

Joona Leivo

ROBOT MANIPULATOR SIMULATION ENVIRONMENT OVERVIEW

Faculty of Engineering and Natural Sciences
Bachelor's thesis
Supervisor: Roel Pieters
October 2025

ABSTRACT

Joona Leivo: Robot manipulator simulation environment overview
Bachelor's thesis
Tampere University
Science and Engineering
October 2025

This thesis paper examines different simulation software for simulating robots. First, the necessary framework in Robot Operating System (ROS) is introduced. Then different simulation environments were studied via their documentation. Finally, a more hands-on comparison between two environments was conducted to further contrast and compare their capabilities. The result of the research is an overview of the environments studied, their strengths, weaknesses and suitability for the use case of a robot manipulator.

Based on the research the different simulation environments have a lot in common in terms of operating systems and programming languages available. In the other hand, biggest differences were found in types of sensors available and in supported filetypes. Different environments were also found out to have different performance in different tasks, suggesting that the choice for the environment used should be based on the task that is being simulated.

Keywords: Automation, Robotics, Robot Manipulators, Simulation, ROS

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Joona Leivo: Robot manipulator simulation environment overview
Kandidaatin Tutkielma
Tampereen Yliopisto
Teknisten tieteiden kandidaattiohjelma, automaatiotekniikka
Lokakuu 2025

Tämän tutkimuksen tarkoitus oli tutkia ja vertailla erilaisia simulaatio-ohjelmia robottien simuloimistarkoituksiin. Ensin esitellään Robot Operating System (ROS), jota useat simulaatio-ohjelmat hyödyntävät. Sen jälkeen erilaisia ohjelmia vertailtiin tutkimalla niiden dokumentaatiota. Lopuksi kaksi ohjelmaa otettiin syvempään vertailuun niiden kykyjen ja toimivuuden selvittämiseksi. Tutkimuksen lopputulos on yleiskatsaus tutkittuihin simulaatioympäristöihin, niiden vahvuuksiin, heikkouksiin ja sopivuuteen robottikäden simuloimisessa.

Tutkimustuloksista huomattiin, että eri ohjelmat tukivat aika pitkälti samoja käyttöjärjestelmiä ja ohjelmointikieliä. Suurimmat erot taas olivat erityyppisten sensorien saatavuus ja minkälaisia tiedostomuotoja ohjelmat tukivat. Ohjelmilla oli myös eroja suorituskyvyssä erilaisia tehtäviä tehdessä, jonka perusteella simulaatio-ohjelma kannattaa valita sen perusteella mitä tehtävää pitää simuloida.

Avainsanat: Automaatio, Robotiikka, Robotti Manipulaattori, Simulaatio, ROS

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

USE OF AI IN THE THESIS

I have utilized AI tools in my thesis:

Yes

No

TABLE OF CONTENTS

1. INTRODUCTION.....	1
1.1 Research objectives	1
2. ROS OVERVIEW	2
2.1 Robot Operating System	2
2.2 MoveIt	2
3. ENVIRONMENT OVERVIEW	3
3.1 Gazebo	3
3.2 SOFA	3
3.3 MuJoCo	4
3.4 Isaac Sim	4
3.5 CoppeliaSim	4
3.6 Webots.....	5
4. COMPARISON.....	7
4.1 UI	7
4.2 Tutorials	8
4.3 Model & Asset Availability	9
4.4 Physics Engines	9
4.5 GPU Support	10
5. CONCLUSION	11
SOURCES.....	12

TABLE OF IMAGES

<i>Image 1: FrankaEmikaPanda in CoppeliaSim simulation window</i>	7
<i>Image 2: Empty Gazebo World</i>	8
<i>Image 3: Gazebo Demo World[17]</i>	8
<i>Image 4: Screenshot of Fuel[16]</i>	9

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface, connection interface between different computer programs
DART	Dynamic Animation and Robotics Toolbox, open-source library for kinematic and dynamic applications in robotics and computer animation.
DXF	Drawing Exchange Format, filetype for bridging files between computer assisted design programs
GPU	Graphics Processing Unit, specialized circuit for processing computer graphics
GUI	Graphical User Interface, visual interface for interacting with a computer or a program
MuJoCo	Multi-Joint dynamics with Contact, open-source physics engine
MJCF	MuJoCo Format, filetype used for models and scenes in MuJoCo
OBJ	Object File, filetype used for 3d models
ODE	Open Dynamics Engine, open-source physics engine
OS	Operating System
ROS	Robot Operating System, an open-source software framework for robots
SOFA	Simulation Open Framework Architecture, an open-source framework for mechanical simulation
STL	Stereolithography File, filetype used for 3d models
UI	User Interface, way for user to interact with the program or application
URDF	Unified Robot Description Format, xml-based filetype used to describe robots in ROS environments
3DS	File format used by Autodesk for 3d models

1. INTRODUCTION

Automation is used in more and more complex environments, which in turn requires more testing to be done before the application phase. However, testing robots can be expensive, they can wear down or even break. Simulation environments can be used to test and design robots in virtual environments, eliminating the need for a physical robot. [1].

Robot manipulators are industrial robots which are used to move and grasp objects according to their programmed instructions. A manipulator consists of an arm made from multiple linked joints with a total degree of motion depending on the robot's intended purposes and from a rotating wrist usually with some sort of a grabber in at its end. The size of the robot can vary too depending on the purpose and payload, ranging from tiny ones doing fine motor tasks to robots doing industrial heavy lifting. Applications of robot manipulators include tasks such as welding, assembly and picking and placing objects.

1.1 Research objectives

This thesis aims to give an overview on multiple of robot simulation software, exploring their strengths and limitations for simulating a robot manipulator.

2. ROS OVERVIEW

Performing an automated grasp requires multiple functions all of which no single framework can do. This chapter covers ROS and MoveIt which are used in some of the simulation environments.

2.1 Robot Operating System

Robot Operating System, or ROS, is a free and open-source software framework. The fundamentals of ROS implementation are the concept of packages, nodes, messages, topics, and services [2]. ROS is used as the central framework for the simulation, as it serves as a platform for other packages and is responsible for communication between all parts of the system. The concepts of ROS are as follows [6]:

Packages are used to organise software in ROS. They can contain things like nodes, ROS-dependent libraries, datasets or configuration files.

Nodes are the computational units of ROS. A ROS system typically consists of multiple nodes where each performs their own function. Nodes communicate with each other by using messages.

Messages in ROS are simple data structures consisting of typed fields. They support primitive types, such as integers and floating points, and arrays consisting of them. Messages are routed with a publish/subscribe based system via usage of topics.

Topics are used to categorize the content of the message. Messages sent by nodes are published to a topic. Nodes that want to know certain kind of data subscribe to an appropriate topic. Topics can be published to by multiple different nodes. Likewise, nodes can subscribe to multiple topics. However, nodes can't request data through a topic.

Services are used to handle request/reply systems. They work by using a pair of message structures: one for the request and one for the reply. A request node sends message to the service, which in turn receives a reply from the target node and forwards it to the requester.

2.2 MoveIt

MoveIt is an open-source motion planning framework for ROS. It makes use of the default core of ROS build and its messaging systems [3]. MoveIt is installed as a ROS package and thus cannot be used without ROS. It can be used for both simulating virtual robots and manipulating real robots.

MoveIt's role in the simulation is to receive data of the desired pose, calculate how the pose is achieved and move the simulated robot to the pose.

3. ENVIRONMENT OVERVIEW

There are multiple options for physics engines for simulating robots. This chapter aims to give an overview for few of the widely used/popular ones, covering their infrastructure, programming languages and some potential use cases.

3.1 Gazebo

Gazebo is an open-source physics focused robotics simulator. Focused on simulating outdoor environments, its simulator allows the simulation of various real-life conditions, such as gravity, friction, etc. For its default physics engine Gazebo uses Dynamic Animation and Robotics Toolkit (DART). Additionally, Gazebo also offers variety of sensors like camera or GPS for data acquiring purposes. It doesn't itself provide motion planning functionality, but it's integration with ROS allows easy usage of other motion planners, in this paper's case MoveIt. The visualization of Gazebo is handled by OpenGL.

Gazebo environment is essentially a collection of static and dynamic models and sensors. Static models are used for creating the ground and other non-interactable objects, while dynamic models are used for robots and other objects within the simulation. Models consist of three parts, bodies, joints and interfaces. Bodies are the geometric shapes that make up the model. Each body has its own physical & rendering parameters assigned to it such as mass or colour. Joints are what binds the bodies of a model together with kinematic and dynamic relationships. Variety of different kinds of joints are available. Joints can also be used to cause motion. Last part of a model is interface, and it is how user interacts with the model. Interface commands can manipulate joints, change sensor configurations or request sensor data. Gazebo sensors are abstract objects and lack physical representation without tying them to a model.[12]

3.2 SOFA

SOFA (Simulation Open Framework Architecture) is an open-source framework for mechanical simulation, emphasising on biomechanics and robotics. The key concept of SOFA is scenegraph-based multi-model representation. Objects and algorithms in the simulation are described with a hierarchical data structure. The objects themselves are decomposed to smaller independent components, each describing a specific feature, for example mass or constraints. This allows for great modularity easy switching of components when building simulations.[9]

Additionally, simulated objects can be decomposed a set of further specialised models each optimized for different types of computation. Typically, objects are described using three models: an internal model for computing degrees of freedom and the mass and

constitutive laws, a collision model for contact geometry and a visual model for detailed geometry and rendering parameters.

Originally created for modelling and simulating in medical and surgical applications, the SOFA engine particularly excels in computing soft and rigid body dynamics, making a great fit for simulating soft robotics.

3.3 MuJoCo

MuJoCo (Multi-Joint dynamics with Contact) is an open-source general purpose physics engine originally developed by Roboti LLC. MuJoCo is a C/C++ library with C API. Models are defined in MJCF scene description language native to MuJoCo. It's an XML file format designed to be easily human readable and editable. In addition, MuJoCo also supports URDF model files. The library also offers interactive visualization with a GUI, rendered by OpenGL.[7]

Within the simulation a built-in compiler into an optimized data structure used for runtime computation. The models used can have tendon wrapping and/or actuator activation states, like muscles or pneumatic cylinders. MuJoCo is capable of computing both forward and inverse kinematics.[13]

Some of MuJoCo's key features include soft, convex & analytically-invertible contact dynamics, tendon geometry, model compilation, reconfigurable computation pipeline and the forementioned intuitive modelling language.

3.4 Isaac Sim

Developed by NVIDIA, Isaac Sim is a reference application built on NVIDIA's own Omniverse platform. Isaac Sim scenes are built using OpenUSD, but it also supports URDF/MJCF imports via an open-source importer.[8] Isaac Sim uses a high-fidelity GPU-based PhysX as its physics engine. Isaac Sim can also directly access the GPU, allowing the platform to simulate various kinds of sensors. Isaac sim also has bridge API to ROS and ROS2. The simulators workflow supports C++ and Python by offering VS Code and Jupyter Notebook extensions.

3.5 CoppeliaSim

CoppeliaSim (formerly Virtual Robot Experimentation Platform or V-REP) is a simulation framework aiming to be as versatile and scalable as possible. This is achieved by supporting multiple programming approaches and languages, multiple different physics engines and cross-platform portability. The programming techniques that can be used in CoppeliaSim to control a model or a scene are embedded scripts, add-ons, plug-ins, remote API clients and ROS nodes. The user doesn't have to pick one, all approaches can be used simultaneously.

Embedded scripts are a way to modify the simulation scene or models via scripting in Lua or Python. Doing changes with scripts guarantees compatibility with other default CoppeliaSim installations. Embedded scripts also act as the binding component between

other programming approaches, as they can register ROS publishers/subscribers, launch executables, load/unload plug-ins, or start remote API server services.

Add-ons are Python or Lua scripts that are automatically loaded on startup, allowing extension of CoppeliaSim's functionality with user written functions. Add-ons persist on all opened scenes and are executed constantly.

Plug-ins are shared libraries that can be loaded in on demand. They are most often used for hardware interfacing or importing/exporting. Remote API and ROS nodes are implemented with plug-ins.

Remote API clients allow communicating with an external entity (i.e. an application running in a different process, or on a different machine). The API interface is composed of remote API server services and remote API clients. The client side of can be programmed in C/C++, Python, Java, JavaScript, Matlab or Octave.

ROS nodes are implemented with a plug-in which enables ROS to call CoppeliaSim commands, data streaming to via ROS publishers & subscribers. Enabling of publishers/subscribers can be done within CoppeliaSim via an embedded script command.

CoppeliaSim doesn't have its own physics engine, but rather it allows you to pick and switch between five different ones depending on one's needs. The five supported physics engines are the Bullet physics library, the Open Dynamics Engine, the MuJoCo physics engine, the Vortex Studio engine and the Newton Dynamics engine.[11]

3.6 Webots

Webots is an open-source professional robot simulation software package for mobile robots. Multiple different locomotion schemes are supported, such as wheeled, legged or flying robots. Webots also supports variety of sensors and actuators within the simulation. Webots also has interfaces to real mobile robots, making porting the control system to a real robot easy after making it behave the wanted way within the simulator. [10]

A Webots simulation consists of three key parts: a Webots world file, one or more controller programs and any optional plugins. The world file defines the robots and their environment. Worlds are organized as hierarchical structures where objects can contain sub-objects. The worlds are saved as proprietary ".wbt" files. The programs are responsible for the control and movement of the robots. They can be programmed with C, C++, Java, Python or MATLAB. Multiple robots can use the same controller code, but each will run their own distinct instance of it. Optional plugins can be used to modify the behaviour of the simulation's physics.

	ROS	Ga-zebo	SOFA	Mu-JoCo	Isaac Sim	Coppeli-aSim	Webots
OS	Linux	Linux	Windows, MacOS, Linux	Windows, Linux and macOS	Windows, Linux	Windows, MacOS, Linux	Windows, MacOS, Linux
Programming language	C++, Python	C++, Python	C++, Python (via plugins)	C/C++	Python, C++	Python, C/C++, Lua, MatLab, Java	C/C++, Python, Java, MATLAB
GUI	No	Yes	Yes, with plugin	Yes	Yes	Yes	Yes
Sensors	Lidar, Radar, Point cloud	Contact, Lidar	Force	Touch, Accelerometer, joint-pos, joint-vel	RTX Lidar, Contact, Lightbeam	Proximity, Vision, Force	Lidar, TouchSensor, Accelerometer
3d model/world file-types	URDF	OBJ, STL, Collada	OBJ, STL	URDF, MJCF	URDF, MJCF	OBJ, DXF, STL, 3DS, Collada, URDF	WBT
Open-source	Yes	Yes	Yes	Yes	Yes (Partial?)	No	Yes
Pricing	Free	Free	Free	Free	Free	Free educational version	Free

Table 1. Table of the simulation environments

4. COMPARISON

Each simulation program comes with their own advantages and disadvantages as listed in the previous chapter. This chapter compares performance and features of a few of them. For the scope of the paper CoppeliaSim and Gazebo were picked for comparison. The software was tested on a desktop PC running Ubuntu 22.04.4. Versions of the software used were Edu V.4.7.0 for CoppeliaSim and Fortress 6.16.0 for Gazebo respectively.

4.1 UI

CoppeliaSim's UI consists of dropdown menus. Various simulation parameters such as the physics engine used can be changed with a few clicks of the mouse. Models can be easily dragged and dropped to the world from the model browser window.

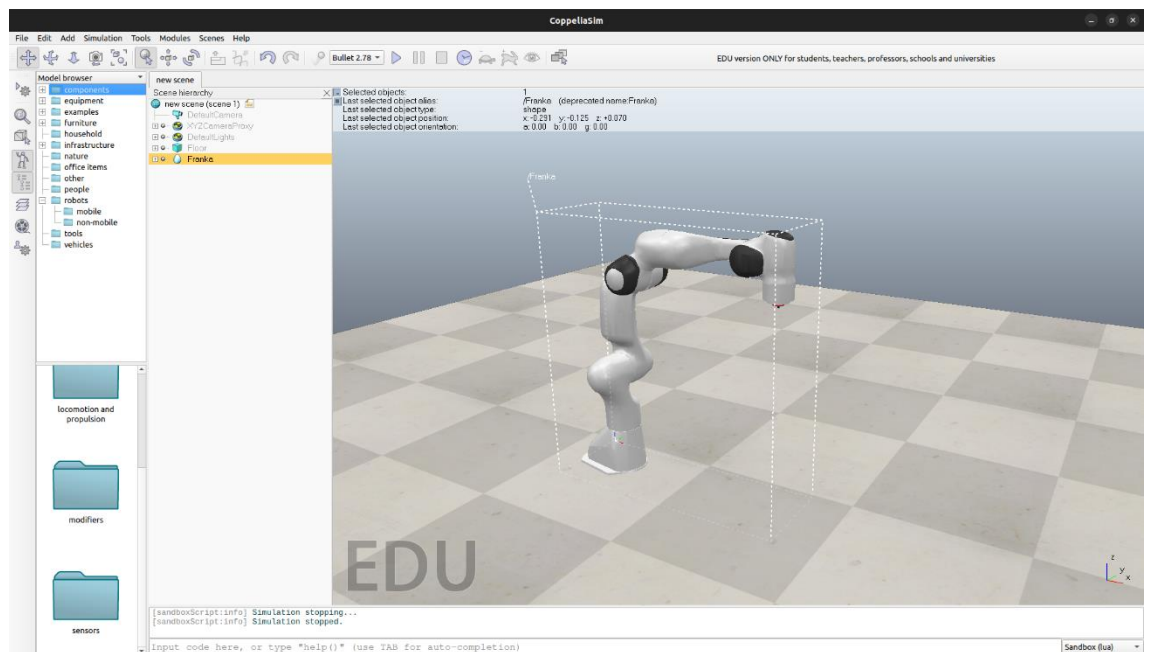


Image 1: FrankaEmikaPanda in CoppeliaSim simulation window

Gazebo UI has the world and entity data displayed on the right side. The quick select menu on the top-left has model spawners for simple shapes and tools for selecting and manipulating entities. Lot of features and tools are hidden behind a search bar accessible by clicking on the three dots icon.

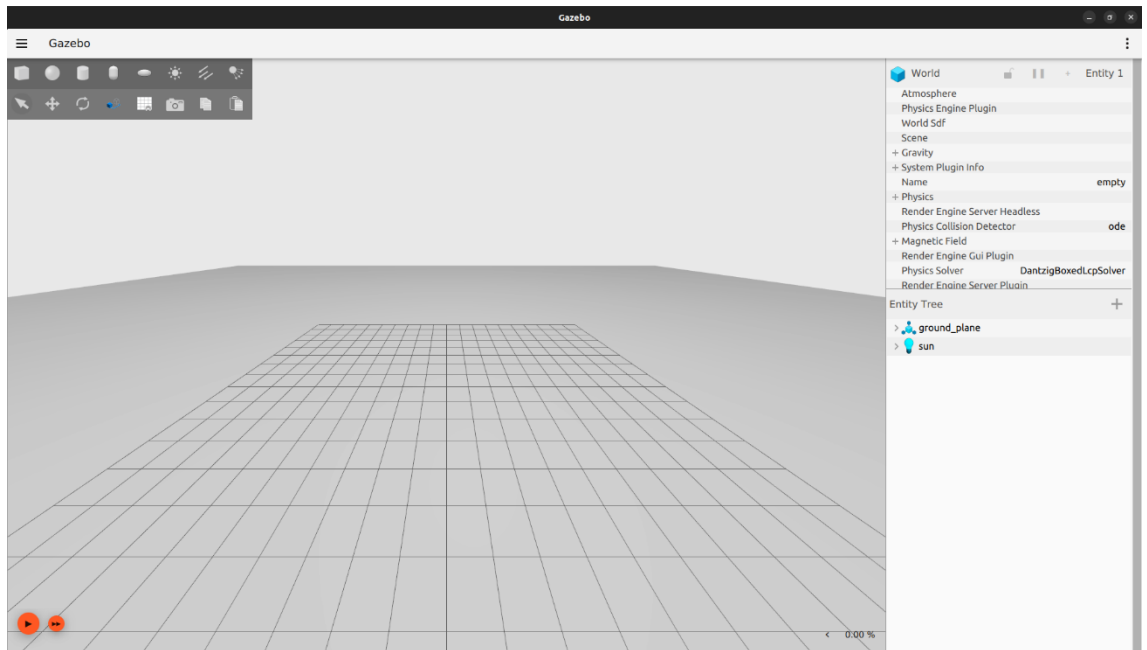


Image 2: Empty Gazebo World

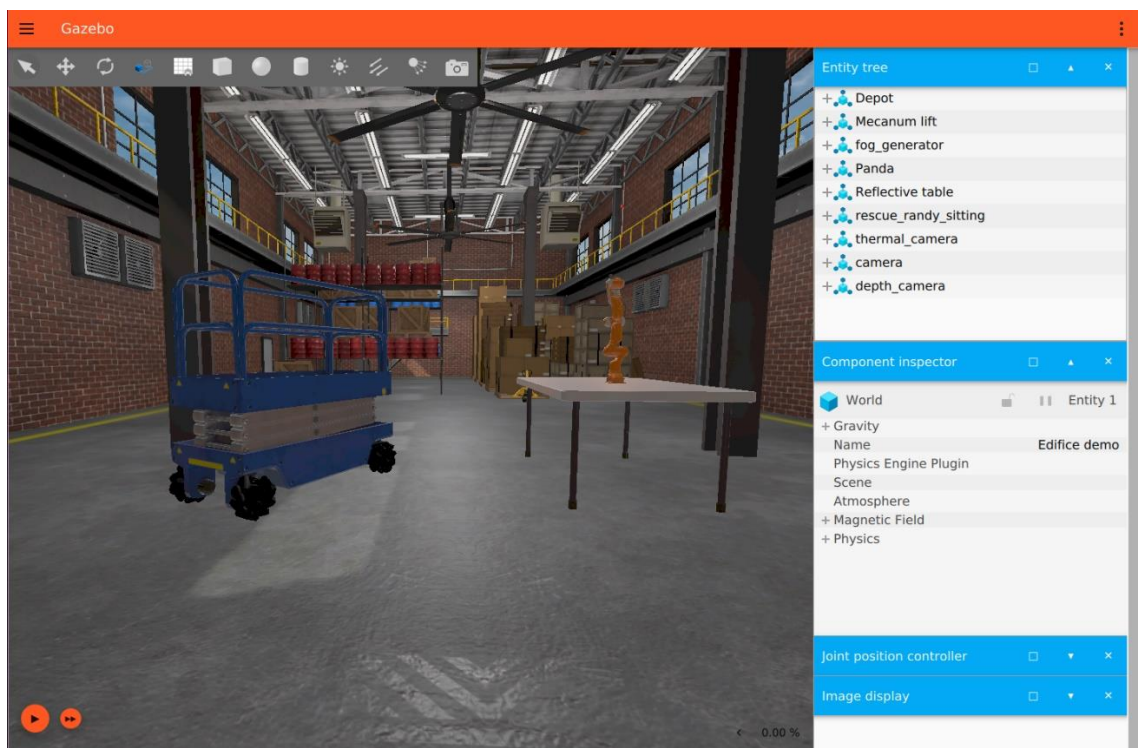


Image 3: Gazebo Demo World[17]

4.2 Tutorials

CoppeliaSim installation comes with an inverse kinematics tutorial. Instructions can be found in CoppeliaSim's online user manual[14]. The tutorial walks you through the whole process from building a simple simulation model to setting up the inverse kinematics task to finally testing the inverse kinematics. In general, the user manual and tutorials are comprehensive and easy to follow.

Gazebo also ships with an example worlds that demonstrate some of the simulator's features. The online tutorials have detailed how-to's on Gazebo's features and help for customizing various configurations, for example the GUI.[4]

4.3 Model & Asset Availability

Gazebo has extensive model support via Fuel, a web application for hosting and accessing simulation assets.[15] Fuel is currently hosting over 3000 user created and simulation ready models, ranging from mobile robots & manipulators to humans and environmental objects. Models can be easily installed within Gazebo's GUI.

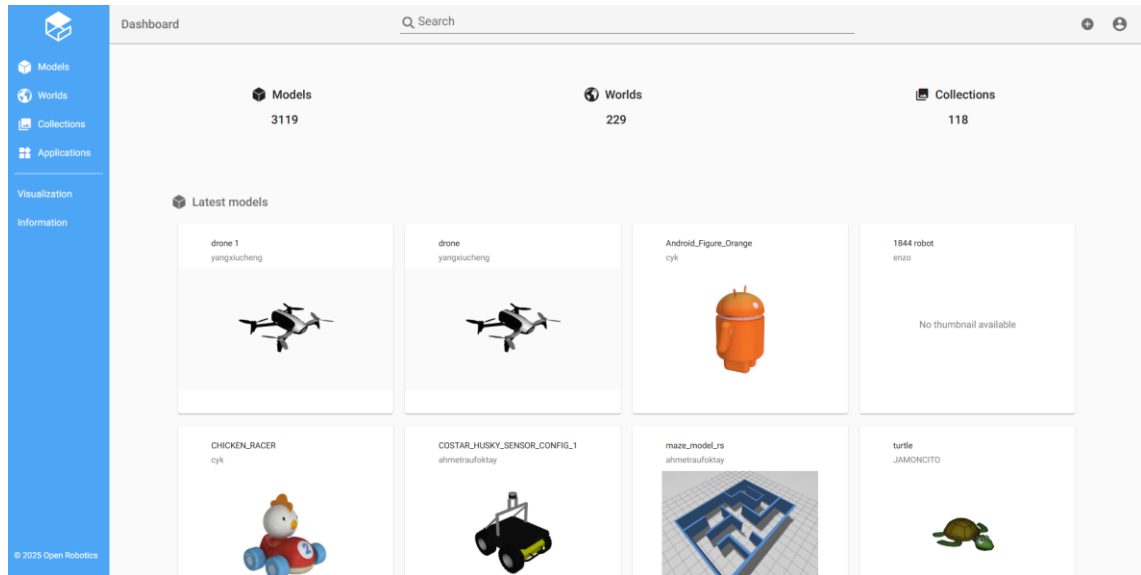


Image 4: Screenshot of Fuel[16]

CoppeliaSim's model support is more limited. Some ready-to-import models exist but not nearly to the level of support as Gazebo. However, as CoppeliaSim supports OBJ, DXF, STL, 3DS, Collada and URDF files, finding and importing models from other online sources is feasible.

4.4 Physics Engines

As mentioned before, CoppeliaSim offers choice between five different physics engines (Bullet, ODE, MuJoCo, Vortex Studio and Newton Dynamics), whereas Gazebo has interfaces for four engines (Bullet, ODE, Simbody, DART). As different engines have different CPU loads, speed and accuracy of results depending on the task performed, having more to choose from is better. For example, in benchmarking done by Erez et al., MuJoCo has found to be most accurate and fastest on a grasping task but being slow in term of CPU speed when simulating systems of multiple disconnected bodies[5]. Different engines have also limitations on what kind of shapes they can use for simulated objects in collision calculations.

	Bullet	ODE	MuJoCo	Vortex Studio	Newton Dynamics	Simbody	DART
Collision primitives	sphere, box, cylinder, capsule, cone, concave triangle mesh, convex, compound	sphere, box, cylinder, capsule, plane, ray, triangular mesh, convex	plane, sphere, capsule, cylinder, ellipsoid, box, geoms	boxes, spheres, capsules, planes, and cylinders, triangle mesh	N/A	ellipsoid, half-space, sphere, triangle mesh	box, cylinder, ellipsoid concave mesh, probabilistic voxel grid
Rigid body dynamics	Yes	Yes	Yes	Yes	N/A	Yes	Yes
Soft body dynamics	Yes	Yes	Yes	Yes	N/A	Yes	Yes
Gravity	Yes	Yes	Yes	Yes	N/A	Yes	Yes

Table 2. Summary of physics engine differences.

4.5 GPU Support

While some simulators can offload some of the calculations for processing the simulation from the CPU to the GPU, both Gazebo & CoppeliaSim only use the GPU for rendering and vision-based sensors. A third different simulator could have been picked for comparing the effects of GPU computing.

5. CONCLUSION

This thesis aimed to compare different environments for simulating a robot manipulator. Information about different environments, their capabilities and differences were gathered. Afterwards, two environments (CoppeliaSim and Gazebo) were taken to more in-depth comparison where their user experience, physics engines and model support were compared to each other.

The analysis revealed lot of similarities between different simulation environments. One key similarity being that different environments support most of the same programming languages. Every platform supports C++, with every platform apart from MuJoCo also supporting Python. With the exception of CoppeliaSim, every simulator is also free and open-source. Biggest differences between the platform came from sensors present in the environments and filetypes supported for model and world files.

In conclusion, since there are lots of similarities in the different environments in the supported OS's and programming languages, the biggest factors for choosing the environment come down to available sensors, models and planned use case(s). For example, Webots is specialised in mobile robotics and SOFA in soft robotics, whereas MuJoCo is more generalised in its applications. Different environments were also found to have different performance in terms of both accuracy and speed, depending on the given task in case-by-case basis, furthermore emphasising the importance of the planned use case.

SOURCES

- [1] J. Collins, S. Chand, A. Vanderkop and D. Howard, "A Review of Physics Simulators for Robotic Applications," in *IEEE Access*, vol. 9, pp. 51416-51431, 2021, doi: 10.1109/ACCESS.2021.3068769
- [2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics* vol. 7, May 2022. doi: 10.1126/scirobotics.abm6074
- [3] D. Coleman, I. Şucan, S. Chitta and N. Correll, Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study, *Journal of Software Engineering for Robotics*, 5(1):3–16, May 2014. doi: [10.6092/JOSER_2014_05_01_p3](https://doi.org/10.6092/JOSER_2014_05_01_p3).
- [4] Gazebo tutorials. URL: <https://gazebo.org/api/gazebo/3/tutorials.html> (visited on 17.10.2025)
- [5] T. Erez, Y. Tassa and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, 2015, pp. 4397-4404, doi: 10.1109/ICRA.2015.7139807.
- [6] ROS Concepts. URL: <http://wiki.ros.org/ROS/Concepts> (visited on 10.10.2025)
- [7] MuJoCo documentation. URL: <https://mujoco.readthedocs.io/en/stable/overview.html> (visited on 18.10.2025)
- [8] NVIDIA Isaac sim landing page. URL: <https://developer.nvidia.com/isaac/sim> (visited on 17.10.2025)
- [9] Faure, F. et al. (2012). SOFA: A Multi-Model Framework for Interactive Physical Simulation. In: Payan, Y. (eds) *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*. Studies in Mechanobiology, Tissue Engineering and Biomaterials, vol 11. Springer, Berlin, Heidelberg. https://doi.org/10.1007/8415_2012_125
- [10] Webots introduction and documentation. URL: <https://cyberbotics.com/doc/guide/introduction-to-webots> (visited on 17.10.2025)
- [11] E. Rohmer, S. P. N. Singh and M. Freese, "V-REP: A versatile and scalable robot simulation framework," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, 2013, pp. 1321-1326, doi: 10.1109/IROS.2013.6696520.

- [12] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Sendai, Japan, 2004, pp. 2149-2154 vol.3, doi: 10.1109/IROS.2004.1389727.
- [13] E. Todorov, T. Erez and Y. Tassa, "MuJoCo: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, 2012, pp. 5026-5033, doi: 10.1109/IROS.2012.6386109.
- [14] CoppeliaSim inverse kinematics tutorial. URL: <https://manual.coppeliarobotics.com/en/inverseKinematicsTutorial.htm> (visited on)
- [15] Gazebo Fuel documentation. URL: <https://gazebosim.org/docs/ionic/fuel/> (visited on 19.10.2025)
- [16] Gazebo Fuel. URL: <https://app.gazebosim.org/dashboard> (visited on 20.10.2025)
- [17] Gazebo Edifice demo. URL: <https://app.gazebosim.org/OpenRobotics/fuel/worlds/Edifice%20demo> (visited on 20.10.2025)