ROS 2 for RoboCup

Marcus M. Scheunemann$^{(\boxtimes)}$ and Sander G. van Dijk

University of Hertfordshire, Hatfield, AL10 9AB, UK
marcus@mms.ai, sgvandijk@gmail.com
https://mms.ai
https://robocup.herts.ac.uk

Abstract. There has always been much motivation for sharing code and solutions among teams in the RoboCup community. Yet the transfer of code between teams was usually complicated due to a huge variety of used frameworks and their differences in processing sensory information. The RoboCup@Home league has tackled this by transitioning to ROS as a common framework. In contrast, other leagues, such as those using humanoid robots, are reluctant to use ROS, as in those leagues real-time processing and low-computational complexity is crucial. However, ROS 2 now offers built-in support for real-time processing and promises to be suitable for embedded systems and multi-robot systems. It also offers the possibility to compose a set of nodes needed to run a robot into a single process. This, as we will show, reduces communication overhead and allows to have one single binary, which is pertinent to competitions such as the 3D-Simulation League. Although ROS 2 has not yet been announced to be production ready, we started the process to develop ROS 2 packages for using it with humanoid robots (real and simulated). This paper presents the developed modules, our contributions to ROS 2 core and RoboCup related packages, and most importantly it provides benchmarks that indicate that ROS 2 is a promising candidate for a common framework used among leagues.

Keywords: ROS 2 · Robot framework · Robot software · Embedded system · Real-time system · Minimal hardware · Open source · Humanoid robots · Autonomous robots

1 Introduction

Having a common framework among teams (or even among leagues) has many advantages. Most notably, rather than concentrating on increasing performance or reliability of the framework, participants can focus on the implementation of artificial intelligence. Solutions can be easily shared and distributed, with possibilities of benchmarking them against each other. The handover and knowledge transfer within a team to a new generation can be done smoothly. If there is a common framework not just within a league but between leagues, then this may foster the collaboration between teams of different leagues. With an eye on

the goal of merging forces of different leagues and eventually merge leagues, a common framework is essential.

In a field such as a RoboCup league, where tasks and constraints are similar for each participating team, one might think that a common framework would naturally emerge. The RoboCup@Home Open Platform League is a good example that this indeed happens. Starting with two teams in 2010, in 2018 all teams announced within their team descriptions the use of the same software framework [9]. The used framework was the open-source Robot Operating System (ROS). In other leagues, such as the Humanoid League (HL) or the Standard Platform League (SPL), ROS didn't establish as a common framework despite some advances [1,2,5,14]. We argue that performance shortcomings on minimal hardware doesn't make it suitable for soccer playing humanoid robots [11]. In the SPL the self-developed framework from team B-Human was adopted by many teams instead. The performance strength is gained through a tight coupling between used software tools (e.g. simulator) and between modules [11]. However, this is also a shortcoming, as it restricts the community using the framework mostly to the football playing domain. NUClear is an example of a framework originating from RoboCup that is more loosely coupled, modular and applicable in different robot projects [7]. This framework solves the overhead of traditional message passing systems through specially optimised paths similar to those we will benchmark here. Additionally, it offers the ability of using more blackboard/whiteboard type data access patterns which are not directly available in the framework discussed in this paper.

The authors themselves developed several frameworks with teams in the Standard Platform League, Humanoid Kid-Size League and 3D Simulation League. Their current team, the Bold Hearts, used a self developed software framework, with almost all modules created from scratch [13]. Although shown to be capable of performing well, over the years the framework has become more and more complex. It is completely custom and some of the original developers have moved on, making it difficult to get new members started and to adapt it to new developments in the competitions. Code dependencies made it challenging to integrate well working modules from other teams or projects. For example, last year's change of the underlying vision pipeline towards a new semantic segmentation couldn't be easily achieved as third-party tools didn't integrate well together [3].

There are reasons why some leagues couldn't agree on a common framework yet. Naturally, without a central committee being in charge, there will always be some healthy argument or a fork of a different, perhaps slightly more efficient implementation.

The fact that ROS is a framework from 'outside' of RoboCup could provide a good common base, but has not been able to gain widespread adoption due to inherent limitations; ROS was never built with support for, e.g., multi-robot systems involving unreliable networks, for robots needing real-time processing capabilities or for robots with minimal hardware [4,8,11].

However, exactly these limitations sparked the development of a second, completely rewritten version of ROS. Although it is not yet deemed fully production ready, it has a wider support than a RoboCup team, a league or the whole com-

munity can offer: the support includes large entities from the industry, such as Intel and Amazon. With an eye on the future and the transferability of skills learned by our (student) members outside/beyond participation in RoboCup, we opted to use ROS 2 [13]. Some benchmarking suggests that ROS 2 is currently in a state that offers the possibility to use it for multi-robot teams, small platforms and real-time systems [6,8].

In this paper, we discuss why we think ROS 2 is a reasonable framework choice and also briefly present its advantages (Sect. 2). We further present the modules we have developed as a basis for participating in the RoboCup (Sect. 3). Additionally, we present two preliminary studies for benchmarking the system and showing the feasibility for using it in RoboCup (Sect. 4).

2   ROS 2 Architecture and Features

ROS 2 is based on the Data Distribution Service (DDS) standard for real-time systems [4,8]. DDS is a connectivity framework aiming to enable scalability and real-time data exchange using a Data-Centric Publish-Subscribe (DCPS) architecture [10].

DDS is specified by the Object Management Group (OMG), which is an open membership, not-for-profit computer industry standards consortium. It is developed for a wide variety of fields such as transportation systems, autonomous vehicles, and aerospace. ROS 2 sits on top of that, providing standard messages and tools to adapt DDS for robotic needs. A range of vendors providing implementations of DDS, such as eProsima's Fast RTPS and RTI Connext are fully supported [8,12]. Compared to ROS 1, ROS 2 has several beneficial features:

– Built-in support for real-time systems.
– Support for defining the 'Quality of Service' of topics. This allows one to make a range of trade-offs between strong reliability and 'best effort' policies, to deal with lossy communication. For instance, an efficient non-blocking 'best effort', 'UDP-like' service is acceptable for high frequency sensor data where missing individual messages is not detrimental. On the other hand, when each message is crucial, a reliable 'TCP-like' policy can be used.
– Nodes can be run in individual executables, or composed, using a variety of executors. In ROS 1, one has to maintain 'nodelet' versions of all nodes to make this possible. In ROS 2, this can be achieved natively, making it possible to remove much of the communication overhead between nodes by having them share memory. We provide a benchmark showing the appeal of this below.
– No need to run the ROS 1 `roscore` instance and maintain environment variables to make it and nodes reachable; with DDS, nodes discover each other through a network automatically.
– Communication between nodes can be strictly restricted by placing them in different 'domains'. This could be very useful in the RoboCup Simulation league for instance, where the programs of all robots in the same team run on the same machine but no communication between them should happen.

Together with the ability to compose all modules for one agent in a single executable, this makes ROS 2 a much better candidate to use as a platform in the simulation league than ROS 1.

3   ROS 2 and RoboCup Contributions

There are many modules that are not available yet, as ROS 2 is still relatively new. To be able to develop a full RoboCup team based on ROS 2, we have developed several modules, consisting of:

Hardware driver. Our robots are based on the Robotis CM-730 sub-controller. Robotis has released ROS 1 packages for their products, but at the moment there is no ROS 2 effort. We have created and published a ROS 2 driver for interacting with the CM-730, and controlling Robotis Dynamixel motors
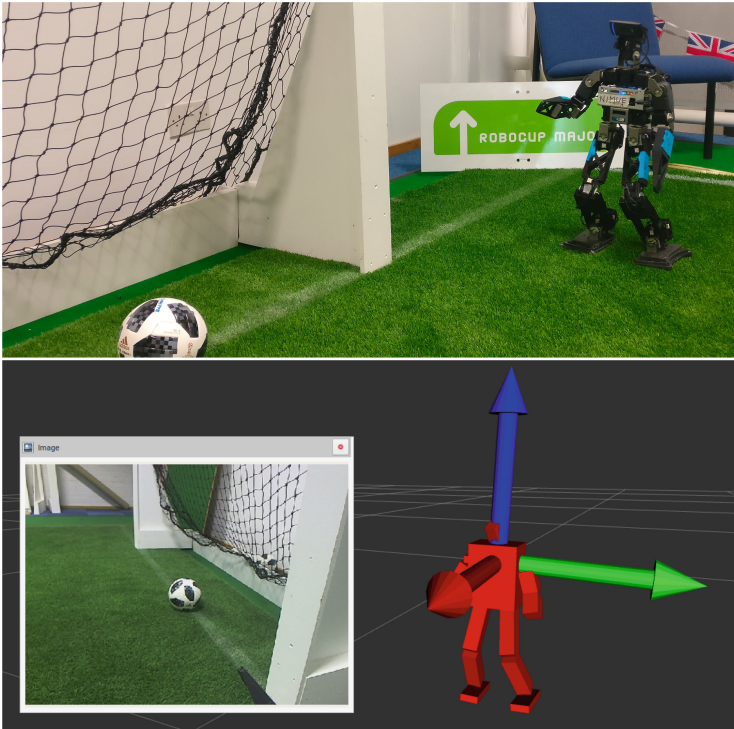

Fig. 1. The upper image depicts a scene with a robot looking at a ball. The lower image is a screenshot of RViz2. It shows the camera image feed retrieved with our USB camera driver (left). Our CM-730 package publishes joint states, accelerometer and gyro information for building the robot model and compute its orientation with our IMU fusion package (both right).
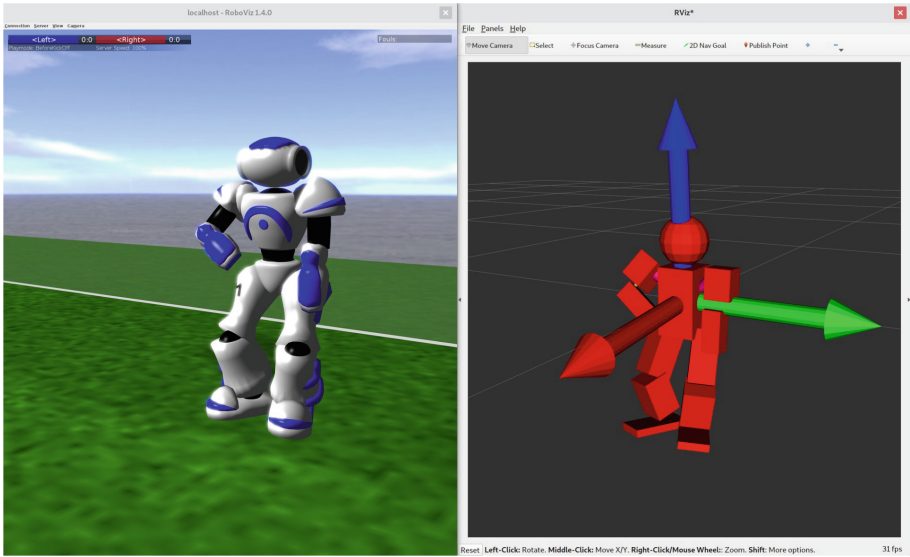
Fig. 2. Depicted is a scene from RCSSServer3D used in the 3D Simulation League (left, using RoboViz). Our package translates the servo information into standard ROS messages and publishes the topic /joint_state. Also, the simulated gyroscope and accelerometer information are published. Our IMU fusion package subscribes to the messages and computes the robot's orientation, exactly as for the real robot in Fig. 1. The interface package allows for using ROS 2 within the context of the 3D Simulation League.

attached to it[1]. Figure 1 (right) shows the result of the robot model built using the output of this driver. We also developed a USB camera driver[2], Fig. 1 (left) shows the output of the camera driver.

Ports of our modules. With all hardware interfaces in place, we now work on porting our existing modules over to the new platform. The IMU fusion filter[3] is one example. We complemented it with a package for visualizing the orientation of the robot in RViz2. Figures 1 and 2 (both right) show the visualization of the orientation.

Humanoid League. The RoboCup humanoid league uses a Game Controller application to manage a competition: it keeps track of and broadcasts the game state and events such as kick-off and penalties to the robots. We have created a package that forms a bridge between the communication protocol of the Game Controller and ROS 2 topics[4].

3D Simulation League. Taking advantage of the benefits described above, we have developed a ROS 2 interface for the RoboCup 3D Simulation

---

[1] https://gitlab.com/boldhearts/ros2_cm730
[2] https://gitlab.com/boldhearts/ros2_v4l2_camera
[3] https://gitlab.com/boldhearts/ros2_imu_tools
[4] https://gitlab.com/boldhearts/ros2_game_controller

Server [15][5]. It uses the same platform and standard message interfaces as for our humanoid robots, making it easier for (new) members to experiment and improve our modules, and deploy them to real robots directly.

ROS 2 core contributions. We have made several contributions to the core ROS 2 project for issues discovered in our use cases, including fixes to make it possible to compile ROS 2 for 32 bit ARM platforms[6], support modern Linux 64 bit library paths[7] and to set complex node parameters using command line tools[8], along with smaller fixes for `geometry2`[9] and demo `image_tools`[10].

4 Benchmark Stand-Alone Versus Composed Nodes

We use the humanoid football robot BoldBot for the benchmarks. Its main board is an Odroid-XU4. This device is based on a Samsung Exynos 5422 Cortex-A15 with 2 GHz and a Cortex-A7 Octa core CPU, which is the same as used in some 2015 model flagship smartphones [13]. The main board runs 32 bit Ubuntu 18.04 with a compiled version of ROS 2 Dashing Diademata, the first long-term support version of ROS 2. All packages used in these benchmarks and the ROS 2 core install have been compiled with GCC's highest optimization level.

In ROS 1, nodes are stand-alone executables. Communication between these nodes is performed through a transport protocol, most often over TCP/IP. This means that all communication between these nodes involves overhead from serialisation and memory copies. So called 'nodelets' were introduced to allow composing node-like building blocks into single executables to alleviate such overhead. As these concepts are separate from normal nodes, a package developer has to choose to support either nodes or nodelets, or maintain both.

In ROS 2, nodes were redesigned to make it possible to run them either standalone or composed in a single process, either single or multi-threaded. ROS 2 Dashing Diademata even adds the ability to dynamically load and unload nodes in a single process at runtime, as so called 'components'.

When nodes are composed in such a way, messages between them can be shared directly, without any intermediate conversion overhead. ROS 2 explicitly supports this form of Intra-Process Communication (IPC), bypassing the DDS layer and performing zero-copy communication where possible. To test the impact of this, we perform several benchmarks of one of the most memory intensive operations for robots: image processing.

4.1 Case 1: Simple Node Graph

A node is created that subscribes to the `/image_raw` topic provided by the camera driver described earlier. This node applies a Sobel gradient operation to

---

[5] https://gitlab.com/boldhearts/ros2_rcss3d

[6] https://github.com/ros2/rcl/pull/365

[7] https://github.com/colcon/colcon-library-path/pull/10

[8] https://github.com/ros2/ros2cli/pull/199

[9] https://github.com/ros2/geometry2/pull/102
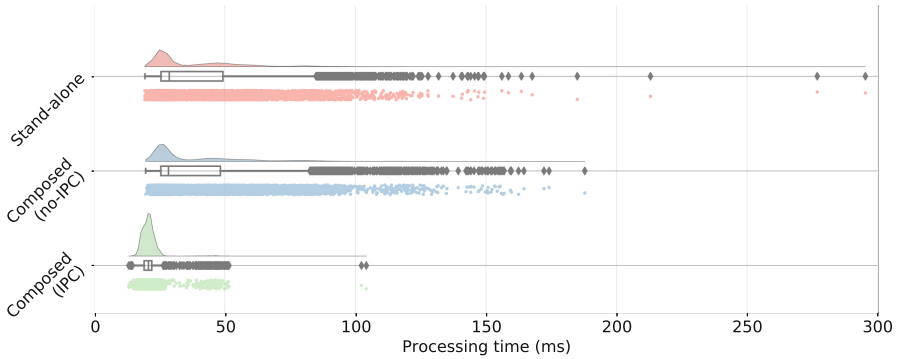
[10] https://github.com/ros2/demos/pull/288

Fig. 3. Distributions of time measured from image capture by camera node until end of processing of a Sobel gradient operation on the full image done by processing node. Both nodes either run as stand-alone executables or run composed in a single executable, the latter with IPC disabled and enabled. Each plot shows the density (top), a boxplot (middle), and individual data points (bottom). 10.000 samples are measured in each case.

incur some actual processing cost to offset overhead costs against, and measures the time from initial image capture (provided in the `Image` header) until it finished processing the image.

Figure 3 shows the effects of composing nodes and intra-process communication. The end-to-end processing time for 10.000 images was compared when running the camera and processing nodes separately, or when composed in a single executable using a multi-threaded executor[11], the latter with IPC disabled and enabled. Composing the nodes seems to result in slightly more stable communication, with less extreme outliers than in the stand-alone case, but the difference is minimal.

However, a clear benefit can be seen when IPC is enabled, with the median processing time dropping by 33% compared to the stand-alone results, from 28.7 ms to 20.7 ms. The communication is also much more reliable, with a distinctly narrower distribution of processing times. The zero-copy transmission that is responsible for this improvement is achieved in ROS 2 by passing direct memory pointers to messages from publisher to subscriber, when using the C++ 'unique pointer' concept to signal that this is safe to do.

4.2 Case 2: Extended Node Graph

We extend the previous benchmark example to further understand the impact of communication overhead in a more realistic configuration with a larger set of nodes. To extend the image processing example, we further introduce nodes to publish and process IMU sensor and joint data, provided by our packages

---

[11] When using a single-threaded executor no processing actually happened, possibly due to the camera node claiming all execution time
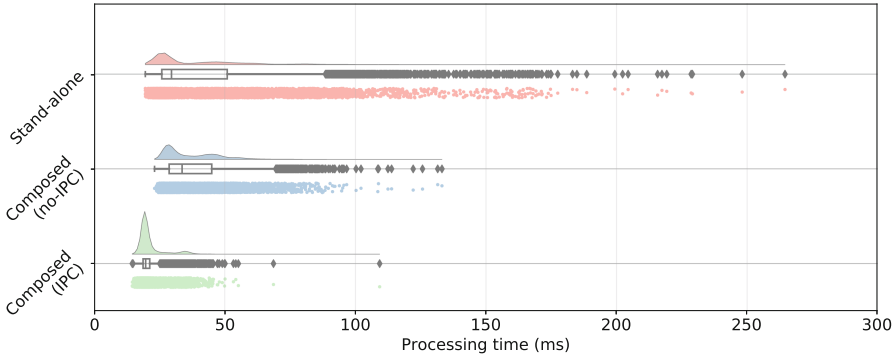
Fig. 4. Image processing times as in Fig. 3, but with the robot system extended with other necessary nodes and topics like `/joint_states` and `/imu/data`, for a total of 7 nodes running at a time. Each plot shows the density (top), a boxplot (middle), and individual data points (bottom). 10.000 samples are measured in each case.

described in Sect. 3. Our CM-730 package reads the servo information from the robot and publishes it as a standard topic `/joint_state`. It also publishes raw IMU readings of the accelerometer and the gyroscope. The IMU fusion package reads these raw messages and computes the robots' orientation and publishes the `/imu/data` topic. Altogether, 7 nodes are involved that communicate on 7 topics (including the image topic). For the benchmark using composition and IPC, only the `/image_raw` topic is IPC-enabled.

Figure 4 shows the results. The median computation time for the image processing for the stand-alone binary increases by 0.8 ms, only a slight increase when compared to the simple case from the previous section. However, the tail of the distribution has become longer, indicating further increase in communication variability.

This same effect is not seen in the composed, no-IPC results; the distribution is actually tighter, although the median processing time has increased from 28.4 ms to 33.6 ms. We do not know the reason for this change compared to the simple system. However, given the results when IPC *is* enabled, this mode is not recommended anyway.

This is because also in this extended case, enabling IPC gives significantly better results; hardly any effect of adding more nodes is visible in the image processing times, with the median time even 1 ms lower than in the simple case. These results show that a composed binary can prevent much of possible costs of creating a modular system without having to maintain any additional code, in contrast to ROS 1 for instance.

5   Conclusion and Future Work

We believe a common framework will ease the process of sharing and comparing solutions between teams and help them concentrate on their research interests.

This is valuable both within and between leagues. For instance, ultimately the simulation league aims more for strategic play, whereas the humanoid leagues focus heavily on developing robot hardware and lower level control. Given the goal of merging these leagues eventually, a common framework is an inevitable basis.

In this paper, we propose ROS 2 as a suitable choice for a framework for a RoboCup team with needs for real-time processing relying on minimal hardware. ROS 2 is supported by a large community, including big industrial partners, that the RoboCup community can benefit from. This makes it a potential candidate for a common framework. We presented our ROS 2 and RoboCup contributions, allowing to start with ROS 2 in the RoboCup. We supported the CM-730 sub-controller manufactured by Robotis, a very popular brand among RoboCup teams in the Humanoid League. Furthermore, we wrote an interface for using ROS 2 with the simulator of the 3D-Simulation League.

Our benchmarks indicate that ROS 2's capability to compose nodes can reduce the communicational overhead known from ROS 1. Our future work mostly includes porting over ROS 1 packages and our custom modules to the ROS 2 ecosystem.

References

1. Bestmann, M., et al.: Hamburg bit-bots and WF wolves team description for RoboCup 2019. Technical report (2019)
2. Bestmann, M., Hendrich, N., Wasserfall, F.: ROS for humanoid soccer robots. In: The 12th Workshop on Humanoid Soccer Robots at 17th IEEE-RAS International Conference on Humanoid Robots, p. 1 (2017)
3. van Dijk, S.G., Scheunemann, M.M.: Deep learning for semantic segmentation on minimal hardware. In: Holz, D., Genter, K., Saad, M., von Stryk, O. (eds.) RoboCup 2018. LNCS (LNAI), vol. 11374, pp. 349–361. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-27544-0_29
4. Fernandez, E., Foote, T., Woodall, W., Thomas, D.: Next-generation ROS: building on DDS. In: ROSCon Chicago, September 2014. https://roscon.ros.org/2014/wp-content/uploads/2014/07/ROSCON-2014-Next-Generation-of-ROS-on-top-of-DDS.pdf
5. Forero, L.L., Yáñez, J.M., del Solar, J.R.: Integration of the ROS framework in soccer robotics: the NAO case. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS, vol. 8371, pp. 664–671. Springer, Berlin Heidelberg (2014). https://doi.org/10.1007/978-3-662-44468-9_63
6. Gutiérrez, C.S.V., Juan, L.U.S., Ugarte, I.Z., Vilches, V.M.: Towards a distributed and real-time framework for robots: evaluation of ROS 2.0 communications for real-time robotic applications. https://arxiv.org/abs/1809.02595
7. Houliston, T., et al.: NUClear: a loosely coupled software architecture for humanoid robot systems. Front. Robot. AI **3**, 20 (2016). https://doi.org/10.3389/frobt.2016.00020
8. Maruyama, Y., Kato, S., Azumi, T.: Exploring the performance of ROS2.In: Proceedings of the 13th International Conference on Embedded Software.pp. 1–10. EMSOFT 2016, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2968478.2968502

9. Matamoros, M., Seib, V., Memmesheimer, R., Paulus, D.: RoboCup@Home: summarizing achievements in over eleven years of competition. In: 2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). IEEE, April 2018. https://doi.org/10.1109/icarsc.2018.8374181
10. Object Management Group (OMG): Data Distribution Service (DDS), Version 1.4., April 2015. https://www.omg.org/spec/DDS/1.4. Accessed 20 Mar 2019
11. Röfer, T., Laue, T.: On B-human's code releases in the standard platform league – software architecture and impact. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS (LNAI), vol. 8371, pp. 648–655. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44468-9_61
12. ROS 2: ROS 2 Documentation, April 2019. https://index.ros.org/doc/ros2/. Accessed 18 April 2019
13. Scheunemann, M.M., van Dijk, S.G., Miko, R., Barry, D., Evans, G.M., Polani, D.: Bold Hearts Team Description for RoboCup 2019 (Humanoid Kid Size League). Technical report, School of Computer Science, University of Hertfordshire, College Lane, AL10 9AB, UK, December 2018. https://arxiv.org/abs/1904.10066
14. Schwarz, M., et al.: Humanoid TeenSize open platform NimbRo-OP. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS (LNAI), vol. 8371, pp. 568–575. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44468-9_51
15. Xu, Y., Vatankhah, H.: SimSpark: an open source robot simulator developed by the RoboCup community. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS (LNAI), vol. 8371, pp. 632–639. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44468-9_59