Source Domain Knowledge Acquisition Using Simulated Environment with Minimized Computing Requirements

Zvezdan Lončarević[1,2], **Andrej Gams**[1,2], **Mihael Simonič**[1,3]

[1]*Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia*
[2]*Jožef Stefan International Postgraduate School, Jamova cesta 39, 1000 Ljubljana, Slovenia*
[3]*Faculty of Electrical Engineering, Tržaška cesta 25, 1000 Ljubljana*
{*zvezdan.loncarevic,andrej.gams,mihael.simonic*}@*ijs.si*

Abstract

Transfer learning became popular technique when learning new robotic skills. It removes burden of the need of doing a lot of reinforcement learning iterations on the real robotic system but puts more accent on the simulation environment in order to obtain the initial source domain knowledge. Although this method accelerates learning, setting up the simulation environment can be demanding in the sense of connecting all the framework components, dealing with incompatibility issues and also computationally expensive. In this paper, we present a scalable framework for gathering the database of throwing actions for humanoid robot Talos using open-source tools including ROS, Gazebo and Docker. Presented method solves the problem of compatibility issues, allows easy changes in the simulated environment, enables parallelization and also reduces computing requirements with offering a possibility for the user to switch on or off graphical interface. This allows the processes to run in the background while normally using computer for everyday work or in cloud environments and high performance clusters, where graphical environment is not accessible. Introspection is still possible using the web interface.

1 Introduction

Adaptation of the robotic skills is one of the crucial facilitators for the robots to enter everyday human lives [1] and also for the fast-changing production lines to keep up with increasing market demands. Reinforcement learning (RL) has made a lot of progress towards learning new skills without necessary expertise knowledge, but its application is connected mainly to the simulation environments. This is caused by the RL algorithms having relatively low sample efficiency [2] thus causing the huge amount iterations of learning required on the real robot. Moreover, actions generated during the exploration of these algorithms might lead to the actions that exceed the robot limits causing the risk of damaging the equipment and robot itself [3].

One of the most widely spread solutions for this problem is called transfer learning (TL). The main idea is to train the robot in simulation (source domain) and then to quickly adapt the knowledge to the real world (target domain) [4, 5]. When using TL, majority of the knowledge is collected in the simulation and only small amount in

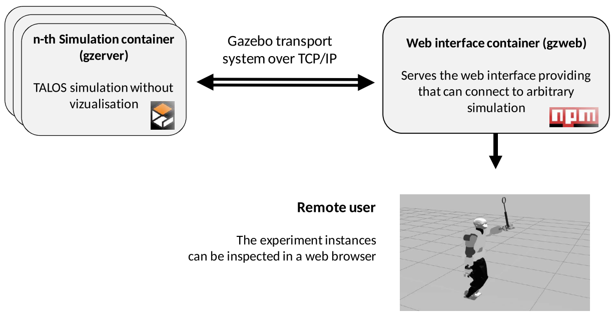
n-th Simulation container (gzerver)
TALOS simulation without vizualisation
Gazebo transport system over TCP/IP
Web interface container (gzweb)
Serves the web interface providing that can connect to arbitrary simulation
npm
Remote user
The experiment instances can be inspected in a web browser

Figure 1: Main components of the simulation setup.

real world. Collecting the data in simulation should be quick and easy in theory. However, in practice this is usually not the case because of the incompatibility problems, slow execution and communication between different components in the system.

It has been noted earlier that most of the well-know simulators have a high entrance barrier, due to the complicated initial set up. In [6] the authors presented an open source test bed for Unmanned Aerial Vehicles education and research that overcomes these barriers. They leveraged the containers technology to carry out simulations on the cloud, so that the end-user has no special hardware requirements. Our approach shares some similarity with [6] by relying on containerisation to ease the setup, but is aimed at data collection and as such focuses on parallelization and scalability.

To our knowledge, no other frameworks that enable simple and parallel data collection were previously presented. We test our framework using a highly complex humanoid robot Talos and the throwing task which requires additional effort in establishing communication in order to follow the ball trajectory.

The main novelty is that we designed a scalable approach for data collection using containerisation. The presented system allows to easily to run multiple simulations simultaneously while keeping it compatible with multiple platforms. Parallelism enables rapid data collection. On top of that, changing of the experimental setup is simple by changing parameters in a human readable YAML configuration file. In order for user to be able to continue its everyday work on the same computer, we also present an approach that allows enabling and disabling graphical interface and in that way reducing the need for computing power. This also enables to use it in cloud environments and high performance clusters, where graphical environment is not accessible.

The paper is organized as follows. In the next section, introduction of our framework, its main components and connection between them are explained. Section 3 shows example of successful usage of the presented approach on the use-case of robotic throwing action. The paper concludes with a short outlook and plans for the future work. An example of configuration of the setup is given in appendix.

2 Overview of the framework

2.1 Robot simulation and control

The Robot Operating System (ROS)provides a suitable framework for developing complex robotic systems with several computers running on different operating systems. Many features and tools are provided within ROS. In our framework we use the following:

- nodes – any program that has connectivity to the ROS network and can therefore access to and publish data across it (in our case, coordinating script, controllers, simulation, etc.)
- topics – a message bus, in which nodes can publish and subscribe to data provided by the other nodes, e. g. robot joint states, ball positions, etc.
- messages – a predefined structure to encapsulate data to be transferred over topics, e. g. robot joint states are written as `sensor_msgs/JointState`
- actions – a request/response based Remote Procedure Call (RPC) interface, that is used to trigger long-running preemptable processes (like executing trajectories) and continuously receive feedback (i.e. to track task execution progress)

Within the ROS community, one of the most often used software for dynamical simulation of robots is Gazebo [7]. Compared to other robot simulators, its biggest advantage is seamless integration into ROS [8]. The robots simulated in Gazebo can be controlled using the `ros_control` framework, which greatly simplifies the transfer of software from simulation to the real robot. The framework provides a hardware abstraction layer, that enables standardized access to actuators and thus to write robot-agnostic controllers [9]. At the same time, the same controllers can be used both in simulation and on the real robot. For Talos robot, the access to the robot middleware is handled by the `talos_hardware` ROS package, while Gazebo configuration is provided by the `talos_hardware_gazebo` package. Note that, Gazebo is based on server-client paradigm, where the computation is done in gzserver, and the result of simulation can be shown graphically either in gzclient or gzweb. This allows that the simulation can be done in headless mode on a cloud environment, and the remote user can view it over web browser as shown in Fig. 1.

2.2 Coordinating script

Although ROS contains standard tools for manipulating and path planning (such as MoveIt!), it is possible to write customized high-level software for performing different tasks with the robot in different programming languages.

As the main purpose of this system is to collect the database and to work with simulation, we need to be able to quickly develop or modify the task. For that purpose, this standard ROS tools are connected with Matlab and all the robot control is done with it. It enabled us to use RobotBlockSet that was developed within our lab and easily integrate the common functions that are required when operating any of the robots. These functions include direct and inverse kinematics, integration of parametric trajectory representations, transformations between different coordinate systems for the whole spectre of robots. This combination of matlab script and already developed functions for controlling the robot allows us to easily adapt the simulation to new tasks.

2.3 Containerization

Although robots can be easily controlled using ROS interfaces, setting the ROS environment still takes some effort and time. Its portability and ability to work with different coding languages makes it also dependent on different packages and therefore prone to having compatibility issues. This becomes a tedious process for the user, especially when the software needs to be deployed on multiple machines to run experiments in parallel.

Our development and overall system are based on Docker [10] to address these two issues. Docker images are isolated read-only templates created from Dockerfiles. The image can be run on any platform that supports Docker, and it will be the same regardless of the platform. The purpose of this file is to specify which software packages and what dependencies should be installed. A docker container is an executable instance of a docker image. Docker containers, unlike virtual machines, don't emulate the host's hardware, but share it instead. This in turn means that, compared to a virtual machine, a Docker uses fewer resources [6]. In terms of deploying simulation software across multiple machines, the developer designs the code in such a way that it runs within the Docker container and eliminates the commonly encountered problem of unmet dependencies.

2.4 Modules

The core of the presented approach is modular design. Essentially, we have four types containers communicating between each other as shown in Fig. 2. The containers are isolated, but can communicate using TCP/IP protocol, similarly as different computers do in a local network:

- ROS Master is running in a container based on the freely available `ros:melodic` Docker image. It is responsible for initiating and terminating connections between ROS nodes, as well as keeping track of topics and actions.

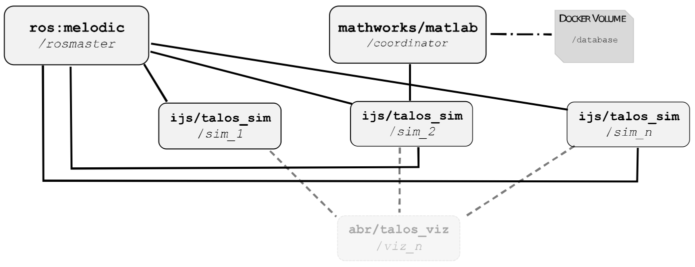

Figure 2: Connectivity between the containers comprising the proposed system. Each rectangle represents a Docker container (image name is given in bold, and container name is given in italics). TCP/IP connections between containers are given with full lines. Optional connections are shown with dashed lines. For persistent storage, a database is provided as a separate Docker volume on the local filesystem (dash-dotted line).

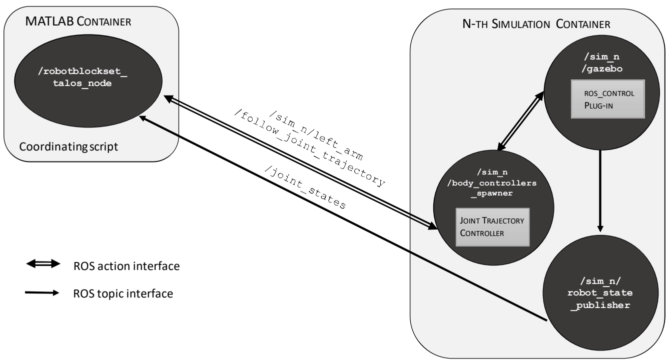

Figure 3: ROS connections between the coordinating script running in the Matlab container and a simulation instance in Gazebo container. The coordinating script itself is a ROS node, that subscribes to joint states via ROS topic interface, and can send/interact with goals of the joint trajectory controller using ROS Action interface. Each simulation container consists of a body controller spawner node, robot state publisher and simulator node. The body controller spawner loads the joint trajectory controller plugin for each of the robot limbs. The robot state publisher advertises this information over the ROS network. The Gazebo simulator node runs the gzserver.

- Each simulation instance is running in a `ijs/talos_sim`-based container. It contains PAL Robotics' TALOS simulation stack[1] along with customisations needed to simulate the process of interest (in our case ball and throwing tool models were added).

- The data collection script is running in a `mathworks/matlab`-based container. It coordinates the data collection process by starting and tracking the simulation instances.

- Optionally, a web server can be initialized using the `ijs/talos_viz` image. It allows the operator to introspect the state of the arbitrary simulation instance, simply by opening a web browser (c.f. Fig. 4).

As we mention previously, communication between the nodes is one of the main characteristics of ROS. In this case, the coordinating script running in the Matlab container is a ROS node, and it is able to communicate with the simulation instance in the Gazebo container using the ROS topics and actions, as shown in Fig. 3.

[1]`https://github.com/pal-robotics/talos_simulation`

2.5 Usage and scalability

The framework aims at ease of use and scalability. The entire system can be started using `docker compose up` command, which runs all the above mentioned containers. It is configured to automatically initialize the needed services. An example configuration is given in the appendix.

3 Throwing use-case

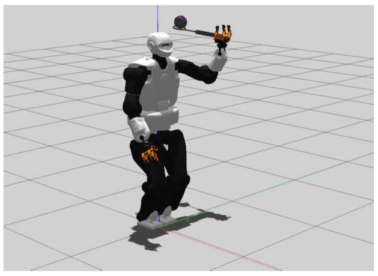

Figure 4: Example of using the developed approach for collecting the data in Gazebo environment

As mentioned in the introduction, we presented use of this approach on the robotic throwing action [11]. For this purpose, we used the model of full-sized humanoid robot Talos that was equipped with the throwing spoon - Fig. 4. When learning to throw at the requested target, robot needs a lot of trials. The idea is to use simulation as the source domain to gather the huge database and train neural network to predict trajectory parameters based on the desired target. Later, with only few shots this knowledge would be adapted to the target domain - real world.

4 Conclusion and future work

In this paper, we have presented an approach that allows us to gather the huge amount of data in a short period of time. Docker containers enabled our system to be lightweight and compatible with the multiple platforms while avoiding the compatibility issues between packages that usually may arise when setting up the simulation. Possibility of turning visualization on or off reduces the need for hardware resources while still allowing us to easily check the learning process when necessary. Also, easy and fast set-up procedure allows us to run simulation on multiple computers within the same network thus enabling great amount of parallelization. On top of that, possibility of controlling the robot through simple Matlab script allows quick modification of parameters and control of robot behaviour thus making this useful also to some other tasks like reinforcement learning.

In the future, we plan to test the possibility of running similar approach on a High Performance Computing Cluster. In order to do that, we first need to prepare Singularity containers based on Docker images. In order to release a full open source solution, we also plan to prepare the coordinating script in Python.

A Example docker-compose configuration

```
networks:
  ros:
    driver: bridge
services:
  rosmaster:
    image: ros:melodic
    command: roscore
    environment:
      - "ROS_HOSTNAME=rosmaster"
    networks:
      - ros
  sim_1:
    image: ijs/talos_sim
    depends_on:
      - rosmaster
    command: roslaunch talos_gazebo \
                      throwing.launch gui:=false
    environment:
      - "ROS_MASTER_URI=http://rosmaster:11311"
      - "ROS_HOSTNAME=sim_1"
    networks:
      - ros
  viz_1:
    image: ijs/talos_viz
    command: npm start  --prefix /opt/ijs/gzweb_talos
    environment:
      - "GAZEBO_MASTER_URI=http://sim_1:11345"
    ports:
      - "8080:8080"
    networks:
      - ros
  coordinator:
    image: mathworks/matlab
    volumes:
      - ros_toolbox:/opt/ros_toolbox_matlab
      - database:/home/user/database
    command: matlab -nodisplay -r data_collection.m
```

Acknowledgement: This work has received funding from the program group Automation, robotics, and biocybernetics (P2-0076) supported by the Slovenian Research Agency.

References

[1] J. Peters, J. Kober, K. Muelling, O. Kroemer, and G. Neumann, "Towards robot skill learning: From simple skills to table tennis," in *European Conference on Machine Learning (ECML)*, 2013.

[2] A. Long, A. Blair, and H. van Hoof, "Fast and data efficient reinforcement learning from pixels via non-parametric value approximation," 2022.

[3] J. García, Fern, and o Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[4] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, 2019.

[5] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[6] M. Schmittle, A. Lukina, L. Vacek, J. Das, C. P. Buskirk, S. Rees, J. Sztipanovits, R. Grosu, and V. Kumar, "OpenUAV: A UAV testbed for the CPS and robotics community," in *ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 130–139, 2018.

[7] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154 vol.3, 2004.

[8] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, "Comparing popular simulation environments in the scope of robotics and reinforcement learning," *CoRR*, vol. abs/2103.04616, 2021.

[9] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, and E. Fernández Perdomo, "ros_control: A generic and simple control framework for ros," *The Journal of Open Source Software*, 2017.

[10] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[11] Z. Lončarević, T. Petrič, and A. Gams, "Fitting constrained trajectory with high variability into redundant robot workspace," in *Advances in Service and Industrial Robotics* (A. Müller and M. Brandstötter, eds.), (Cham), pp. 167–175, Springer International Publishing, 2022.