between VMs and containers
(a) type 1 hypervisor, (b) type 2 hypervisor, (c) a container [37]

Using ROS with Docker is not overly complicated since OSRF (Open Source Robotics Foundation) provides images with ROS and Gazebo already installed [38]. These pre-built images can potentially let new learners forgo the tedious installation process and environment setup, and keep them engaged on the actual tasks. Furthermore, collaborative debugging is quite easy since the whole setup can be neatly packaged in something called a Dockerfile - a text file that contains all the commands for building an image [39].

A shared kernel accelerates development, increases performance, eases interaction with the host's file system and makes it easy to rapidly start and stop containers, however this tighter integration makes Docker containers more insecure than VMs by introducing vulnerabilities related to kernel exploits, resource starvation and shared namespace access [40]. Figure 15 shows an overview of these published by OWASP (Open Web Application Security Project).

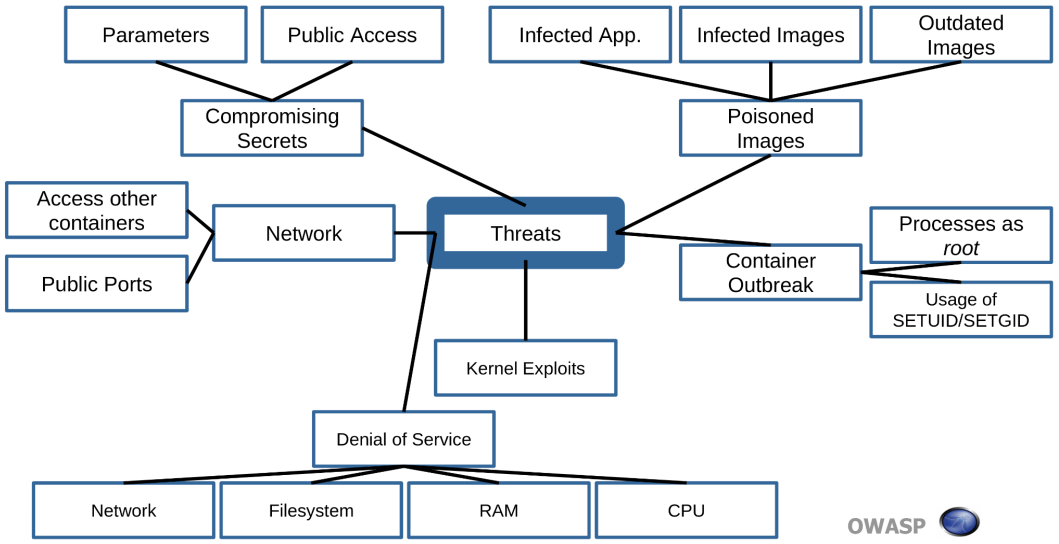


Figure 15. *OWASP overview of threats in Docker* [41]

1.3.2 Review of a good practice example

An excellent example of a secure remote development lab with Kuka LBR iiwa 7 R800 robots was found in Wiedmeyer *et al.*, where it was mostly achieved using Docker [42]. The general architecture of the project is a fully automated code submission web application, whose high throughput allows room for a large user base. Multiple robot work cells enable parallel execution of submitted jobs.

Users submit their code to the database corresponding to a specific project that is defined by the on-site setup of the specific robot and the software packages that can be used in its context (Figure 16). The users can view the status of the job, and once it successfully terminates log files and video recordings are available.

Code validation

Firstly, the code is run in a Gazebo simulation (the models are also made available to the users for testing on their personal computers), this step prevents malfunctioning code from running on the physical robot and allows the user to correct their mistakes without wasting on-site time. Secondly, the trajectories computed by the user's code are checked for collisions using the motion planning framework 'MoveIt!'. If at any point the robot-specific system detects an unfixable internal error, it terminates itself and awaits an operator for a safety check.

Network isolation

To create a secure environment for executing the compiled user's source code a two container system is defined - a client container and a proxy container. The client container runs the code, while the proxy container is responsible for isolating the host and lab's network from said code (Figure 16).

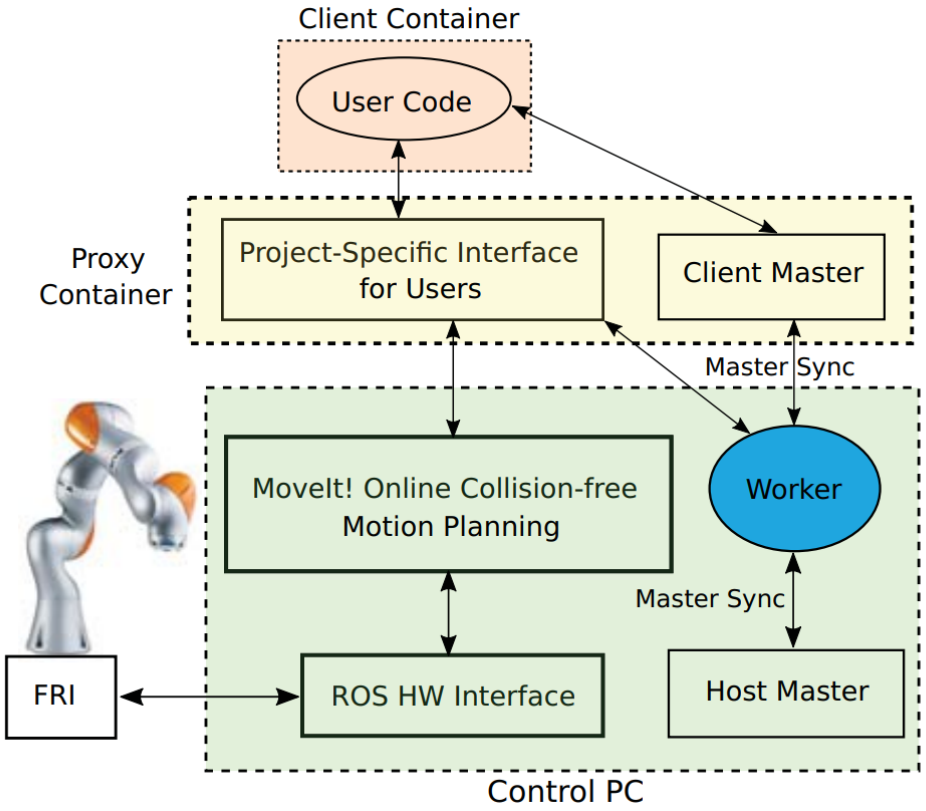


Figure 16. *KUKA Robot Learning Lab with user code isolation* [42]

The client and the proxy containers reside on the same Docker network (Figure 16). The proxy container, however, has a ROS master running on it and can also access the lab's network; it has a project-specific interface for the client container that allows only select registered nodes to be interacted with. The ROS master running in the proxy container syncs up with the host master, which does the actual work. This elaborate setup ensures that the user code is both hardware and network isolated, which means that a malicious user would have a hard time causing damage.

1.4 Cloud robotics

Cloud robotics is a novel concept that could offer easy access to composable ecosystems of pre-packaged services [43]. Using already existing cloud infrastructure to instantiate and access virtual machines has the following benefits:

- No expensive hardware needs to be acquired to run computationally heavy applications like Gazebo
- Everything can be managed and monitored through a convenient browser interface (Figure 17)
- Easy to scale up

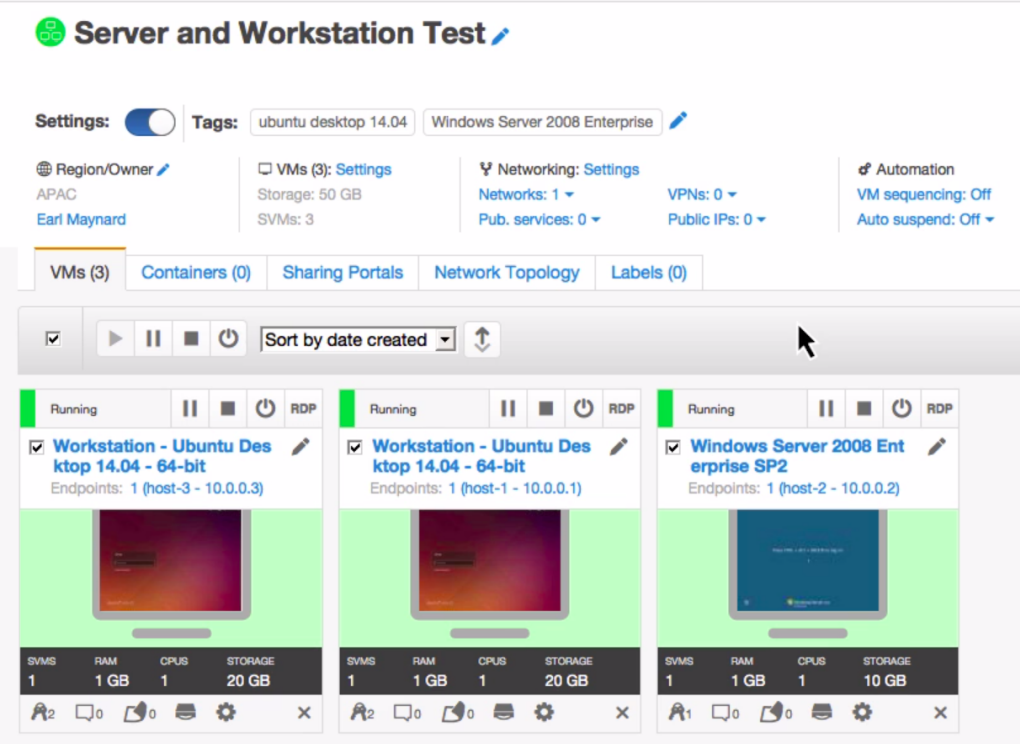


Figure 17. *Skytap virtual machine management interface [44]*

An example of an IaaS (Infrastructure as a Service) company is Skytap, it specialises in cloud automation and offers virtual machine management, development and testing services. Furthermore, it has developed a way to easily access the desktop environment of the VM from a web browser by using their own SRA (Secure Remote Access) client [44]. The VMs can also be shared using a "sharing portal", which is a URL-based access control feature that can be used to set the session duration, authenticate users and set permissions (like allowing starting and stopping of VMs) [45]. An application could potentially be built that integrates

Skytap's API and provides links to the VMs within sharing portals. Remote access to the robots could be enabled by creating a VPN (Virtual Private Network) link between the cloud network and the local one, where the robots are stationed [46].

Some of the issues with using Skytap's services would be dealing with platform changes that tend to cause incompatibilities between the existing application code and the changing API endpoints. Deprecation and new versions would compromise the application's stability - committing to a single cloud provider can backfire. Moreover, the instructions that would need to be provided to the users on how to use the sharing portals could become outdated and require updates. The closest Skytap's servers are located in Germany, which can potentially increase the latency of Estonian users when compared to locally-run servers. A hybrid architecture for a remote learning web application is an interesting prospect, the computationally-heavy robotics simulations could be done in the cloud, while the servers for remote access to the real robots could be hosted locally, where the networking of robots is easier to manage.

2 REQUIREMENTS

This thesis is a part of a larger project that aims to create an online ROS course for beginners. It would be beneficial if the students taking the course do not have to go through a painstaking environment setup that can bring an end to any early excitement they might feel about the learning prospect, so web technologies that could potentially avert such a case were under consideration. Therefore, the aim of this thesis was to prototype a web application that allows students to access physical ROS robots remotely, and in the process determine the optimal specification for the server hardware.

2.1 Functional requirements

2.1.1 Browser-based access to a learning environment that mimics the Linux OS a ROS developer uses.

2.1.2 The learning environment is preconfigured to connect with a physical mobile robot.

2.1.3 The learning environment can also be used to validate ROS programs in a Gazebo simulation.

2.1.4 The learning environments are available simultaneously for multiple learners.

2.1.5 A video feed is provided about the physical robot.

2.2 System requirements

2.2.1 Software

- Virtualization to enable multiple sandbox environments within a single server computer
- ROS learning environment:
  - ROS Noetic
  - Ubuntu 20.04 (Focal)
  - GNOME as the default desktop environment

2.2.2 Hardware

- Omni-directional mobile robotics platform Robotont:
  - Intel Core i5 (7th Gen) 7260U (2 cores, up to 3.4 GHz)
  - RAM DDR4 2133 MHz 4 GB
  - GPU Intel Iris Plus Graphics 640
  - Network: Intel Dual Band Wireless-AC 8265, IEEE 802.11a/b/g/n/ac

3 DESIGN

Taking into account the previously laid out requirements, the general architecture depicted in Figure 18 was developed. It consists of the following key components:

- Remote desktop display software that can embed the client in a web browser.
- Virtualization - the management of "clean" desktop environments that will be served over the internet.
- The backend and frontend frameworks for system automation and simple prototype deployment.

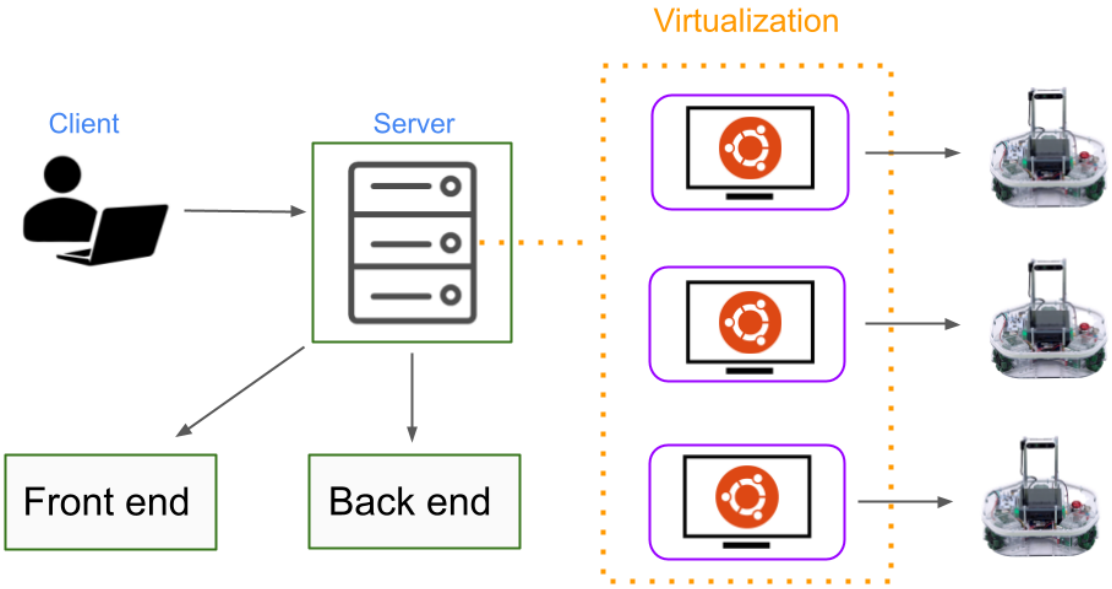


Figure 18. *The general components required for the system*

The next sections will describe in detail what solutions were deemed to be most suitable for each of the components and what approaches were used in implementing them. Section 3.1 looks into what virtualization solutions were prioritised, section 3.2 covers the investigation done in finding the most appropriate remote desktop tool. Section 3.3 gives an account of the server's configuration and software, while section 3.4 describes the hardware.

3.1 Creating virtual environments

In order to provide users with sandbox environments where they can do whatever they desire, and also to ensure that they are isolated from the server computer (where the operational processes that should not be interfered with are located), some form of virtualization should be present (see section 1.3). The first option considered was to use virtual machines, management of which can be outsourced to cloud services (see section 1.4). Another prospect was the use of Docker containers, similarly to Anand *et al.* [33] and Wiedmeyer *et al.* [42].

When it comes to containerization technologies Docker is by far the most comprehensive software framework available. There is extensive documentation for it due to its widespread adoption in cloud computing, where it enables easy deployment, management and scaling of applications. What is equally valuable are the abundance of open source projects that incorporate Docker and showcase the ins-and-outs of working with it, including the creation of operable desktop environments for the containers [47], [48], [49], hence it was decided to build upon it.

During the outlining of a Docker-based system, two problems were defined:

1) How to enable automation of a container's lifecycle in a web-based context (i.e. starting, stopping, monitoring them remotely)?
2) How to establish the container-robot network connections in the context of ROS communication?

The principal control point from which the containers are booted is the Docker daemon process, where a server is located that can be accessed through the Docker Engine API (Figure 19); it enables management of containers, images, networks and data volumes (sharing of files with the host system).

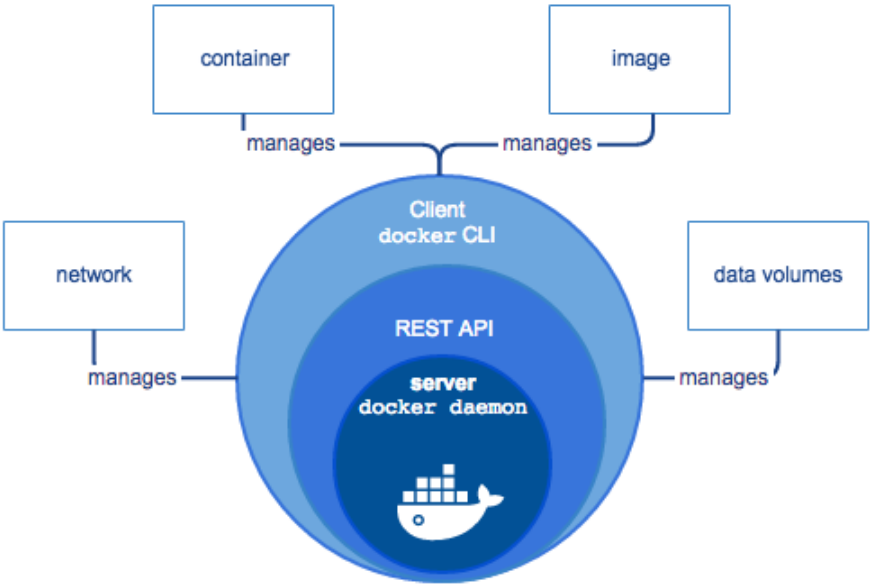


Figure 19. *Interaction layers for Docker* [50]

Although the most common way to interact with this API is through the Docker CLI client, a different approach needed to be taken in order to deal with the first problem. If the API was exposed to the outside directly, it would introduce extra overhead in authentication and authorization management, for the latter a plugin might even be required [51]. If a multi-server system exists with multiple Docker hosts, such an option is not viable. For these reasons it was decided to implement an API proxy, where only specific container operations could be exposed and authentication/authorization could be managed centrally.

Next, in order to solve the second problem, the options that Docker provides in regard to networking were examined [52]. The default bridge network driver is used in interconnecting applications deployed in separate containers, which does not help in attaching them to a LAN. An option exists to share the host's network, but in that case there is potential only for a single robot-connected container due to port conflicts. The most logical choice was to use the *macvlan* network driver. It stands for MAC virtual LAN, and permits containers to appear as separate network entities (Figure 20).

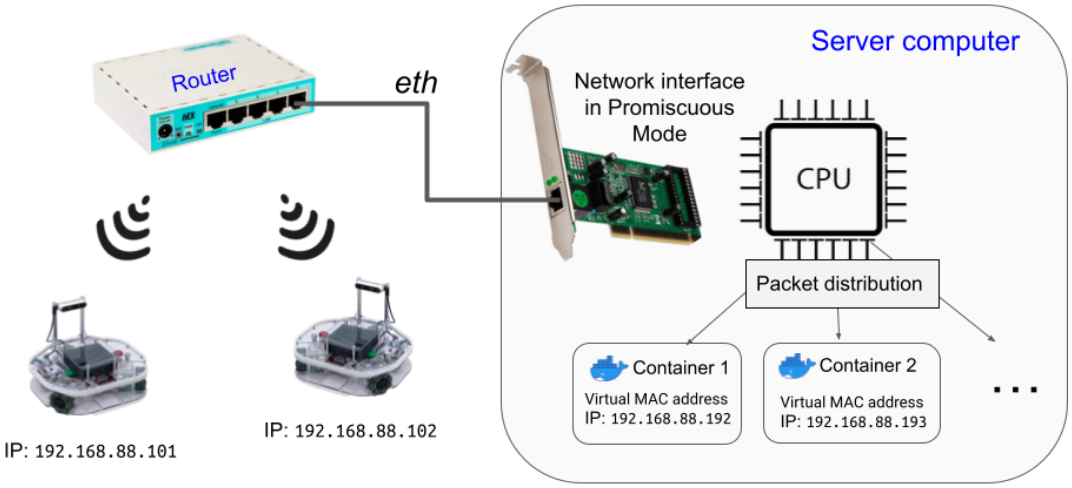


Figure 20. *MAC VLAN*

The process of packet routing is as follows:

1) When the packets coming from the containers are sent out by the host computer, the MAC address of the sender (host MAC) is changed to a virtual one. The router will pass them along according to their target (the robot).

2) The robot will send back the response, but the router does not know where to forward it, since the MAC address is spoofed, so it sends it to all of the connected devices.

3) Usually a network interface card (NIC) will drop any packets not addressed to it, but if promiscuous mode is enabled then it will instead pass all of them to the computer's CPU, where the Docker macvlan network will distribute them accordingly.

3.2 Remote desktop in-browser display

A popular and well-established cross-platform remote desktop system is called VNC (Virtual Network Computing). In its implementation the frames, which contain information about the pixel locations and their color values, are read from a RAM store known as the framebuffer and sent over the internet. VNC consists of two parts - the server and the client (Figure 21). To integrate the VNC view into a web browser, the client is the component that needs to be adapted.

Figure 21. *The two components of VNC*

In order to fulfil the requirements set for this thesis, both proprietary and open-source VNC software was examined, with the focus on readily available browser integrations.

RealVNC is a commercially successful company that originated the RFB (Remote Framebuffer) protocol. The VNC SDK (Software Development Kit) that it provides for developers to create custom applications also offers the option to embed the VNC view into a web browser [53]. The SDK was tested but was not found to be a viable solution. Firstly, some serious latency issues were encountered, perhaps due to the fact that both the client and the server needed to be connected via RealVNC's cloud, where the browser implementation is hidden. Secondly, a way to scale an application with the limited options this proprietary software offered was not at all obvious.

The open-source VNC HTML client known as noVNC was considered next [54]. The project is actively maintained and runs in any modern web browser. In the VNC protocol the client is the one that dictates what encoding will be used, which means that the server is usually required to support any pixel format the client needs; noVNC supports decoding of different types: Raw, Tight, JPEG, TightPNG etc. The VNC server chosen to work in tandem with noVNC was the general-purpose, highly performant TigerVNC [55]. The following encodings were found to function in the communication between noVNC and TigerVNC: Raw, Hextile, Tight, ZRLE. Due to the fact that both noVNC and TigerVNC can be easily integrated into a

larger project, and offer plenty of features (e.g. in-built authentication), it was decided that these components would be the backbone of the in-browser display of a remote desktop. The source code for the Dockerfile defining an image with a TigerVNC server is found in Appendix 1.

3.3 Server architecture

Having established the primary components to be used in the server, some due consideration was required as to how to wire them together. This section introduces the technologies that were chosen to aid in VNC/ROS container management, and reveals how each of the parts was integrated with others.

3.3.1 VNC client reverse proxies for multiple connections

The base interface the users of the web application would use is the already mentioned noVNC client (section 3.2) that uses the WebSockets protocol, which is the standard for full-duplex real-time web applications. However, TigerVNC does not have support for it, as it is a simple HTTP server, so a program called Websockify is used to translate the Websockets traffic to TCP traffic (Figure 22).

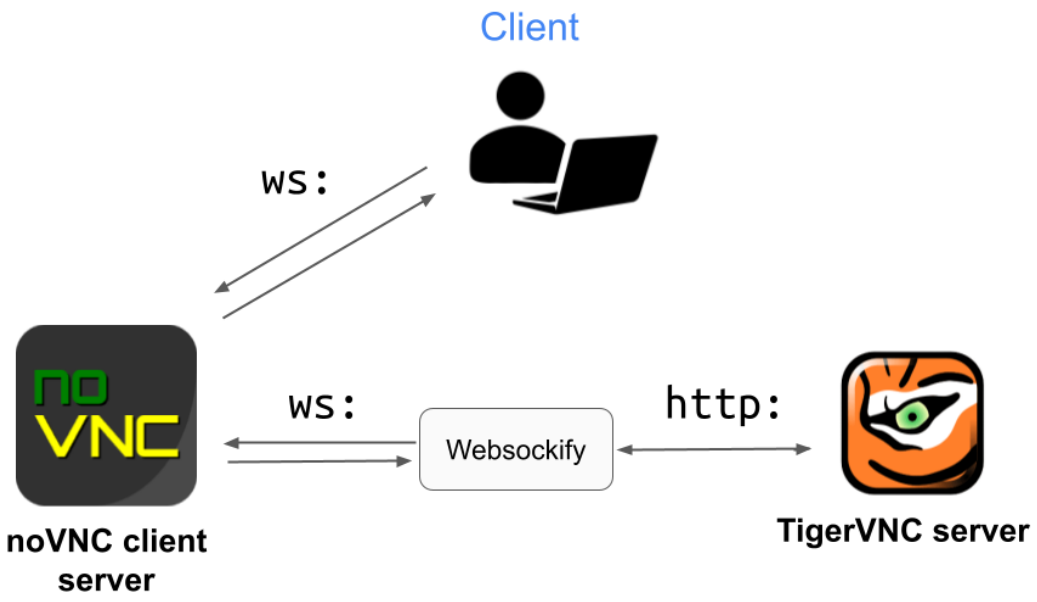


Figure 22. *Websockify - the WebSockets to TCP proxy*

In the initial stages of testing, both the VNC client and server were being run inside the Docker containers, each with their own Websockify instance. This, however, causes unnecessary overhead, so the redundant clients were combined into a single host instance that is capable of routing all the VNC traffic coming from the containers, each of which can be distinguished by a URL token passed to Websockify.

To have all the containers available from a single entrypoint, it was important to find a way to proxy the different container VNC paths. The most convenient tool for this was determined to be Nginx, a versatile open-source web-server. Initially, when multiple VNC clients were present, a separate path was used for each of them, but setting up individual proxies for an unknown amount of containers is not scalable or maintainable, so these were eliminated with the consolidation of the clients. Figure 23 shows the corresponding Nginx configurations for these cases.

```
location /robo-1 {
        rewrite /robo-1/(.*) /$1 break;
        proxy_pass http://robo-1:6080/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
}

location /robo-2 {
        rewrite /robo-2/(.*) /$1 break;
        proxy_pass http://robo-2:6080/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
}
```

(a)

```
location /novnc {
      # Websockify bridge
      proxy_pass http://localhost:6085/;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "Upgrade";
      proxy_set_header Host $host;
      proxy_read_timeout 600s;
      # Disable caching of VNC stream.
      proxy_buffering off;
}
```

```
Specifying the target:

  /novnc/vnc.html?token=robo-1
  /novnc/vnc.html?token=robo-2
```

(b)

Figure 23. *Nginx configurations:*
(a) Proxying individual VNC clients,
(b) Single Websockify instance with parameter routing

3.3.2 Remote container management

With the VNC part of the system being instituted, a way to give a user the privilege to start their own container with a password-protected VNC server was required. For this purpose two components needed to be introduced:

1) The front-end application served to the user
2) The back-end server responsible for responding to the user requests, and communicating with the Docker Engine API

The language chosen for the back-end server was the ubiquitous JavaScript, with server-side scripting made possible by the NodeJS runtime environment. The primary reason for this being the multitude of open-source packages available via NPM (Node Package Manager) - the world's largest software registry [56].

Three packages were found to be particularly useful in enabling remote container management (Table 1).

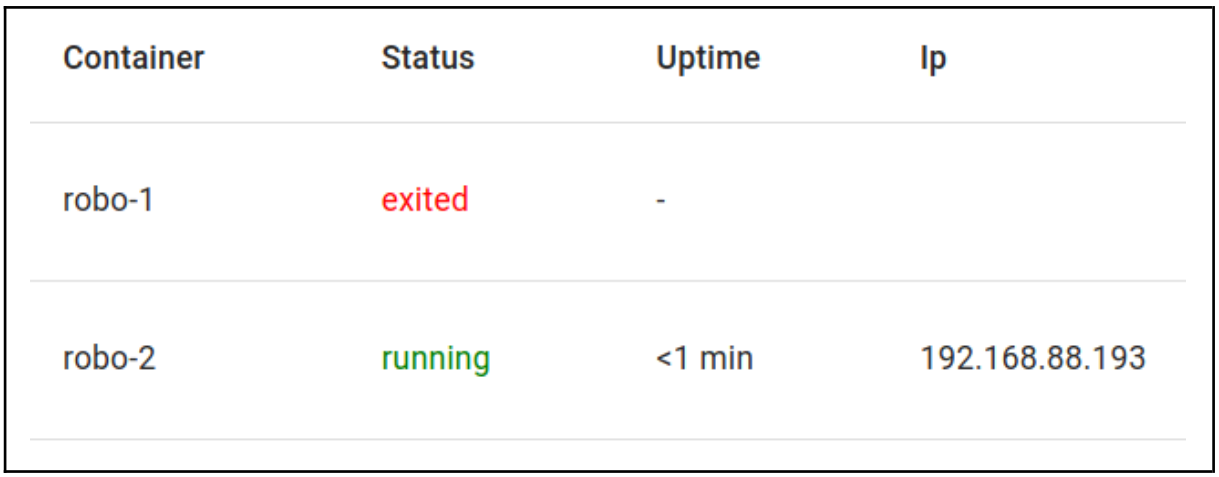
npm package	Description
`express` [57]	*de facto* standard server framework for NodeJS
`dockerode` [58]	Abstraction of interactions with the Docker Engine API
`docker-compose` [59]	Creation of child processes to start containers using the docker-compose configuration files

Table 1. *The npm packages used in the back-end server*

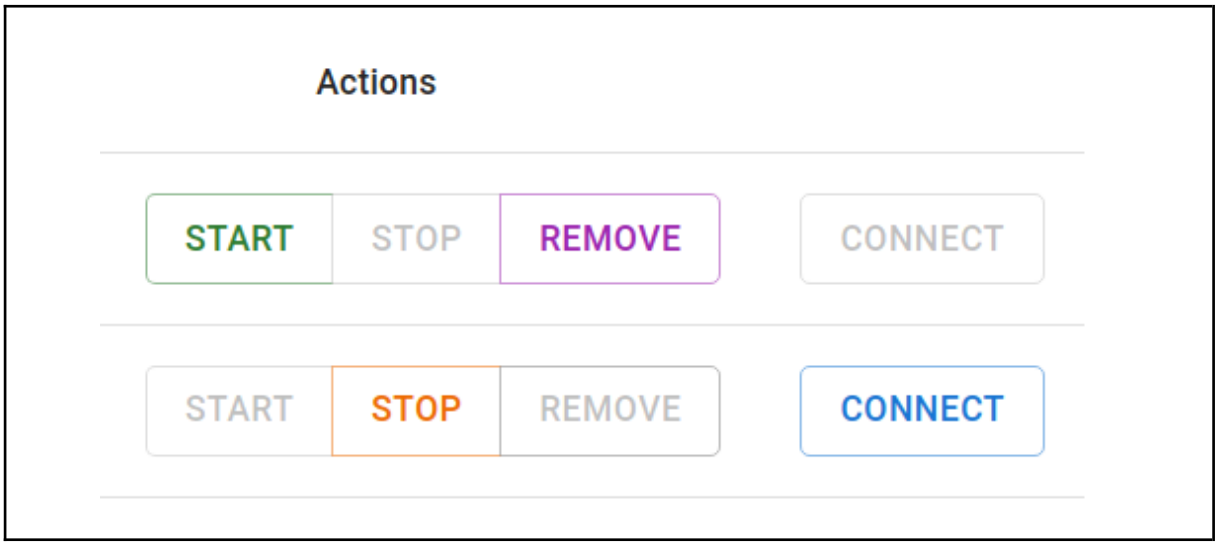
The minimalist Express framework allows to quickly bootstrap a NodeJS server capable of responding to HTTP requests, while Dockerode provides a simple way to communicate with the Docker Engine API on behalf of the client.

Docker Compose is a tool that allows to define the container startup configuration in a single YAML file. Once the server receives a POST request for starting a container, a custom `docker-compose.yaml` file then is generated that has the VNC password specified as an environment variable. Docker Compose is interfaced with the homonymous docker-compose package that with the help of NodeJS's `child_process` module starts the container in a subprocess. Once the container initialises, the environment password is accordingly passed to TigerVNC. All further actions, e.g. container inspection and termination, are done via Dockerode.

A front-end application with which to test the container management API was built based on the ReactJS library that makes it simple to manage the application state; Figure 24 shows the example interface. The noVNC client provides an option to autofill the VNC password by passing it in the URL, so when a user has initialised the container using the provided "Start" button, the link with the corresponding password token is returned within the "Connect" button.



(a)



(b)

Figure 24. *Fronted interface: (a) Container status table, (b) User actions*

3.3.3 Final system overview

Figure 25 shows how all the parts described previously are interconnected. The main entrypoint that consolidates the system is the Nginx web server which was introduced in section 3.3.1. The primary purpose of the main application is to eventually transfer the user to the noVNC client application, so the front-end and back-end components are there to support this by way of authenticating users and giving them the authority to start up their own environments. The source code for the server system's software is available in Appendix 1.

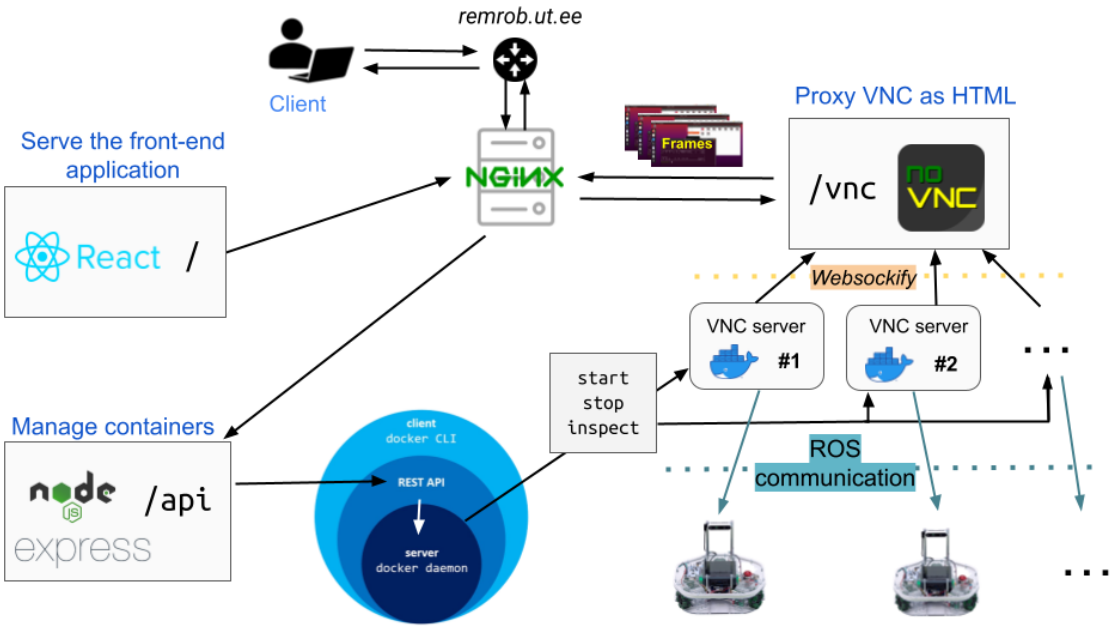


Figure 25. *The server system architecture*

3.3.4 Server hardware

The server computer used in running the containers and testing the system was an MSI GE66 Raider 11UH; its hardware is specified in Table 2.

CPU	11th Gen Intel® Core™ i9-11980HK @ 2.60GHz × 16
GPU	NVIDIA GeForce RTX 3080
RAM	32GB DDR4
Network	Killer Gb LAN (Up to 2.5G); Killer ax Wi-Fi 6E

Table 2. *The server computer's hardware*

The load it could handle was tested by running eight containers all connected to a single robot and displaying a depth map of its surroundings. Figure 26 shows a snapshot of the NGINX Amplify server monitoring interface during this test, where it can be seen how the CPU utilisation grows with each new mapping program started. The eight containers yielded a 90% CPU load, at which point their performance was starting to degrade (low data display framerate).

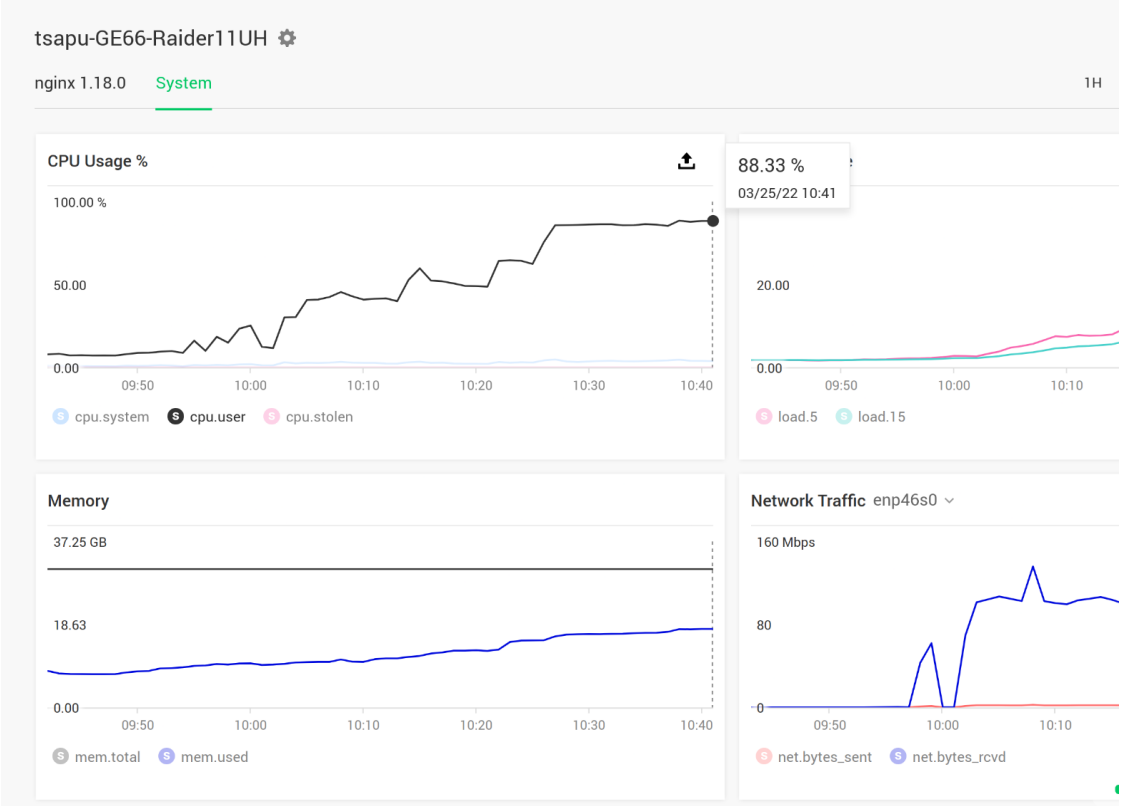


Figure 26. *Testing the CPU load of the containers*

The conclusion that can be made from this test is that in order to provide performant containers to more than just a few users either a multi-server system needs to be devised with more than a single Docker host or an even more powerful server computer must be acquired.

4 DISCUSSION AND FUTURE WORK

The objectives of this thesis were completely fulfilled - a prototype web application was developed which gives remote learners the ability to program physical ROS robots with a natural Linux OS interface. However, when considering the larger project this thesis was a part of, some improvements can be done and additional features can be implemented.

4.1 Limitations

Currently it is possible to see the remote lab via the robot's own Intel Realsense D435 camera. It would be beneficial if the user could also see the robot itself, preferentially from multiple different angles, which could be done by setting up video cameras and serving their stream over the internet. This would enable the user to have the remote video in a separate browser window, which would allow for a more convenient workflow.

Since ROS is a distributed computing environment, communication between a container and the robot requires full, bi-directional connectivity on all ports. This was successfully achieved by setting up a *macvlan* network configuration. However, this open local network also means that there is nothing preventing one container from accessing other robots not intended for its use, or interacting with other containers. Moreover, the containers were also opened up to the host in order to enable communication with the VNC servers, so the Docker Engine API is also freely available since no authentication is currently enabled for it. A possible solution to prevent misuse is to implement a blacklist firewall, which would allow a single container to only access a single robot, and in communication with the host whitelist only the VNC ports. As a general preventive measure securing the Docker daemon socket with TLS (Transport Layer Security) should be considered.

By default Docker containers have the undesired root privileges, so a configuration to run them as a regular user was added. Yet, in comparison to other, more lightweight desktop environments (e.g. LXQt, LXDE, XFCE) running GNOME inside a Docker container requires *systemd*, the standard system and service manager for Linux operating systems. To enable it in a Docker container SYS_ADMIN capabilities need to be granted and the host's cgroup needs to be mounted as a shared volume. Whether this introduces any dangerous vulnerabilities and whether they can be avoided was not determined in the scope of this work.

4.2 Future work

This thesis was a part of a larger project, which aims to create a comprehensive online ROS course for beginners. This means that a database would need to be added that could provide the base for the user system, where default and administrator privileges could be distinguished.

In order to automate the access to the remote robots and create a smooth user experience, reservation of time slots for the robot use would need to be made possible with a booking system.

Docker provides an easy way to save the state of the container in a new image via the "committing" process, so to have the user environment persist (i.e. any files or programs they might have created or installed), this needs to be incorporated into the container management API and the prospective database.

It can be frustrating if a student encounters a problem with the remote robot, but is not able to do anything about it (e.g. the robot gets stuck in some place), so some type of communication protocol with the on-site personnel should be established.

REFERENCES

[1] K. Lamesoo & E. Tagamets, Emergency Remote Education in Higher Education Institutions: Estonia's Response to COVID-19. Methodology.;19:4

[2] T. Vaimann, M. Stępień, A. Rassõlkin and I. Palu, "Distance Learning in Technical Education on Example of Estonia and Poland," 2020 XI International Conference on Electrical Power Drive Systems (ICEPDS), 2020, pp. 1-4, doi: 10.1109/ICEPDS47235.2020.9249317

[3] N. T. Fitter, N. Raghunath, E. Cha, C. A. Sanchez, L. Takayama and M. J. Matarić, "Are We There Yet? Comparing Remote Learning Technologies in the University Classroom," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 2706-2713, April 2020, doi: 10.1109/LRA.2020.2970939

[4] https://www.theconstructsim.com/need-robotics-standards/ (Accessed May 07, 2022)

[5] https://metrics.ros.org/misc_citations.html (Accessed Mar. 05, 2022)

[6] http://wiki.ros.org/ROS/Tutorials (Accessed May 07, 2022)

[7] Michieletto, Stefano & Ghidoni, Stefano & Pagello, Enrico & Moro, Michele & Menegatti, Emanuele. (2014). Why teach robotics using ROS?. Journal of Automation, Mobile Robotics & Intelligent Systems. 8. 60-68. 10.14313/JAMRIS_1-2014/8

[8] https://wiki.ros.org/ROS/Introduction (Accessed Mar. 05, 2022)

[9] http://wiki.ros.org/Master (Accessed Mar. 05, 2022)

[10] https://robolabor.ee/homelab/en/ros/subscribepublish (Accessed Mar. 05, 2022)

[11] https://www.turtlebot.com/ (Accessed May 18, 2022)

[12] https://robotont.github.io/html/files/overview.html (Accessed May 18, 2022)

[13] http://wiki.ros.org/ROS/NetworkSetup (Accessed Mar. 31, 2022)

[14] https://robotont.github.io/html/files/setup_robot_pc.html#getting-the-robotont-and-pc-into-the-same-ros-environment (Accessed Mar. 05, 2022)

[15] http://wiki.ros.org/Security (Accessed May 19, 2022)

[16] http://robotwebtools.org/ (Accessed Mar. 01, 2022)

[17] http://wiki.ros.org/rosbridge_suite (Accessed Mar. 01, 2022)

[18] http://wiki.ros.org/roslibjs (Accessed Mar. 01, 2022)

[19] M. Karaca and U. Yayan, *ROS Based Visual Programming Tool for Mobile Robot Education and Applications*. 2020.

[20] https://developers.google.com/blockly/ (Accessed Apr. 10, 2022)

[21] D. D. Rajapaksha *et al.*, 'Web Based User-Friendly Graphical Interface to Control Robots with ROS Environment', in *2021 6th International Conference on Information Technology Research (ICITR)*, Dec. 2021, pp. 1–6. doi: 10.1109/ICITR54349.2021.9657337

[22] http://gazebosim.org/gzweb (Accessed Apr. 10, 2022)

[23] B. Pitzer, S. Osentoski, G. Jay, C. Crick and O. C. Jenkins, "PR2 Remote Lab: An environment for remote development and experimentation," 2012 IEEE International Conference on Robotics and Automation, 2012, pp. 3200-3205, doi: 10.1109/ICRA.2012.6224653

[24] Lee, Jihoon. "Web Applications for Robots using rosbridge." Brown University, 2012

[25] T. O. Almeida, J. F. de M. Netto and M. L. Rios, "Remote robotics laboratory as support to teaching programming," 2017 IEEE Frontiers in Education Conference (FIE), 2017, pp. 1-6, doi: 10.1109/FIE.2017.8190472

[26] R. Grandi, R. Falconi, C. Melchiorri, UniBot Remote Laboratory: A Scalable Web-Based Set-up for Education and Experimental Activities in Robotics, 2011, https://doi.org/10.3182/20110828-6-IT-1002.03103

[27] D. Pickem et al., "The Robotarium: A remotely accessible swarm robotics research testbed," 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 1699-1706, doi: 10.1109/ICRA.2017.7989200

[28] G. A. Casañ, E. Cervera, A. A. Moughlbay, J. Alemany and P. Martinet, "ROS-based online robot programming for remote education and training," 2015 IEEE International

Conference on Robotics and Automation (ICRA), 2015, pp. 6101-6106, doi: 10.1109/ICRA.2015.7140055
[29] M. Kulich, J. Chudoba, K. Kosnar, T. Krajnik, J. Faigl and L. Preucil, “SyRoTek—Distance Teaching of Mobile Robotics,” in IEEE Transactions on Education, vol. 56, no. 1, pp. 18-23, Feb. 2013, doi: 10.1109/TE.2012.2224867
[30] The Construct: A Platform to Learn ROS-based Advanced Robotics Online, https://www.theconstructsim.com/ (Accessed 15.04.2022)
[31] Ricardo Tellez. “A thousand robots for each student: Using cloud robot simulations to teach robotics”. In: Robotics in Education. Springer, 2017, pp. 143–155
[32] The Construct. “How to use Robox 24/7 ROS Remote Real Robot Lab” YouTube, 30.11.2020, https://www.youtube.com/watch?v=8Met5vzusig
[33] H. Anand et al., “OpenUAV Cloud Testbed: a Collaborative Design Studio for Field Robotics,” 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), 2021, pp. 724-731, doi: 10.1109/CASE49439.2021.9551638’.
[34] DREAMS Laboratory. “Jezero Crater initial world for NSF CPS Challenge Mars 2020 Edition in OpenUAV testbed” YouTube, 10.05.2020, https://www.youtube.com/watch?v=w9bnfIWW09U
[35] Kolyshkin, Kirill. “Virtualization in linux.” White paper (2006)
[36] https://www.docker.com/blog/docker-index-shows-continued-massive-developer-adoption-and-activity-to-build-and-share-apps-with-docker/ (Accessed Mar. 03, 2022)
[37] Amit M Potdar, Narayan D G, Shivaraj Kengond, Mohammed Moin Mulla, “Performance Evaluation of Docker Container and Virtual Machine”, Procedia Computer Science, Volume 171, 2020
[38] https://hub.docker.com/r/osrf/ros/ (Accessed Mar. 06, 2022)
[39] White, Ruffin & Christensen, Henrik. (2017). ROS and Docker. 10.1007/978-3-319-54927-9_9
[40] T. Combe, A. Martin and R. Di Pietro, “To Docker or Not to Docker: A Security Perspective,” in IEEE Cloud Computing, vol. 3, no. 5, pp. 54-62, Sept.-Oct. 2016, doi: 10.1109/MCC.2016.100
[41] https://github.com/OWASP/Docker-Security (Accessed May 23, 2022)
[42] W. Wiedmeyer, M. Mende, D. Hartmann, R. Bischoff, C. Ledermann and T. Kroger, “Robotics Education and Research at Scale: A Remotely Accessible Robotics Development Platform,” 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 3679-3685, doi: 10.1109/ICRA.2019.8793976
[43] G. Toffetti and T. M. Bohnert, ‘Cloud Robotics with ROS’, in *Robot Operating System (ROS): The Complete Reference (Volume 4)*, A. Koubaa, Ed. Cham: Springer International Publishing, 2020, pp. 119–146. doi: 10.1007/978-3-030-20190-6_5.
[44] https://help.skytap.com/accessing-vms-with-a-browser.html (Accessed May 03, 2022)
[45] https://help.skytap.com/sharing-portals.html (Accessed May 03, 2022)
[46] https://help.skytap.com/wan-create-vpn.html (Accessed May 03, 2022)
[47] https://github.com/fcwu/docker-ubuntu-vnc-desktop (Accessed Apr. 25, 2022)
[48] https://github.com/ConSol/docker-headless-vnc-container (Accessed Apr. 25, 2022)
[49] https://github.com/theasp/docker-novnc (Accessed Apr. 24, 2022)
[50] https://docs.docker.com.xy2401.com/engine/docker-overview/ (Accessed May 05, 2022)
[51] https://docs.docker.com/engine/extend/plugins_authorization/ (Accessed May 06, 2022)
[52] https://docs.docker.com/network/ (Accessed May 06, 2022)
[53] https://www.realvnc.com/en/news/control-computer-within-your-web-browser/ (Accessed Apr. 25, 2022)
[54] https://github.com/novnc/noVNC (Accessed Apr. 25, 2022)
[55] https://tigervnc.org/ (Accessed Mar. 04, 2022)
[56] https://docs.npmjs.com/about-npm (Accessed May 17, 2022)
[57] https://github.com/expressjs/express (Accessed May 17, 2022)
[58] https://github.com/apocas/dockerode (Accessed May 17, 2022)
[59] https://www.npmjs.com/package/docker-compose (Accessed May 17, 2022)

APPENDIX

1. Source code

`https://github.com/unitartu-remrob/remrob-docker`
`https://github.com/unitartu-remrob/remrob-server`

2. Video demonstration

`https://www.youtube.com/watch?v=PbIzgEJhSUg`

3. Non-exclusive licence to reproduce thesis and make thesis public

I, **Dāvis Krūmiņš**,

1. grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

Web-based learning and software development environment for remote access of ROS robots,

supervised by Prof. Karl Kruusamäe and PhD Veiko Vunder.

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Dāvis Krūmiņš

27/05/2022