How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion

Andrew Farley[a], Jie Wang[a,*], Joshua A. Marshall[a,b]

[a] *Ingenuity Labs Research Institute, Queen's University, 69 Union St W, Kingston, K7L 3N6, ON, Canada*
[b] *Department of Electrical & Computer Engineering, Queen's University, 19 Union St W, Kingston, K7L 3N6, ON, Canada*

Abstract

The number of available tools for dynamic simulation of robots has been growing rapidly in recent years. However, to the best of our knowledge, there are very few reported quantitative comparisons of the most widely-used robot simulation tools. This article attempts to partly fill this gap by providing quantitative and objective comparisons of four widely-used simulation packages for mobile robots. The comparisons reported here were conducted by obtaining data from a real Husky A200 mobile robot driving on mixed terrains as ground truth and by simulating a 3D mobile robot model in a developed identical simulation world of these terrains for each simulator. We then compared the simulation outputs with real, measured results by weighted metrics. Based on our experiments and selected metrics, we conclude that CoppeliaSim is currently the best performing simulator, although Gazebo is not far behind and is a good alternative.

Keywords: Robot simulation, Quantitative analysis, Robot software, Robot Operating System (ROS)

*Corresponding author

Email addresses: 16amf8@queensu.ca (Andrew Farley), jwangjie@outlook.com (Jie Wang), joshua.marshall@queensu.ca (Joshua A. Marshall)

1. Introduction

The use of computer-based simulations is good practice in the process of developing new robot designs and algorithms, prior to building and executing code on physical robotic systems. As robots and associated software become more dynamically capable and complex, simulation tools are increasingly being used to emulate the realistic motion of 3D robot models in a range of virtual environments. At the time of writing and since the onset of the COVID-19 pandemic, mobile robots in particular are being widely used for disinfection, as well as logistics and delivery across the world [1]. However, due to safety issues associated with the pandemic, mobile robotics researchers and practitioners have also found it increasingly difficult to conduct laboratory and field testing on real robots. Many have turned to simulation. Thus, selecting a suitable robot simulation tool is of timely importance.

The number of dynamics simulation software tools for robots has been growing in the past decade [2, 3] alongside advances in computing power and lower costs. However, a comparison of these simulators is challenging because different simulators have different qualitative features and may need to be tailored to solve custom problems. Most were not built as general-purpose tools [4]. There are plenty of criteria to consider when selecting a robot simulator, and researchers and practitioners with different needs and objectives use different metrics. In a user feedback survey of roboticists mainly working on control and mobile robots [4], accuracy (i.e., measurably close to reality), open-sourced, and the ability to employ the same code for both real and simulated robots, were cited as the most important criteria for choosing a simulator. Of noted secondary importance was the quality of the visualization and the ensuing computational load.

The accuracy of a simulator is mainly determined by its underlying physics engine. Robots interact with the environment via contact forces, thus accurate calculations of contact dynamics by physics engines are crucial for physically-accurate simulations. However, in the reviewed literature, accuracy is often compared relatively among different physics engines rather than absolutely to a ground-truth data set [5, 6]. In [5], numerical integration errors from mathematical solvers rather than actual model errors are plotted as a speed-accuracy curve for physics engine accuracy estimations. In a recent study [6], a benchmark result of contact dynamics simulation on state-of-the-art physics engines is provided. No real physics knowledge obtained from experiments was used for benchmark comparison. They used

analytical solutions of object trajectory as ground-truth references obtained from Newton's rigid body dynamics and Coulomb's friction cone model for systems with a single rigid body and small number of contacts. For complex systems, kinetic energy or linear momentum was used to calculate the object trajectory references. Both errors integrated from mathematical solvers and errors deviate to physics model calculations were used as indications of the physics engine's accuracy.

There is an obvious benefit if one can employ the same code for both real and simulated robots, which points to the importance of a seamless interface between virtual and real robots. In this paper we define a Robot Operating System (ROS) [7] compatibility metric to evaluate this functionality. Similar to OpenAI Gym, for example, compatibility is an essential feature for "sim2real" of reinforcement learning research [8], ROS compatibility is important in the goal of testing and validating navigation, control, and planning strategies by using simulations. Finally, the visualization criterion highlights the need for providing a simultaneous visual result of conducted simulation experiments, which can reduce the effort to create and debug newly developed algorithms.

The primary objective of this article is to help mobile roboticists to select a suitable simulator for their application and development needs. Four popular robot simulators, namely CoppeliaSim (formerly called V-REP) [9], Gazebo [10], MORSE [11] and Webots [12], shown in Figure 1, were selected because they are widely and actively used in robotics community [2, 4, 13, 14]. Clearly, without quantitative comparisons, it proves difficult to select the best one for a given project. Moreover, as opposed to many of the available dynamics simulation software are physics engines [5], these four are system-level simulators. System-level simulators are built on physics engines to simulate contact dynamics between the robots and the environment. But, in addition, they provide tools for sensor simulation, robot model and simulation world editors, and visualization user interface (UI) functionalities. A comparison of physics engines alone (e.g., see [5, 6]) was not the objective of this study.

In this study, we categorized comparison criteria as simulator qualitative features and quantitative metrics to evaluate the simulator accuracy in Section 2. Instead of merely showing the relative accuracy of physics engines among simulators, we compared simulated inertial measurement unit (IMU) data and data obtained on a real Husky A200 mobile robot [15] to evaluate the absolute overall accuracy of simulators in Section 3. Finally, a weighted quantitative comparison of both simulator qualitative features

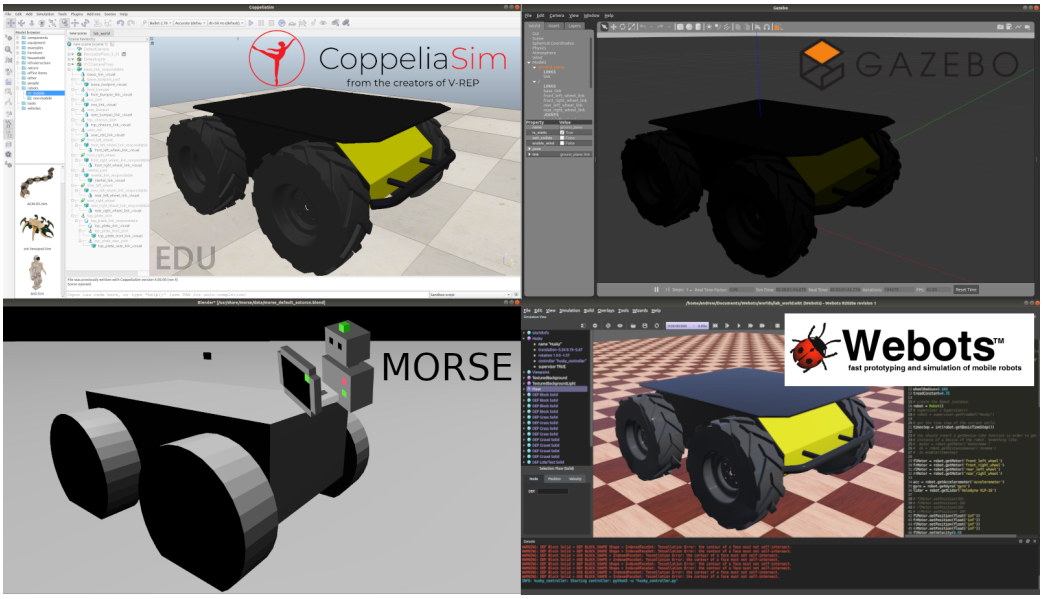

Figure 1: The user interface windows of CoppeliaSim 4.0.0, Gazebo 9.0.0, MORSE 1.4 and Webots R2020b. A 3D model of Husky A200 robot was created in the default simulation world of each simulator respectively.

and quantitative metrics to characterize accessibility, as well as speed and accuracy of simulators is presented in Section 4. Conclusions and summary discussions are in Section 5. The implementation steps of the simulator comparisons in this paper have been made available online at `https://github.com/offroad-robotics/Robot-Simulator-Comparison`.

2. Selection of Evaluation Metrics

One could consider a large number of potential criteria in the process of selecting a mobile robot simulator. We categorized these possible criteria as: 1) *qualitative features*; and, 2) *quantitative metrics*, as listed in the first column of Table 1 and Table 2, respectively. Qualitative features are selected key features of a simulator, and quantitative metrics focus on simulation accuracy and computational load. Each selected metric is signed with a metric type. Boolean metrics receive a score of 1 if the simulator possesses the given quality and 0 if it does not. String metrics are a list of supported qualities in a given category. Real valued metrics range from 0 to 1 and their evaluation depends on specific rules, as defined for each metric. The

Table 1: Selected simulator qualitative features, including metric types.

Metric Name	**Description**	**Metric Type**
Free to Use	Is the software free or paid for academic purposes?	Boolean
Open Source	Is the software maintained by an online community?	Boolean
ROS Compatibility	Compatibility and ease of use with ROS	Real
Programming Languages	List of supported programming languages	String
UI Functionality	Simplicity and utility of the UI	Real
Model Format Support	List of supported model formats	String
Physics Engine Support	List of supported physics engines	String

score values of simulator qualitative features and quantitative metrics were determined by experiments that are explained in Section 3.

Four simulators—CoppeliaSim[1], Gazebo[2], MORSE[3] and Webots[4]—were selected for comparison. These four robot simulators are all ROS compatible as well as widely and actively used in the robotics research community. They are all system-level simulators that provide high-level 3D modeling and visualization rather than mere physics engines. **CoppeliaSim**, formerly named V-REP, has powerful simulator supports for multiple physics engines including Bullet [16], ODE [17], Vortex [18] and Newton [19]. It comes installed with an extensive list of example worlds, robots, and sensors. It does not have native ROS support but provides an official plugin allowing the use of it. It supports programming languages including C/C++, Python, Java, Lua,

[1]https://www.coppeliarobotics.com
[2]http://gazebosim.org/
[3]https://www.openrobots.org/morse/doc/stable/morse.html
[4]https://cyberbotics.com/

Table 2: Selected quantitative metrics, including metric types.

Metric Name	Description	Metric Type
Real Time Factor	The average real time factor of all data associated with a simulator	Real
Average Load CPU Efficiency	CPU load during an average simulation run	Real
Intense Load CPU Efficiency	CPU load during a particularly intense simulation	Real
IMU Accuracy	Accuracy of IMU data output for angular velocity and linear acceleration	Real

MATLAB, and Octave.

Gazebo was created by the Open Source Robotics Foundation, which also created ROS. Gazebo comes installed with a full ROS install and thus has developed a widespread following. Gazebo supports multiple physics engines including ODE, Bullet, DART [20], and Simbody [21]. Particularly useful is the fact that it comes with prepared examples of robot models and environment worlds. The Gazebo community is active to creating new robot models and environment worlds. For example, the simulation assets[5] including 240 models of mobile robots and subterranean environments were created during DARPA's SubT Challenge Virtual Competition. Outdoor agriculture, industrial inspection, and indoor construction environment worlds[6] were recently released by Clearpath Robotics.

MORSE is an open-source simulator built on top of the Blender game engine [22]. Blender is a 3D game engine created on top of the Bullet physics engine. It comes with a couple of examples but is relatively limited when compared to the two aforementioned simulators. ROS support comes installed with MORSE. MORSE sets up simulation components (worlds, robot models and sensors) entirely through Python, thus it has a very limited UI.

Webots is also widely used by the mobile robotics community for indus-

[5]https://www.darpa.mil/news-events/2020-02-07

[6]https://clearpathrobotics.com/blog/2020/07/clearpath-robots-get-new-gazebo-simulation-environments

trial and academic research projects. Webots has a resourceful library that includes robots, sensors, actuators, objects and materials. Like the others, it has built-in ROS support. However, the only supported physics engine is ODE. Robots can be programmed in C, C++, Python, Java, MATLAB. Webots uses Proto nodes to represent real world objects, it also provides an official plugin to import existing CAD models in URDF format [23].

3. Simulator Evaluation Experiments

In this study, we conducted quantitative comparisons of the four selected simulators by way of a set of test trials devised to elucidate the metrics defined in Section 2. The test virtual environments for each simulator, which we refer to as "worlds" in Section 3.1, were created to replicate as closely as possible an actual laboratory environment that was used to capture real baseline data. The simulator qualitative features listed in Table 1 evaluation results were obtained after completing the setup of each virtual environment and summarized in Table 3. Scores assigned to each qualitative features based on the rules defined in Section 3.2 were summarized in Table 4. Then, each simulator was evaluated on the basis of simulation runs aimed at generating the quantitative metrics listed in Table 2. For instance, the CPU efficiency was evaluated by two simulation scenarios with different computational loads. Moreover, the overall accuracy of the simulators was studied by comparing the simulated output versus real IMU sensor data obtained in the parallel laboratory experiments. The scores for each metric were generated and used in a summary weighted metrics evaluation, presented in Section 4.

3.1. Simulation World Setup

This section describes the creation of the virtual worlds, provides a discussion about the selection of physics engines, as well as the controller used in both physical and simulation experiments.

3.1.1. Virtual Worlds

Each virtual test world was created to simulate the indoor open laboratory environment Ingenuity Labs Research Institute[7] at Queen's University. Three terrain pieces, namely artificial grass, bumps, and gravel, as shown in Figure 2, were used to create uneven ground profiles within the laboratory

[7]https://ingenuitylabs.queensu.ca/lab-facilities/

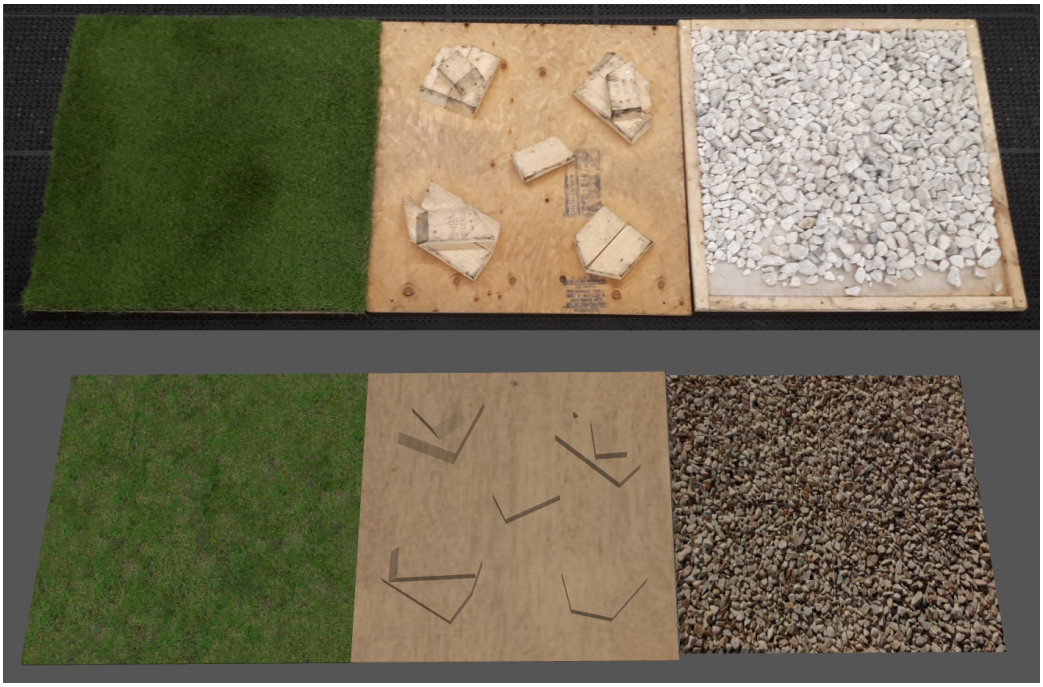

Figure 2: Three terrain pieces were used to create uneven ground profiles within the indoor laboratory environment and simulation worlds. Upper: Left to right are the "grass", "bumps", and "gravel" pieces of the actual laboratory terrain, respectively. Lower: The corresponding grass, bumps, and gravel pieces created in the simulation worlds.

environment. The corresponding identical terrain pieces were created in each simulator as shown in Figure 2. Each piece was created to simulate its corresponding real-world equivalent as accurately as possible by tuning respective friction coefficient numbers. The friction coefficient of the bumps piece, the artificial grass piece, and the gravel piece were given coefficient values of 0.95, 0.2 and 0.55 respectively. These friction coefficient values were obtained from [24, 25].

A suitably equipped real Clearpath Husky A200 mobile ground robot was used to collect baseline data by moving the Husky around in the laboratory environment shown in Figure 3. A corresponding 3D model of the Husky robot in a simulated virtual representation of the physical laboratory was created by using each simulator tool, as illustrated in Figure 4. In both the real lab and simulated environment, the robot was commanded to run tests with straight lines and circular paths shown in Figure 3. The input speed of the straight path was 0.5 m/s, the input angular velocity of the circular

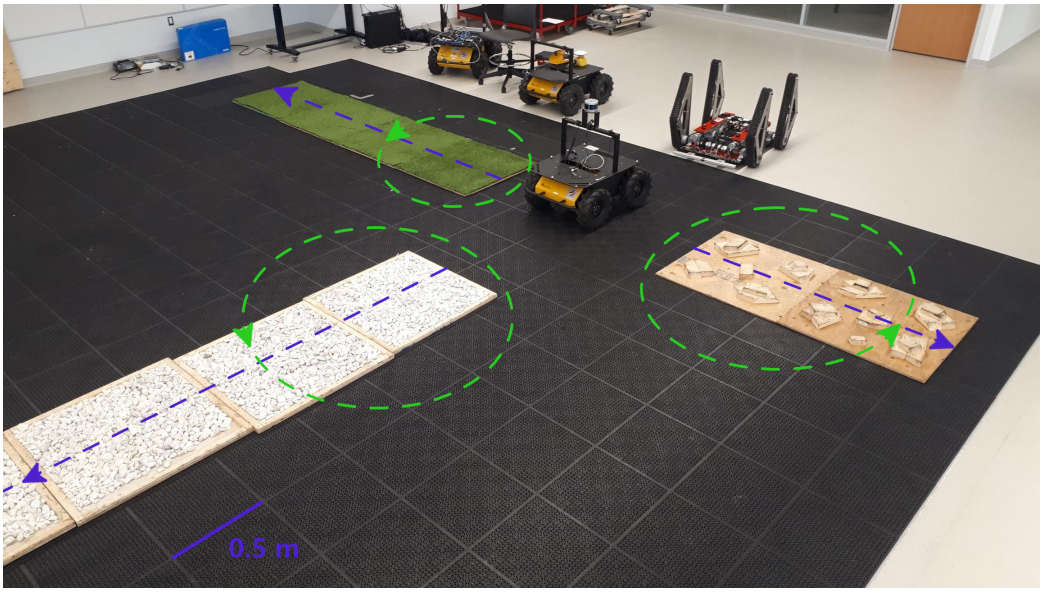

Figure 3: A Husky A200 mobile robot in the indoor laboratory environment at the Ingenuity Labs Research Institute at Queen's University. The IMU data in this lab environment and identical simulation world in four selected simulators was collected by controlling the robot to move straight and turn over three terrain profile pieces. Note that the Husky A200 configuration here is for visualization purpose only, the robot configuration used in the experiments is shown in Figure 5.

path was 0.5 rad/s.

3.1.2. Physics Engines

Each system-level robot simulator is built on top of a physics engine. For CoppeliaSim and Gazebo, users are able to select one of four physics engines. We tested each of the engines with our specific 3D model of Husky A200 shown in Figure 5, and selected one for further evaluation by running a series of basic motion tests with straight lines and circular paths. The purpose of this paper is not to quantitatively compare physics engines, which can be found in [5, 6]. Instead, we picked a suitable physics engine by eliminating those that tended to generate obvious glitches or other visibly undesirable outputs in simulations, as described in sequence. These undesirable outputs included jumping in velocity or position, unrealistic behaviours during collisions with objects, and erratic sensor output. The physics engine with the least number of observed problems was selected for the remaining experiments.

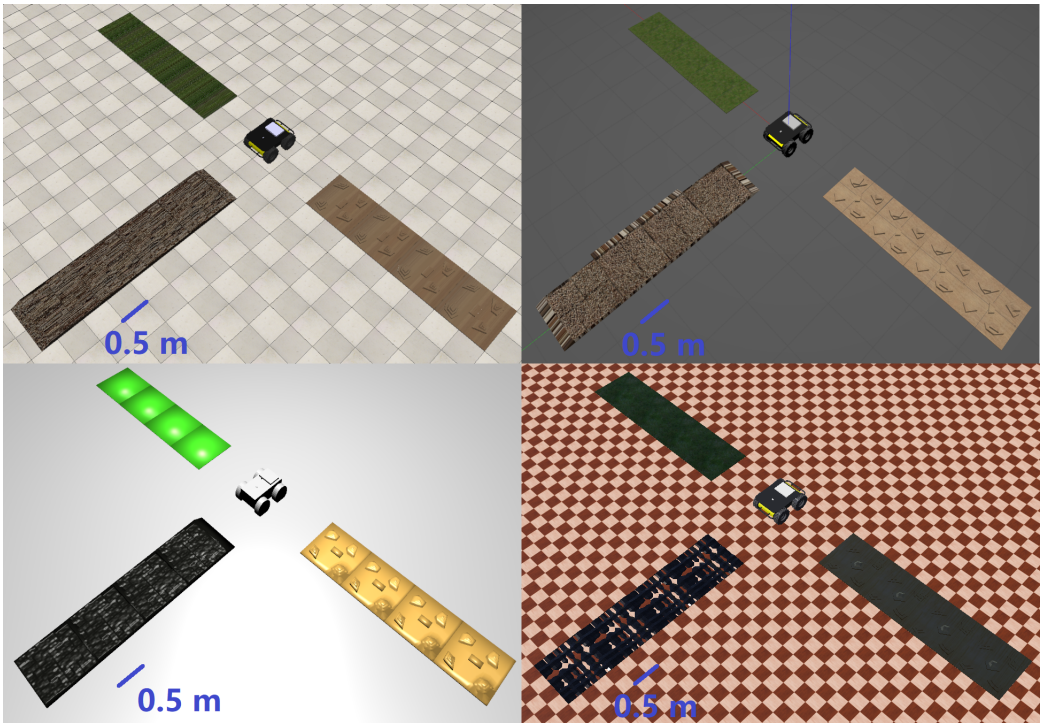

Figure 4: A 3D model of the Husky A200 robot in an identical simulation world of the lab in each simulator. The upper left is CoppeliaSim, upper right is Gazebo, lower left is MORSE, and lower right is Webots.

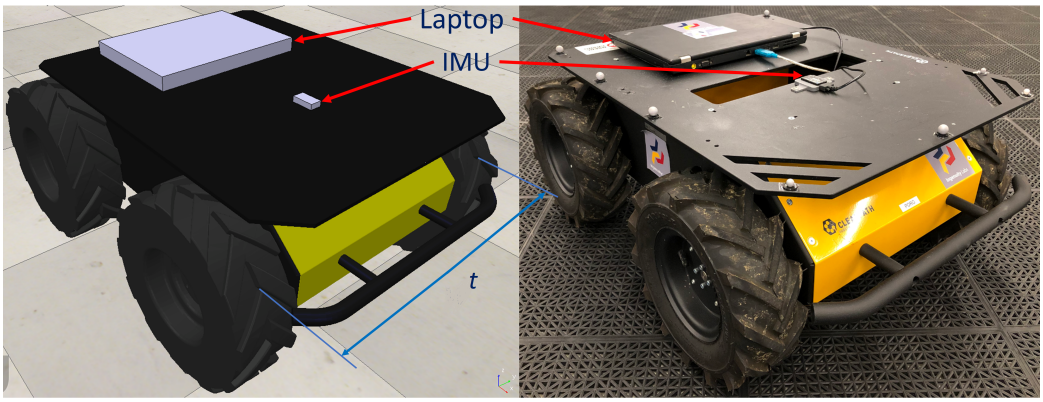

Figure 5: A 3D model of the Husky A200 robot (left) was created to replicate as closely as possible an actual Husky A200 mobile robot (right). An IMU sensor and laptop were mounted on the robot to collect the linear acceleration and angular velocity data.

CoppeliaSim supports Bullet 2.78, Bullet 2.83, ODE, Vortex, and Newton. Unfortunately, we were not able to get Vortex to work correctly; the robot often fell through the floor and was not able to move. ODE struggled when the robot was turning, generating erratic jumps in velocity and the robot was jittery when colliding with objects. Newton suffered from similar problems as ODE. Bullet 2.83 performed well when the robot was moving straight but not when the robot turned. While turning, the robot would jump in position and not move in a circle. The robot also tended to slide while sitting still. Bullet 2.78 had small issues when turning the robot and produced the least number of issues, such that it was selected as the physics engine of choice for CoppeliaSim. Gazebo supports ODE, Bullet, DART, and SimBody. After testing, the simulators performed similarly with some key differences. DART and SimBody had jumps in position and velocity while turning and the robot would occasionally slide when sitting still. ODE and Bullet had similar issues with turning, but no sliding issues. Because ODE is the default physics engine for Gazebo, users likely use it more often and it performed as well as Bullet and better than DART and SimBody. Hence, ODE was selected as the physics engine for Gazebo in subsequent virtual world experiments.

3.1.3. Motion Controller

A baseline vehicle controller was used to actuate the Husky A200 robot for all experiments. Specifically, the ROS differential drive skid-steer controller described in [26] was used in Gazebo and MORSE. In Coppeliasim and Webots, this standard ROS version of the controller was not compatible with the corresponding Husky model in these simulators, thus a custom differential drive controller was written to be as identical as possible to the standard ROS controller. Based on the recommendations provided in the Husky A200 user manual [15], the left and right wheel velocities were calculated by a kinematic approximation as

$$v_l = v - \omega\frac{\tau}{2} \tag{1a}$$

$$v_r = v + \omega\frac{\tau}{2}, \tag{1b}$$

where v_l and v_r represent the left and right wheel velocities, respectively. The speeds v and ω are the instantaneous longitudinal speed and the instantaneous rotational speed of the vehicle, respectively. τ is the effective track

Table 3: Summary of the simulator qualitative features evaluation results, after completing the setup of each simulation environment.

Metric Name	**CoppeliaSim**	**Gazebo**	**MORSE**	**Webots**
Free to use	True	True	True	True
Open source	False	True	True	True
ROS Compatibility	A built-in plugin provided	Out of the box	Out of the box	A built-in plugin provided
Programming languages	C/C++, Python, Lua, MATLAB, Java, Octave	C/C++, Python	Python	C/C++, Python, Java, MATLAB
UI functionality	Full functionality	Full functionality	Visualization only	Full functionality
Model format support	URDF, SDF, Stl, Obj, Dxf, Collada	URDF, SDF, Stl, Obj, Collada	Blend	Proto Nodes
Physics engine support	Bullet, ODE, Vortex, Newton	Bullet, ODE, DART, Simbody	Bullet	ODE

of the vehicle [15], which is the length between the geometrical width centers of the left and right sides of wheels and equals to 0.555 m as shown in Figure 5.

3.2. Simulator Qualitative Features

The selected simulator qualitative features listed in Table 1 were used to evaluate the qualitative features of each simulator with the simulation worlds described in Section 3.1. The qualitative features evaluation results are summarized in Table 3. For results presented in this article, a score was assigned to each qualitative feature based on the following rules.

Free to use. If a simulator is free to use it received a score of 1, otherwise 0.

Open source. If a simulator is open-sourced it received a score of 1, otherwise 0.

ROS compatibility. If a simulator has ROS support on installation, it received a score of 1. If a simulator has an official plugin for ROS that can be installed manually, it received a score of 0.8. Otherwise, it received a score of 0.

Programming languages. If a simulator supports programming languages C/C++, Python, MATLAB and more, it received a score of 1. If a simulator supports C/C++ and Python, it received a score of 0.667. If a simulator supports C/C++ or Python, it received a score of 0.333. Otherwise, it received a score of 0.

UI functionality. By "full control" of a simulator we mean the ability to start, stop, and step the simulation, visualize the models in the scene, as well as add models in at any given moment. If a simulator has a UI allowing users full control over the simulation process, it received a score of 1. If a simulator has a UI allowing a subset of these controls, it received a score of 0.667. If a simulator can only visualize what happens in a scene, it received a score of 0.333. If a simulator has no UI then it received a score of 0.

Model format support. The Universal Robot Description Format (URDF) and Simulation Description Format (SDF) are arguably the two most widely used representation formats for robotics simulations [23]. If a simulator supports model formats URDF, SDF and more, it received a score of 1. If a simulator supports URDF or SDF, it received a score of 0.5. If a simulator has a plugin for URDF or SDF that needs to be installed manually it received a score of 0.25. Otherwise, the simulator received a score of 0.

Physics engine support. This metric evaluates support for multiple physics engines rather than the accuracy of the physics engines themselves. Multiple physics engines available in a simulator is beneficial because users can select one that suits their particular application(s). Quick verification of simulation projects can also be conducted by switching physics engines. If a simulator supports more than one physics engine, it received a score of 1. If a simulator supports only one physics engine, it received a score of 0.

Table 4: Summary of the simulator qualitative features scores.

Metric Name	**CoppeliaSim**	**Gazebo**	**MORSE**	**Webots**
Free to use	1.000	1.000	1.000	1.000
Open source	0	1.000	1.000	1.000
ROS compatibility	0.800	1.000	1.000	0.800
Programming languages	1.000	0.667	0.333	1.000
UI functionality	1.000	1.000	0.333	1.000
Model format support	1.000	1.000	0	0.250
Physics engine support	1.000	1.000	0	0

The simulator qualitative features score of each simulator were summarized in Table 4.

3.3. Quantitative Metrics

Multiple tests of the quantitative metrics in Table 2 were conducted by running a 3D Husky robot model in the virtual world shown in Figure 4. The CPU efficiency evaluations included average load and intense load tests during simulations. The ground-truth IMU data included the linear acceleration and angular velocity by running a Husky A200 robot in the laboratory environment. Then the simulated IMU data obtained by running the 3D robot model in an identical simulated world were compared to the real IMU data respectively for each simulator.

In the current work, we evaluate the proprioceptive accuracy of the vehicle's motion in a simulator. More specifically, we focused on evaluating how well the vehicle's velocities or accelerations were simulated rather than position and orientation estimations of a vehicle. For that we use IMU data collected on a real mobile robot (Husky A200), which is one of the most reliable interoceptive sensor measurements, as ground-truth data for an absolute quantitative comparison. Indeed, the exteroceptive sensors play paramount roles nowadays in mobile robot applications, while controllers trained in simulators by using interoceptive sensor measurements only provided impressive results [27]. The quantitative evaluation of the simulators incorporating dense 3D sensors is a promising future work to provide more generalizable

Table 5: Specifications of the IMU sensor and computers.

IMU	**Specification**
LORD Microstrain 3DM-GX3-25	Gyroscope range = ± 300°/s Accelerometer range = ± 5g Sampling Rate = 100 Hz
Computer 1	**Specification**
ThinkPad T410s	CPU: Intel Core i5-520m @ 2.40 GHz RAM: 4 GB DDR8 OS: Ubuntu 18.04 GPU: Intel HD Graphics Card
Computer 2	**Specification**
Custom Desktop	CPU: Intel Core i7-9700k @ 3.60 GHz RAM: 32 GB DDR4 OS: Ubuntu 18.04 GPU: MSI Radeon RX 580 6 GB

comparison results.

3.3.1. Hardware Platform

A Husky A200 mobile robot [15] was used as the mobile robotic platform to perform all experiments in the indoor laboratory environment shown in Figure 3. The configuration of the robot is shown in Figure 5. A LORD Microstrain 3DM-GX3-25 IMU and Thinkpad T410s laptop were mounted on the robot to collect the linear acceleration and angular velocity data. A custom-built desktop was used to conduct simulator comparison experiments and data processing. The specifications of the IMU sensor and computers were summarized in Table 5.

3.3.2. Real Time Factor

The real time factor is a commonly used metric to measure the computational load of a simulator [13]. The real time factor was calculated by taking the ratio of the sum of the simulated time step to the sum of the desired real time step. There were six ratios associated with six tests (straight lines and circular paths on grass, bumps, and gravel) shown in Figure 3 for each simulator. These six ratios were summed to calculate the real time factor for

a given simulator as

$$R_i = \frac{SimTime}{RealTime} = \frac{\sum_{n=1}^{6} \sum_{j=1}^{j} \Delta t_{n,j}^{s}}{\sum_{n=1}^{6} \sum_{k=1}^{k} \Delta t_{n,k}^{r}} \tag{2}$$

where R_i presents the real time factor for a given simulator i, n is the motion test number shown in Figure 3, $\Delta t_{n,j}^{s}$ is the simulator time step j of test n in seconds, and $\Delta t_{n,k}^{r}$ represents the real time step k of test n in seconds. The calculated results of each simulator were listed in Table 10. In order to include the real time factor metric to the overall weighted metrics evaluation of simulators, the real time factor scores were calculated with the obtained real time factor by

$$F_i = \frac{1}{R_i} \min_{i \in \{1,2,3,4\}} R_i, \tag{3}$$

where R_i is the calculated real time factor of CoppeliaSim, Gazebo, MORSE or Webots listed in Table 10. The real time factor scores were summarized in the third to sixth column of Table 11.

3.3.3. CPU Efficiency

Each simulator has a varying computational load. In all selected simulators, the CPU is used by physics engines for simulating rigid body dynamics as well as collision and contact forces. GPU acceleration is not supported by any of physics engines in the simulators, therefore only the CPU is evaluated for the computational load metric. The CPU efficiency was tested by two sub-test scenarios with different computational loads for each simulator.

Scenario 1. A virtual Husky robot moved around in the virtual world, over and around terrain features. This provided an estimate of how resource-intensive the application was under an average computational load.

Scenario 2. This was a stress test involving five Husky robots moving around simultaneously in the virtual world.

The results of Scenarios 1 and 2 are reported as the Average Load CPU Efficiency and Intense Load CPU Efficiency metric in Table 2, respectively.

The measured CPU loads for each simulator were recorded during each of the above mentioned scenarios. Each simulator updates at a rate of between 50-60 Hz for each of these tests. Because each simulator had a similar update

rate for a given CPU load scenario, the load value corresponds to a measure of simulator efficiency. The results of Scenarios 1 and 2 are summarized in Table 10. These values were gathered from the Ubuntu System Monitor when only the simulators were running.

We used ε_i to represent either the average or intense load CPU efficiency of simulator i, and the CPU efficiency was computed by

$$\varepsilon_i = \frac{1}{L_i} \times \min_{i \in \{1,2,3,4\}} L_i, \tag{4}$$

where i is a given simulator, and L_i represents either the average or intense load CPU efficiency for a given simulator listed in Table 10. The CPU efficiency scores are listed in the third to sixth column of Table 11.

3.3.4. IMU Accuracy

The Husky robot moved straight and turned over each terrain profile pieces in the lab environment and the identical simulation world in each simulator shown in Figure 3 and 4, respectively. The collected data sets were the linear accelerations and angular velocities in the X, Y, and Z axis of the IMU sensor. The linear acceleration and angular velocity data collected on the real Husky A200 robot were used as the ground-truth to evaluate the IMU simulation accuracy of each simulator. Due to the complexity of the IMU data collected in different experiments, we proposed a method that uses the mean and range of the IMU data to provide a whole comparison of all data collected in different experiments. We will explain the data processing process by an example that uses CoppeliaSim. The example data was collected in the CoppeliaSim and indoor lab environment showing the angular velocities and linear accelerations of the robot while it moves straight over gravel terrain pieces shown in Figure 6.

The collected data were first processed by calculating the mean and range of the X, Y, and Z axes by using

$$\mu_d = \frac{\sum_{k=1}^{n} D_{d,k}}{n} \tag{5}$$

$$\delta_d = \max D_d - \min D_d, \tag{6}$$

where μ_d is the mean for a given axis of a plot, δ_d is the range of a given axis of a plot, D_d is the set of data for a given axis in a plot, $D_{d,k}$ is a data point in a given axis for a given plot, and n represents the number of the data point

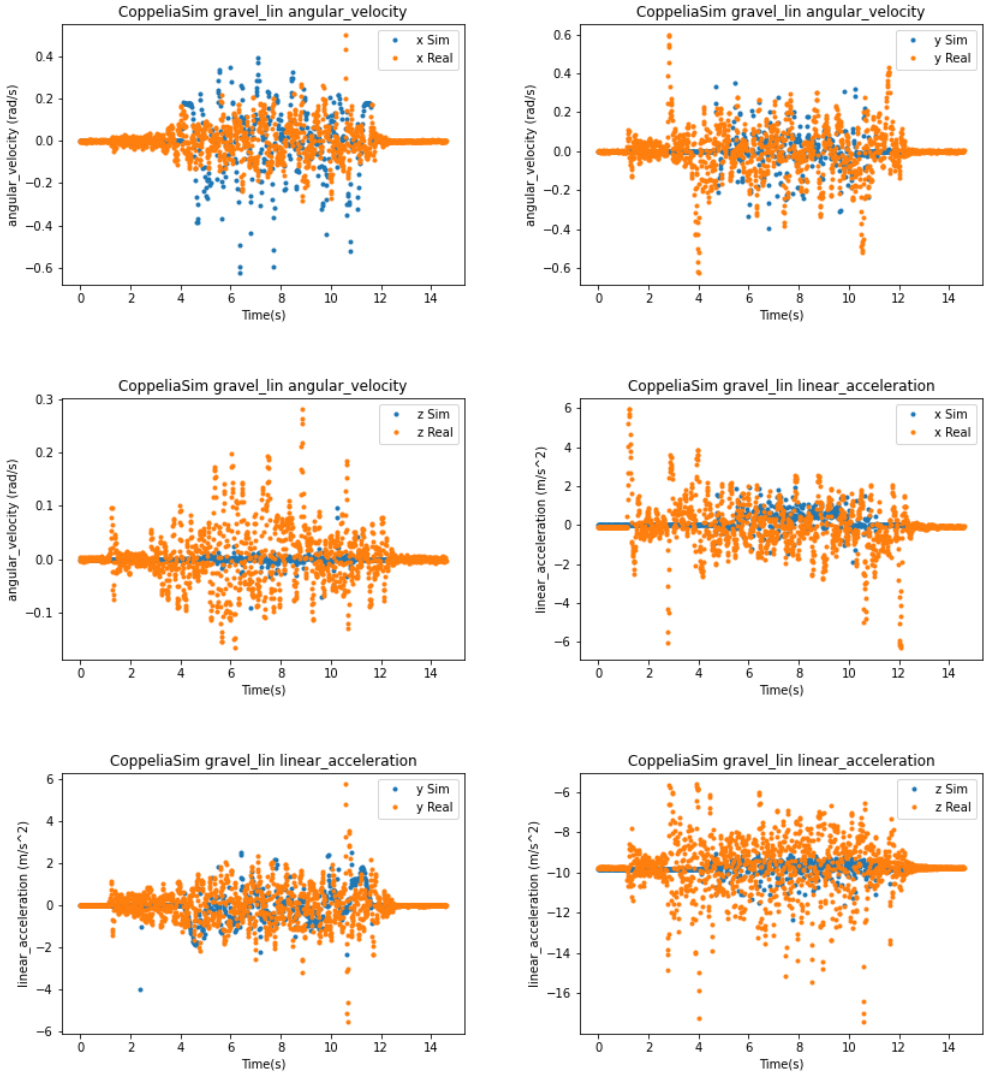

Figure 6: An example plot of the IMU data. The data was collected in the CoppeliaSim simulator (Sim Data) and indoor lab environment (Real Data) showing the angular velocities and linear accelerations of the robot while it moves straight over gravel terrain pieces. The first row left, right and the second row left are the angular velocities of the X, Y, and Z axis respectively, the second row right, the third row left and right are the linear accelerations of the X, Y, and Z axis respectively.

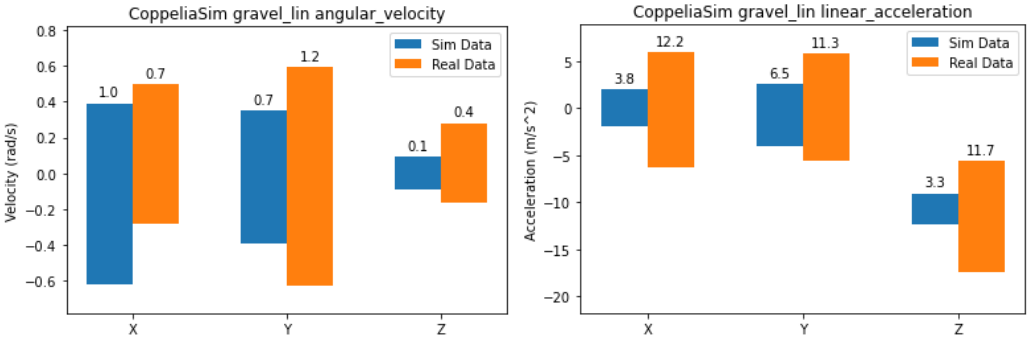

Figure 7: An example plot of the IMU data processing. The data was collected in the CoppeliaSim simulator (Sim Data) and indoor lab environment (Real Data) showing the angular velocities and linear accelerations of the robot while it moves straight over gravel terrain pieces. The collected data were plotted to show the mean and range of X, Y, and Z axis of IMU. Each bar shows the maximum and minimum values sampled and the range of the data. The exact range number was indicated on the top of each bar. Left: angular velocities. Right: linear accelerations.

in a given axis for a given plot. The example data was processed and labeled as "Sim Data" and "Real Data" and plotted as shown in Figure 7, which shows the processed data of X, Y, and Z axis of IMU. Each bar shows the maximum and minimum values sampled and the range of the collected data, as well as the total range values on top of each bar. The maximum value was determined as the average of the set of values in the top one percent. The minimum value was determined as the average of the set of values in the bottom one percent. The data collected by the real world sensors as well as the simulated ones both occasionally gave erratic measurements that did not line up with what the robot was doing. This is why this averaging was done. The range and mean values of the angular velocities and linear accelerations example data were summarized in Table 6 and 7 respectively.

To evaluate the overall performance of a simulator, we summed the absolute difference between the sim and real data collected in all six motion tests (Figure 3) as the total errors in the X, Y, or Z axis as

$$E_j = \sum_{n=1}^{6} \left(|R^s_{j,d} - R^r_{j,d}| + |M^s_{j,d} - M^r_{j,d}| \right), \tag{7}$$

where E_j presents the total error of either the linear acceleration or the angular velocity, $R^s_{j,d}$ and $R^r_{j,d}$ are the range of sim data and real data of the

Table 6: The Angular Velocities Processing Example of IMU Data.

Axis	**Simulated Range**	**Real Range**	**Simulated Mean**	**Real Mean**
X	1.000 rad/s	0.780 rad/s	-0.120 rad/s	0.110 rad/s
Y	0.740 rad/s	1.200 rad/s	-0.020 rad/s	-0.010 rad/s
Z	0.190 rad/s	0.450 rad/s	0.003 rad/s	0.060 rad/s

Table 7: The Linear Accelerations Processing Example of IMU Data.

Axis	**Simulated Range**	**Real Range**	**Simulated Mean**	**Real Mean**
X	3.90 m/s^2	12.20 m/s^2	0.02 m/s^2	-0.18 m/s^2
Y	6.50 m/s^2	11.30 m/s^2	-0.73 m/s^2	0.10 m/s^2
Z	3.30 m/s^2	11.80 m/s^2	-10.70 m/s^2	-11.50 m/s^2

linear acceleration or the angular velocity in the X, Y, or Z axis dimension respectively, $M^s_{j,d}$ and $M^r_{j,d}$ are the mean of sim data and real data of the linear acceleration or the angular velocity in the X, Y, or Z axis dimension respectively, and n is the motion test number shown in Figure 3. The calculation results of Table 6 and 7 by Equation 7 were summarized in Table 8 and 8 respectively. In Table 8, the total error in the X, Y, or Z axis can be summed as the final error for the angular velocities going straight over gravel pieces of CoppeliaSim simulator, which is 1.237 rad/s. In Table 9, the total error in the X, Y, or Z axis can be summed as the final error for the linear accelerations is 23.19 m/s^2.

Above processes were conducted to all angular velocity scenarios (grass, bumps, and gravel) and each total errors was summed providing the total angular velocity error of CoppeliaSim. The same processes were done for linear acceleration scenarios of CoppeliaSim. The processed sim data of CoppeliaSim together with the actual data were plotted in Figure 8. The same data processing steps applied on the example data were repeated for Gazebo, MORSE, and Webots, then the results were plotted in Figure 8. The final total error of the angular velocity and linear acceleration of CoppeliaSim,

Table 8: The Final Errors of the Angular Velocities Processing Example.

Axis	Range Error	Mean Error	Total Error
X	0.220 rad/s	0.230 rad/s	0.450 rad/s
Y	0.460 rad/s	0.010 rad/s	0.470 rad/s
Z	0.260 rad/s	0.057 rad/s	0.317 rad/s

Table 9: The Final Errors of the Linear Accelerations Processing Example.

Axis	Range Error	Mean Error	Total Error
X	8.30 m/s^2	0.16 m/s^2	8.46 m/s^2
Y	4.80 m/s^2	0.83 m/s^2	5.63 m/s^2
Z	8.50 m/s^2	0.80 m/s^2	9.30 m/s^2

Gazebo, MORSE, and Webots were summarized in Table 10.

Similar to other metrics (i.e., larger metric values imply better performance) the accuracy metric scores of the IMU angular velocity and linear acceleration were calculated with the obtained IMU errors by

$$A_i = \frac{1}{E_i} \min_{i \in \{1,2,3,4\}} E_i, \tag{8}$$

where E_i is the total final errors of the linear acceleration or the angular velocity of CoppeliaSim, Gazebo, MORSE or Webots listed in Table 10. The IMU accuracy metric scores were summarized in the third to sixth column of Table 11.

3.4. Bugs, Errors, and Troubleshooting

Before conducting the final comparison by the weighted simulator qualitative features and quantitative metrics, this section presents bugs and errors that had to be troubleshooted for each simulator, and the solutions we used to address these issues in order to carry out out useful tests.

CoppeliaSim had the fewest number of recorded errors and bugs compared to the other three simulators. The Vortex physics engine did not recognize the Husky model as a solid mesh leading to the robot falling through

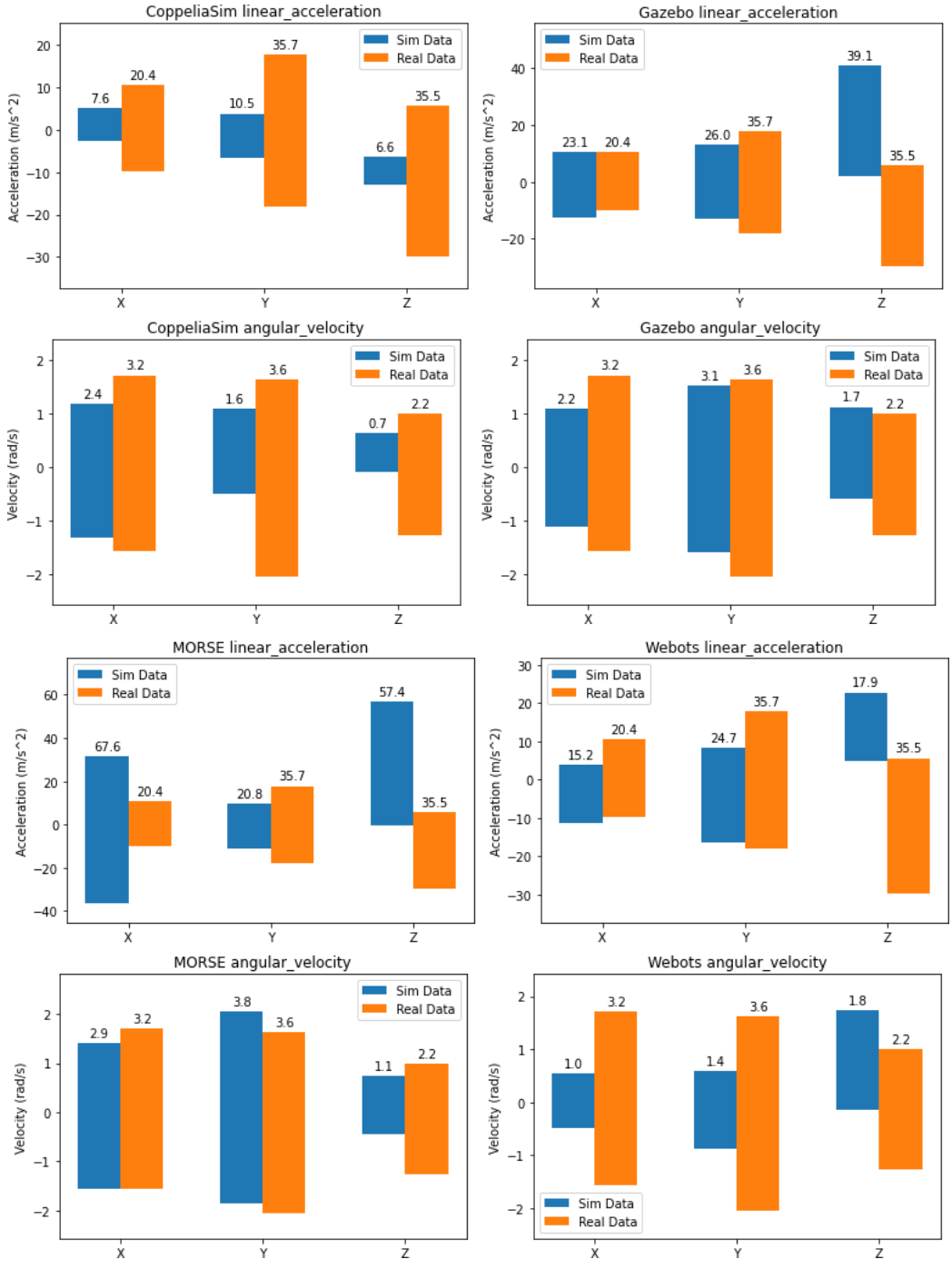
CoppeliaSim linear_acceleration
Acceleration (m/s^2)
Sim Data
Real Data
7.6
20.4
10.5
35.7
6.6
35.5
X
Y
Z
Gazebo linear_acceleration
Acceleration (m/s^2)
Sim Data
Real Data
23.1
20.4
26.0
35.7
39.1
35.5
X
Y
Z
CoppeliaSim angular_velocity
Velocity (rad/s)
Sim Data
Real Data
2.4
3.2
1.6
3.6
0.7
2.2
X
Y
Z
Gazebo angular_velocity
Velocity (rad/s)
Sim Data
Real Data
2.2
3.2
3.1
3.6
1.7
2.2
X
Y
Z
MORSE linear_acceleration
Acceleration (m/s^2)
Sim Data
Real Data
67.6
20.4
20.8
35.7
57.4
35.5
X
Y
Z
Webots linear_acceleration
Acceleration (m/s^2)
Sim Data
Real Data
15.2
20.4
24.7
35.7
17.9
35.5
X
Y
Z
MORSE angular_velocity
Velocity (rad/s)
Sim Data
Real Data
2.9
3.2
3.8
3.6
1.1
2.2
X
Y
Z
Webots angular_velocity
Velocity (rad/s)
Sim Data
Real Data
1.0
3.2
1.4
3.6
1.8
2.2
X
Y
Z

Figure 8: The total accumulated angular velocity and linear acceleration error of all conducted motion tests (straight move on and turn over grass, bumps and gravel) in simulators and the indoor lab environment. The collected data were plotted to show the mean and range of X, Y, and Z axis of IMU. Each bar shows the maximum and minimum values sampled and the range of the data. The exact range number was indicated on the top of each bar. The first row left and right are the linear acceleration error of CoppeliaSim and Gazebo respectively, the second row left and right are the angular velocity error of CoppeliaSim and Gazebo respectively, The third row left and right are the linear acceleration error of MORSE and Webots respectively, and the fourth row left and right are the angular velocity error of MORSE and Webots respectively.

Table 10: Summary of the quantitative metrics evaluation results.

Metric Name	**CoppeliaSim**	**Gazebo**	**MORSE**	**Webots**
Real Time Factor	0.973	1.064	0.839	0.903
Average Load CPU Efficiency	11 %	23 %	12 %	4 %
Intense Load CPU Efficiency	12 %	23 %	12 %	7 %
IMU Angular Velocity Error	21.46 rad/s	18.76 rad/s	24.42 rad/s	22.30 rad/s
IMU Linear Acceleration Error	247.36 m/s^2	340.39 m/s^2	624.23 m/s^2	359.66 m/s^2

the ground. The standard ROS differential drive skid-steer controller could not be used with the Husky model, so a custom differential drive script was created.

Gazebo had significant issues with turning the robot. The Husky model used a skid-steer method of turning and the asymmetric friction on the wheel caused the robot to jump in velocity and position. This could be smoothed out by setting the coefficient of asymmetric friction (μ_2), however this would cause errors in the radius of the turn as well as the speed of the turn. It was decided that having as little error in the radius and speed of the turn was more important than the jumpy behaviors of the standard Husky model, so μ_2 was left at its standard value of 1 for data collection.

MORSE had a significant number of issues during robot model creation. To get the required Blender format file of the Husky robot model, several ROS and Blender tools were required. A ROS tool, `collada_urdf`, was used to create Collada files from the Husky's URDF files. The generated files were then imported into Blender for physics setup and then saved. However, the Blender model had over 400,000 vertices that MORSE could not handle, thus a simplified 3D Husky robot model was created with around 400 vertices. The lack of detail of the robot model may have contributed to the issues that MORSE exhibited in the simulation experiments.

Webots had multiple problems with experimental data processing. The global frame of reference ($X - Y - Z$) of the other three simulators is a right-handed East-North-Up (ENU) coordinate, while Webots use a East-Up-South (EUS) coordinate as the global frame of reference [28]. Thus, the experimental data required extra processing steps. Similar to CoppeliaSim, a custom differential drive script was needed to control the robot. Despite the fact that there was no difference between the control script for CoppeliaSim and that used for Webots, Webots displayed strange behaviors at certain speeds. The robot sometimes moved in an elliptical pattern.

4. Weighted Metrics Evaluation

Overall comparisons of the simulators were quantitative and objectively evaluated by weighted simulator qualitative features and quantitative metrics scores. We assigned each metric a weight ranging from a value of 1 to 10 to quantify its importance. The weights were selected to correspond with the users' feedback in the survey [4], although we recognize that selecting

specific weights is ultimately a subjective process. The readers can custom these weights to meet their specific evaluation needs.

Free to use (weight of 4). If a simulator is free to use it is more accessible to average users. This is good in itself and can also lead to the availability of more community support. Community support can be vital when dealing with unexpected problems and learning how the simulator works.

Open source (weight of 2). If a simulator is open source then the source code is freely available. This means that it can be easier to debug issues and this also allows our robotics community to add new features. Open source software often grows to support many features, however it can sometimes be unreliable or inefficient because it may not be written by programming professionals.

ROS integration (weight of 6). ROS is a vital part of each simulator because it is a widely-used robotics middle-ware. We feel that it is most useful if a simulator has ROS support available on installation or through a plugin.

Programming Languages (weight of 3). The more programming languages supported, the more accessible a simulator may be to a broad user base.

UI functionality (weight of 6). Every simulator needs to be able to run simulations in a user-defined environment. A powerful UI allows for easy creation of simulations by drag and drop controls on models. A good UI can also provide live data in a more visual way, which can help researchers to understand and debug simulation runs.

Model format support (weight of 4). Creating reliable robot models and simulation worlds can be time-consuming. If a simulator supports the most widely used file formats including URDF and STL, or via a plugin, this allows for faster prototyping and reliable development.

Physics engine support (weight of 3). A simulator that supports multiple physics engines is beneficial because this different physics engines may be more or less suitable for specific applications. Quick verification (and sanity checks) might also be conducted by switching physics engines.

Real time factor (weight of 4). Real time factor is a good measure of how efficient a simulator is regardless of the hardware it is being run on. This has been shown in [14]. Due to its use and accuracy in simulator efficiency, it has been given a weight of 4.

CPU efficiency (weight of 2). For simulators, CPU demand often dictates how closely a simulation might be run to real-time. While it is more efficient to have less CPU usage, this factor may not be as important as others because CPU demand should not affect the simulator outputs.

IMU accuracy (weight of 10). The accuracy of the data collected from simulated IMU indicates the overall accuracy of a simulator. If the values are significantly different from comparable real-world data, the simulator is less useful for research and development purposes.

The weighted metrics evaluation result for each simulator is shown in Table 11. Based on these metrics and our tests, we found CoppeliaSim to rank the best overall. Moreover, it was also our favorite simulator to set up and use for simulation-based experiments. The UI of CoppeliaSim is the most powerful among the four selected simulators, CoppeliaSim is easy to get working with ROS (using a plugin), and it has more included robots and sensor models than any other simulator. It also supports the largest number of physics engines and is the easiest to switch between physics engines. It supports many model types and is relatively easy to convert model files into one of the types that CoppeliaSim supports. In our rankings, Gazebo was ranked almost equal to CoppeliaSim. Gazebo is a good alternative for researchers who do not want to use CoppeliaSim or are unable to for some reason. Although Webots was the most computational efficient simulator, it did not score well in the IMU accuracy metric, which puts it at a significant disadvantage. Thus, it may only be a good choice if computational resources are at a premium. Finally, MORSE did not score well on many of our metrics.

Besides the comprehensive and detailed metrics evaluation results shown in Table 11, a decision tree comprising the most distinctive features among four simulators is created shown in Figure 9. By giving answers to several questions, the decision tree can tell users which is the best fit simulator quickly.

Table 11: Weighted metrics value evaluation results.

Metric Name	**Weight**	**CoppeliaSim**	**Gazebo**	**MORSE**	**Webots**
Free To Use	4	1.000	1.000	1.000	1.000
Open Source	2	0	1.000	1.000	1.000
ROS Integration	6	0.800	1.000	1.000	0.800
Programming Languages	3	1.000	0.667	0.333	1.000
UI Functionality	6	1.000	1.000	0.333	1.000
Model Format Support	4	1.000	1.000	0	0.250
Physics Engine Support	3	1.000	1.000	0	0
Real Time Factor	4	0.914	1.000	0.789	0.849
Average Load CPU Efficiency	2	0.364	0.174	0.333	1.000
Intense Load CPU Efficiency	2	0.583	0.304	0.583	1.000
IMU Angular Velocity Accuracy	10	0.875	1.000	0.792	0.867
IMU Linear Acceleration Accuracy	10	1.000	0.713	0.424	0.736
Total	**56**	**49.100**	**49.087**	**32.148**	**44.226**

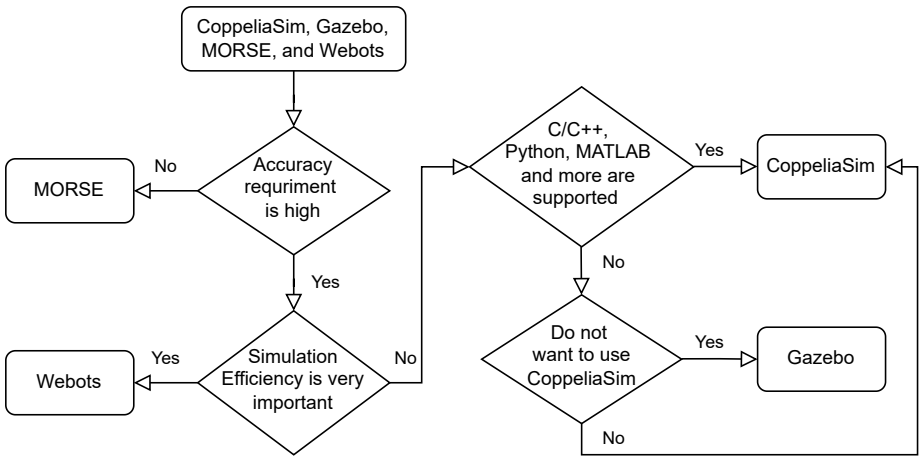

Figure 9: A decision tree comprising the most distinctive features among four simulators. The users can pick the best fit simulator quickly by answering to several questions.

5. Conclusions and Discussions

Although simulation is always a recommended step in the development of new robot systems, the ongoing COVID-19 pandemic means that many researchers and developers are turning to simulation when access to physical systems is limited due to safety concerns. The purpose of this paper is to help mobile roboticists choose a simulation platform suitable for their application. Four widely-available and popular robot simulators—CoppeliaSim, Gazebo, MORSE, and Webots—were, to the extent possible, quantitative and objectively compared by weighted simulator qualitative features and quantitative metrics. In comparing simulation output with actual laboratory data collected using a Husky A200 mobile robot and LORD IMU, the most accurate simulator was CoppeliaSim. CoppeliaSim also supports the largest number of programming languages, physics engines, and model types, making it the most versatile simulator of those we tested. It is worth noting that Gazebo had only slightly lower scores than CoppeliaSim, making it a very good alternative.

In the current work, we compare the selected robotic simulators quantitatively with a focus on the accuracy of motion. The accuracy of the simulated velocity and acceleration in each simulator is evaluated by comparing to the IMU data collected on a real Clearpath Husky A200 robot. The testing environment has three ground profiles (grass, bumps, and gravel) to provide

enough "uneven complexity" for motion accuracy evaluations. However, we didn't include exteroceptive dense 3D sensors, such as Lidar or camera in the simulator comparison. The quantitative evaluation of the simulators incorporating dense 3D sensors is a promising future work to provide more generalizable comparison results. A more complex experimental scenario with "visual complexity" is needed to evaluate the quality of dense 3D data virtual acquisition.

6. Acknowledgements

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the NSERC Canadian Robotics Network under grant number NETGP 508451-17.

References

[1] T. Barfoot, J. Burgner-Kahrs, E. Diller, A. Garg, A. Goldenberg, J. Kelly, X. Liu, H. E. Naguib, G. Nejat, A. P. Schoellig, F. Shkurti, H. Siegel, Y. Sun, S. L. Waslander, ., Making sense of the robotized pandemic response: A comparison of global and canadian robot deployments and success factors (2020). arXiv:2009.08577.

[2] M. S. P. de Melo, J. G. da Silva Neto, P. J. L. da Silva, J. M. X. N. Teixeira, V. Teichrieb, Analysis and comparison of robotics 3D simulators, in: 2019 21st Symposium on Virtual and Augmented Reality (SVR), IEEE, 2019, pp. 242–251.

[3] A. Harris, J. M. Conrad, Survey of popular robotics simulators, frameworks, and toolkits, in: 2011 Proceedings of IEEE Southeastcon, IEEE, 2011, pp. 243–249.

[4] S. Ivaldi, J. Peters, V. Padois, F. Nori, Tools for simulating humanoid robot dynamics: a survey based on user feedback, in: 2014 IEEE-RAS International Conference on Humanoid Robots, IEEE, 2014, pp. 842–849.

[5] T. Erez, Y. Tassa, E. Todorov, Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX, in: 2015 IEEE international conference on robotics and automation (ICRA), IEEE, 2015, pp. 4397–4404.

[6] D. Kang, J. Hwangho, SimBenchmark physics engine benchmark for robotics applications: RaiSim vs. Bullet vs. ODE vs. MuJoCo vs. DartSim, Available at `https://leggedrobotics.github.io/SimBenchmark` (2018/04/02).

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, ROS: An open-source Robot Operating System, in: ICRA Workshop on Open Source Software, Vol. 3, Kobe, Japan, 2009, p. 5.

[8] D. Ferigo, S. Traversaro, G. Metta, D. Pucci, Gym-Ignition: Reproducible robotic simulations for reinforcement learning, in: 2020 IEEE/SICE International Symposium on System Integration (SII), IEEE, 2020, pp. 885–890.

[9] E. Rohmer, S. P. Singh, M. Freese, V-REP: A versatile and scalable robot simulation framework, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2013, pp. 1321–1326.

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

[11] G. Echeverria, N. Lassabe, A. Degroote, S. Lemaignan, Modular open robots simulation engine: MORSE, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 46–51.

[12] O. Michel, Cyberbotics Ltd. Webots™: Professional mobile robot simulation, International Journal of Advanced Robotic Systems 1 (1) (2004) 5.

[13] L. Nogueira, Comparative analysis between Gazebo and V-REP robotic simulators, Seminario Interno de Cognicao Artificial-SICA 2014 (5) (2014).

[14] F. M. Noori, D. Portugal, R. P. Rocha, M. S. Couceiro, On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo?, in: 2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), IEEE, 2017, pp. 19–24.

[15] Husky A200 user manual, [Online]. Available at: https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot.

[16] Bullet real-time physics simulation, [Online]. Available at: https://pybullet.org.

[17] Open Dynamics Engine (ODE), [Online]. Available at: http://www.ode.org.

[18] Vortex Studio, [Online]. Available at: https://www.cm-labs.com/vortex-studio.

[19] Newton Dynamics, [Online]. Available at: http://newtondynamics.com.

[20] Dynamic Animation and Robotics Toolkit (DART), [Online]. Available at: https://dartsim.github.io.

[21] Simbody: Multibody physics API, [Online]. Available at: https://simtk.org/projects/simbody.

[22] Blender game engine, [Online]. Available at: https://www.blender.org.

[23] A. Munawar, Y. Wang, R. Gondokaryono, G. S. Fischer, A real-time dynamic simulator and an associated front-end representation format for simulating complex robots and environments, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2019, pp. 1875–1882.

[24] P. D. Cenek, N. J. Jamieson, M. W. McLarin, Frictional characteristics of roadside grass types, Opus International Consultants, Central Laboratories, Gracefield, New Zealand (2005).

[25] C. Lex, Maximum tire-road friction coefficient estimation, Ph.D. thesis, Graz University of Technology (2015).

[26] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, E. Fernández Perdomo, ros_control: A generic and simple control framework for ROS, The Journal of Open Source Software (2017). doi:10.21105/joss.00456.

[27] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, M. Hutter, Learning quadrupedal locomotion over challenging terrain, Science robotics 5 (47) (2020). doi:10.1126/scirobotics.abc5986.

[28] Webots reference manual, [Online]. Available at: `https://cyberbotics.com/doc/reference/inertialunit`.