Botball meets ARIS, a novel Autonomous Robot Interface Simulation System

Lukas Leskovar*, Marcel Dinhof, Manuel Kempf, Antonia Oberhauser, Florian Zachs, Niklas Wieser
Technical Secondary College
Department of Computer Science
2700 Wiener Neustadt, Austria
*Corresponding author Email: leskovar.lukas@student.htlwrn.ac.at

***Abstract*—Testing is a critical, time-consuming, and tedious part of the development of a Botball® robot. A simulation system could support the user in this process and improve results as errors due to over-adjustment can be ruled out. This work describes the implementation and validation of a Robot Operating System (ROS) based simulation system called ARIS (Autonomous Robot Interface Simulation System). For this purpose, the precision of simple robot movements in the setting of a game table using ARIS was analyzed and compared to data obtained on robot behavior in the real world. The results provide evidence for the general feasibility of the developed simulation system, thus being a starting point for further projects. Considering the highly complex scenario of Botball® the simulation at its present state does not sufficiently meet all requirements to sustainably aid physical testing.**

I. Introduction

While developing a robot, many aspects have to be taken into account, but one of the most critical and time-consuming parts is testing. Especially when striving for high precision and reliability, it is essential to repeat and document the robot's task many times to achieve meaningful results, particularly in preparation for a Botball® competition. For example, defining and fine-tuning the startup routine can be challenging for inexperienced users. Therefore simulating a robot's movements in a virtual environment could facilitate robot development.

As Botball® is a highly complex environment, it is challenging to simulate every aspect of the competing robot. Accurately, modeling a general-purpose simulation with all available sensors and actuators would require much more extensive development than possible during a Botball® season.

Therefore the authors decided to develop a simple simulation of fundamental motion. The simulated movements were then compared to a real-world system.

This paper proposes a prototype for simulating a robot carrying out linear motion on a Botball® game table by implementing the system of ARIS (Autonomous Robot Interface Simulation).

II. Study of Literature

As R. D. Smith states, "Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model" [1]. Consequently, simulation is one of the most important branches in the technological sector, states Smith.

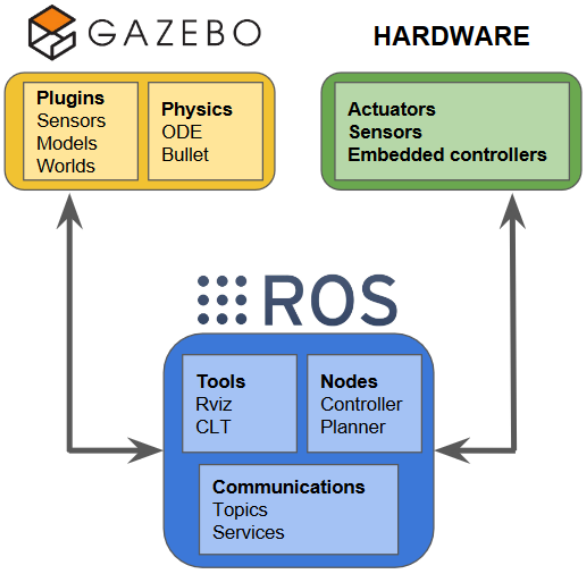


Fig. 1: Diagram visualizing the correlation between ROS, GAZEBO and the robot hardware to be simulated, whereas the pluggable design of ROS allows for easily substituting a real world system without having to change the controlling algorithms [2].

Its use-cases range from relatively simple situations like simulations for educational purposes and video games to simulating computational experiments in physics, chemistry, and other scientific areas. For example, in robotic engineering, simulation systems' main task is testing a robot's behavior and conveniently adjusting its parameters until it performs as expected.

One example for the simulation of robots is the AWS DeepRacer competition, where robots have to follow a predefined track as quickly as possible[3]. The robots are controlled by a machine-learning algorithm trained in the provided simulation environment.

Another significant example within the space of Botball® is the KISS IDE Simulator, an online typescript-based simulation of the KIPR demo bot with an integrated IDE tailored to Junior Botball® teams [4].

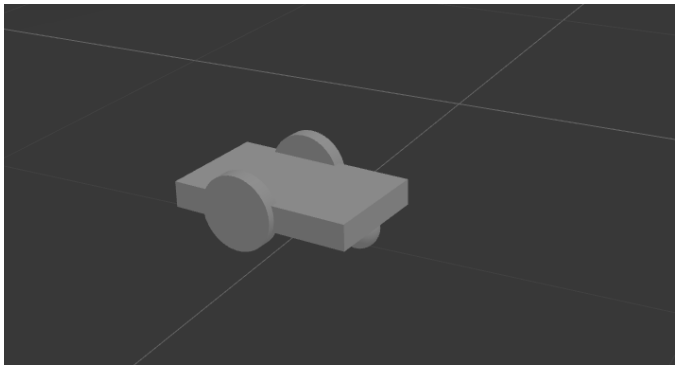


Fig. 2: A depiction of a basic robot model in the GAZEBO simulation which is used in section V.

Although the KISS IDE Simulator already proposes an easy-to-use simulation, it lacks configurability of robot hardware. This renders it infeasible for many Botball® teams operating with custom robots rather than the demo bot.

III. Systems & Tools

ARIS takes a first step into implementing a fully customizable robot simulation.

The system is based on the Robot Operating System (ROS) [6] in combination with GAZEBO [7]. ROS is responsible for controlling the robot, while GAZEBO is used for simulating the 3D environment (see Figure 1).

A. ROS

ROS is an open-source framework providing many libraries and tools for developing robotic applications. It implements functionalities found in conventional operating systems and robot frameworks, such as hardware abstraction, process communication, and package management. ROS facilitates code reuse in robotic applications through integration in various languages and frameworks.

By implementing packages as lightweight as possible, with each aiming to contain only enough functionality as necessary for a specific purpose, the broad applicability of ROS is emphasized.

Furthermore, its open-source code repository system enables the community to share and use packages within ROS [8].

B. GAZEBO

GAZEBO is an open-source 3D dynamics simulator with the ability to accurately and efficiently simulate robots in intricate indoor and outdoor environments. It uses a distinct set of libraries for physics simulation, rendering, user interface, communication, and sensor generation [9].

A robot in GAZEBO can be defined in the Simulation Description Format (SDF), an XML format used for robot simulation, visualization, and control [10].

Figure 2 shows how such a robot may be represented within GAZEBO.

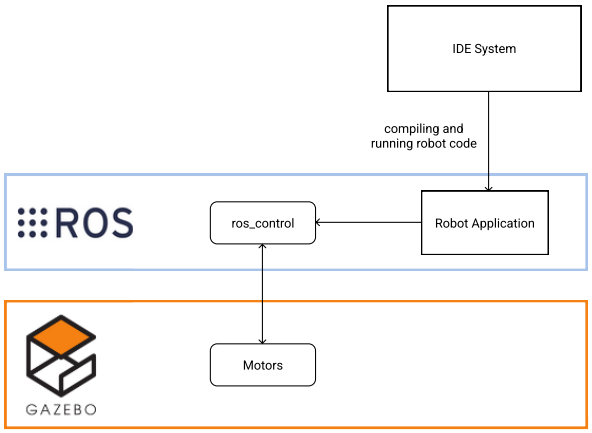


Fig. 3: Diagram depicting the architecture of ARIS.

To autonomously control a robot in the GAZEBO simulator, ROS provides the gazebo_ros_pkgs [12] package, which contains the necessary interfaces for communication between ROS and GAZEBO.

IV. Autonomous Robot Interface Simulation

To improve the process of testing a robots movement on the game table, a simple GUI with a built-in IDE was included in the project. A library with Wallaby-like motor control functions was implemented to allow ARIS to use the same code as for the Wallaby. While their function signature remains identical to the libwallaby, they internally implement robot control using the ros_control package. Following error-checking of the code, ROS interprets the input and uses GAZEBO to control a simulated robot model. GAZEBO offers a high level of detail, as realistic sensor inputs can be simulated easily. Ultimately the system should provide the user with an accurate estimation of how the robot would behave with the given code basis and helps to improve the movement strategies of the robot on a Botball® game table.

A. Concept

The structure of ARIS is split into two parts.

The first part deals with the interaction between ROS and GAZEBO. Description files for the simulation were stored in catkin [13], a build system based on CMake and Python Scripts. It is primarily used in ROS for structuring packages. The most essential files for GAZEBO are the launch file and the Unified Robot Description Format (URDF) file. Both are structured in XML and contain the information needed for the robot and its environment. GAZEBO's so-called "empty world" was used for the experiments. It is a pane with a single light source into which a self-made 3D model of the 2019 game table was placed. In order to put the robot into motion, a ROS plugin for differential drive was included in the simulation.

The second part is the Graphical User Interface (GUI) written in Java and Python, providing a simple editor with auto-completion. Additionally, the connection to ROS was

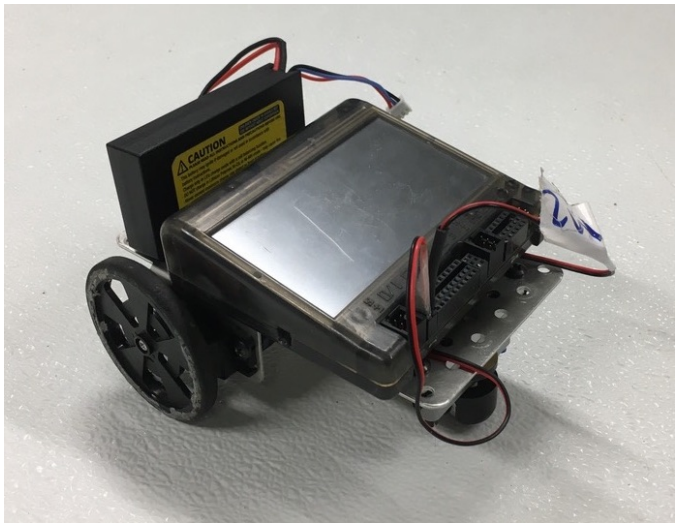


Fig. 4: For experimental purposes, a relatively simply built robot was used, consisting of a metal base plate, two solarbotics wheels powered by standard Botball® motors, a ball-caster wheel and the KIPR Wallaby.

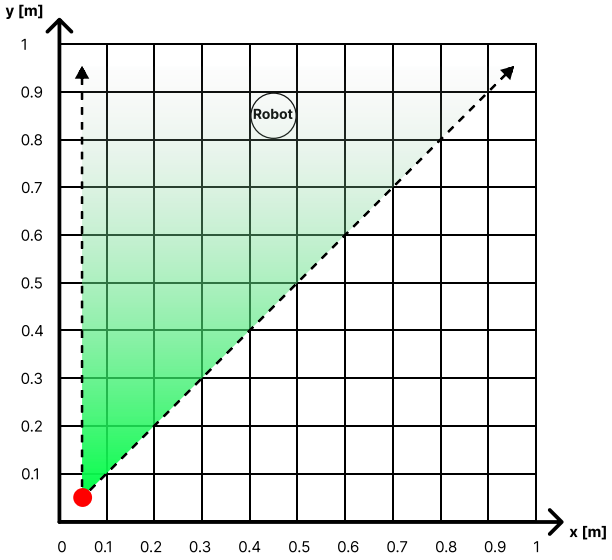


Fig. 5: Schematic representation of a 1x1m grid serving as the experimental setup with the possible movement area of the robot illustrated by the two dashed arrows. The red dot represents the starting position while the circle labeled robot corresponds to a potential end location.

established by Python scripts operating in the background. When running ARIS, a GAZEBO window opens, showing the simulated world in which the robot moves.

Subsequently, ROS establishes a connection to GAZEBO and the simulated robot. After all communication between the subsystems is set, ARIS is able to execute the code in the simulation. In order to better reflect the system's structure, the interaction between all components can be seen in Figure 3.

V. Experiments

The system aims to simulate movement conditions as close to reality as possible. This ensures that the code written for the simulated robot can be used in real-world applications without significant adjustments.

A. Setup

For the experiment, a camera was installed on the ceiling above the game table, which captured a 1x1m area in which the robot operates. With OpenCV [14] a grid identical to the one in the 3D simulation was superimposed with the captured footage. The grid was used to position and orientate the robot at the same starting point as the simulated robot, thus, providing a link between reality and simulation. Each quadrant of the grid had a size of 10x10cm. To indicate the position of the robot in the grid, the center of the robot was highlighted.

B. Execution

The simulated robot was put into the center of the bottom left quadrant.

The robot then was given the objective to move to the center of another given quadrant with an alignment angle between 90° and 45° from the starting point, resulting in distances between 10 and 127.28 cm, as seen in Figure 5. Afterwards, the robot's position within the simulated coordinate system was recorded. This process was repeated for all quadrants within the movement area.

The center of the targeted quadrant was used as a reference point for analyzing the deviation measured in the experiments.

The setup for the real robots experiments was identical to the simulation, with the robot also performing a linear movement. After the machine stopped, the coordinates of the robot on the game table were determined by measuring the distance between the boundaries of the grid and the center of the robot using analog techniques (see Section V-A).

Two values where collected for each iteration:

- position of the simulated robot
- position of the real robot

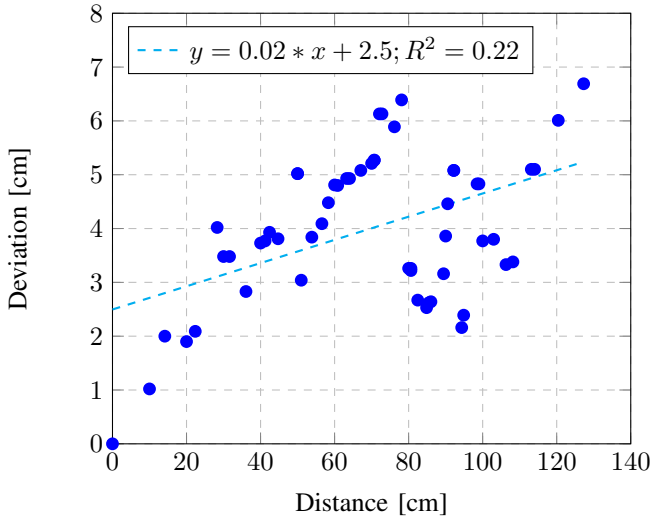
As mentioned, the center of each quadrant served as the reference for calculating the deviation in both scenarios, yielding the overall precision of both systems.

VI. Results

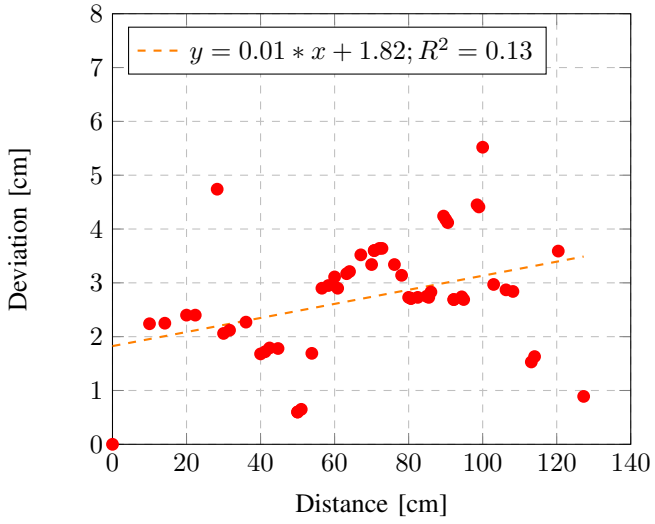
A. Simulated robot

The blue marks visualize the results for the simulated robot in Figure 6A, which represent the distance driven (as the x-axis) and the deviation between the position of the simulated robot and the reference value (as the y-axis).

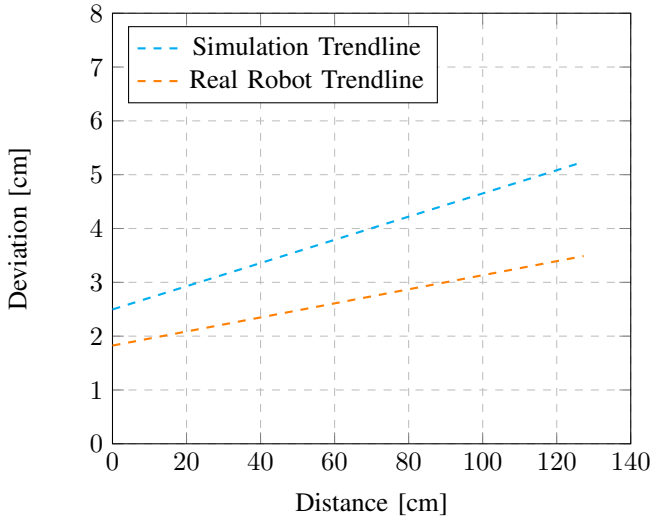
As visible, the deviation indicates an increase in values with growing distances which is clarified by superimposing a linear regression. The observed tendency meets the expectation that the deviation increases with growing distances. However, the



(A) Deviation from the reference value in correlation with increasing distances for the simulated robot.



(B) Deviation from the reference value in correlation with increasing distances for the real robot.



(C) Linear regressions of the simulated (cyan) and real (orange) robot data.

Fig. 6: Visualized data which was obtained in Section V-B.

coefficient of determination R^2 shows that the trend line can explain only 22% of data points. This property indicates that the linear regression model cannot sufficiently describe the correlation between distance and deviation of the simulation. The visible scattering of values reflects the simulation's similarity to reality and its corresponding inexactness.

B. Real robot

As mentioned above, the same experiment was performed with the real robot, which produced the results highlighted by the red marks in Figure 6B.

Comparable to the results for the simulated robot, an increasing deviation can be observed, as illustrated by the linear regression. Additionally, R^2 shows that the calculated linear fit is not suitable for predicting values of the real-world experiment, similar to the one observed in the simulation. Dealing with real-world mechanical inaccuracies as well as inexactness in the experimental setup explains the spread of values.

C. Comparing simulation and reality

When comparing the simulated robot's results with real robot experiments, the obtained data points do not match exactly. However, a notable parallel course is detectable.

As shown in Figure 6C the calculated linear regression for both settings are not identical in terms of slope and absolute deviation values, thus pointing out remaining inaccuracies of the simulation. As the simulated robot has not been extensively calibrated to meet factors such as friction, weight and wheel imperfections may have influenced the simulation may not precisely match real-world error patterns.

VII. Conclusion

Our data show evidence for the basic applicability of ARIS in the practical usage of Botball® thus facilitating the development of competing robots for inexperienced users and Junior Botball® students. However, the fact that the presented simulation system was tested exclusively for linear movements and its weaknesses in terms of precision necessitates further research. Specifically, the interaction of factors influencing the robot's behavior in the real world environment could be reflected more accurately in the simulation. In summary, a prototype for the simulation of a simple Botball® robot is provided, thus building the base for the continuous development of a sophisticated tool for substantially more complex robots, environments, and scenarios.

Although ARIS only implements the most basic concept of a robot, the topic does introduce promising ideas to the field of Botball®. Furthermore, the connection of KIPR libraries, ROS, and GAZEBO may open up new possibilities for modeling, testing, and deploying competing robots.

The development of this basic model will be continued as a fully-fledged software project to implement a fully functional software application for Botball®.

ACKNOWLEDGMENT

The authors would like to thank Dr. Michael Stifter for his support throughout the work on this paper.

REFERENCES

[1] R. D. Smith - Simulation The Engine Behind The Virtual World - https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.158.1876&rep=rep1&type=pdf (Accessed December 4th 2021)

[2] Mayank Mittal - Introduction to Robot Simulation (Gazebo) - https://mayankm96.github.io/assets/documents/teaching/ae640a/ae640a_lecture2.pdf (Accessed December 27th 2021)

[3] Amazon Web Services, Inc. - AWS DeepRacer - https://aws.amazon.com/deepracer/?nc=sn&loc=0 (Accessed December 4th 2021)

[4] KISS Institute for Practical Robotics - KISS IDE Simulator - https://github.com/kipr/simulator (Accessed March 27th 2022)

[5] KISS Institute for Practical Robotics - KIPR Wallaby - https://www.kipr.org/kipr/hardware-software (Accessed February 20th 2022)

[6] Open Source Robotics Foundation - ROS Wiki - https://wiki.ros.org/ (Accessed December 6th 2021)

[7] Open Source Robotics Foundation - GAZEBO - http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1 (Accessed December 4th 2021)

[8] K. Lampalzer, M. Grill - Autonomous Aerial Reconnaissance Drone - https://robo4you.at/publications/Autonomous_Aerial_Reconnaissance_Drone.pdf (Accessed December 6th 2021)

[9] Architecture - GAZEBO Tutorial - http://gazebosim.org/tutorials?tut=architecture&cat=get_started (Accessed December 25th 2021)

[10] Open Source Robotics Foundation - Simulation Description Format - http://sdformat.org/ (Accessed December 25th 2021)

[11] ROS integration - GAZEBO Tutorial - http://gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros (Accessed December 25th 2021)

[12] gazebo_ros_pkgs - ROS Wiki - https://wiki.ros.org/gazebo_ros_pkgs (Accessed December 6th 2021)

[13] catkin overview - ROS Wiki - http://wiki.ros.org/catkin/conceptual_overview (Accessed February 20th 2022)

[14] OpenCV Team - Open Source Computer Vision Library - https://opencv.org/about/ (Accessed January 15th 2022)