A ROS-based Software Framework for the NimbRo-OP Humanoid Open Platform

Philipp Allgeuer, Max Schwarz, Julio Pastrana, Sebastian Schueller, Marcell Missura and Sven Behnke

Autonomous Intelligent Systems, Computer Science Institute VI, University of Bonn, Germany
{pallgeuer, pastrana, missura}@ais.uni-bonn.de
{schwarzm, schuell1, behnke}@cs.uni-bonn.de
http://ais.uni-bonn.de/nimbro/OP

Abstract—Over the past few years, a number of successful humanoid platforms have been developed, including the Nao [1] and the DARwIn-OP [2], both of which are used by many research groups for the investigation of bipedal walking, full-body motions, and human-robot interaction. The NimbRo-OP is an open humanoid platform under development by team NimbRo of the University of Bonn. Significantly larger than the two aforementioned humanoids, this platform has the potential to interact with a more human-scale environment. This paper describes a software framework for the NimbRo-OP that is based on the Robot Operating System (ROS) middleware. The software provides functionality for hardware abstraction, visual perception, and behavior generation, and has been used to implement basic soccer skills. These were demonstrated at RoboCup 2013, as part of the winning team of the Humanoid League competition.

Fig. 1. The NimbRo-OP robot.

I. Introduction

Progress in humanoid robotics research benefits from the availability of robust and affordable platforms that provide a suitable range of out of the box capabilities. Inspired by the success of the DARwIn-OP, a robot developed by the company Robotis, work commenced on the NimbRo-OP in early 2012. The first hardware prototype was displayed and demonstrated at the IROS and Humanoids 2012 conferences. As an open platform, both the hardware and software designs for the robot are committed to being regularly released open source to other researchers and the general public. This hopes to foster interest in the robot, and encourage research collaboration through community-based improvement of the platform. Following the first hardware release of the NimbRo-OP, an initial software release was completed that was heavily based on the framework released by Robotis for the DARwIn-OP. This was however only a transitional release pending an original and improved software framework based on the ROS middleware. The second software release for the NimbRo-OP, available at [3], constitutes exactly such a framework, and is referred to as the NimbRo-OP ROS Framework. The NimbRo-OP robot is shown in Figure 1.

ROS [4], an open source meta-operating system designed for robots, provides features such as hardware abstraction, low-level device control, packages for common functionalities, a communications framework, and a variety of other libraries and tools. It is maintained by the Open Source Robotics Foundation, but is largely a community project in terms of the contribution of packages. As the NimbRo-OP robot is currently predominantly being tested and demonstrated in the setting of humanoid soccer (i.e. RoboCup), numerous aspects of the framework are specific to this domain. By virtue of the modular architecture of R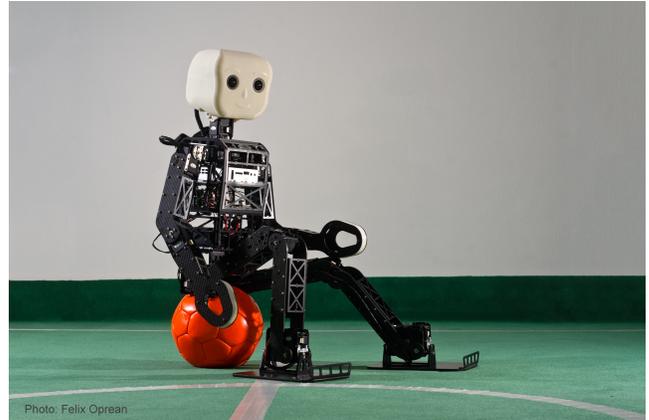OS and the framework however, this does not prevent the robot from being used in other settings. For example, the soccer nodes can simply be omitted from being launched.

II. Related Work

Due to its modularity, flexibility and portability, ROS has to date proven to be a popular choice in the research community for the implementation of robot control architectures. One of the main attractions of ROS is that it is so widespread, and allows the work of different research groups from around the world to be combined and shared in the form of ROS packages, even across robot hardware barriers. This facilitates collaboration, and allows groups to both benefit from the robotics research community, as well as contribute to it. Examples of wheeled robots that have ROS integration include the PR2 and the TurtleBot. While ROS has many advantages due to its abstraction and modularity, it also increases the communication requirements within the system, with data such as vision detections and behavior commands needing to be transferred via pipes.

Fewer examples of humanoid robots exist that have been integrated with ROS. For example, the DARwIn-OP nominally uses its own custom software framework, even though this framework was written with future ROS integration in mind [2]. To the knowledge of the authors, no open source ROS framework for the DARwIn-OP has become available to date. To a limited extent, a ROS framework for the HUBO robot, a 130 cm tall humanoid robot developed at the Korea Advanced Institute for Science & Technology, has been developed [5]. So far, this has only progressed to simple joint level control

WiFi adapter
Standards: IEEE 802.11b/g/n

Wide-angle camera
Logitech C905

Speaker

ZBOX nano XS
AMD E-450 processor 1.65 GHz
2 GB RAM, 64 GB SSD
USB 3.0, HDMI
Gigabit Ethernet
Memory card slot

CM730 control board
Servo communication
3-axis accelerometer
3-axis gyroscope

LiPo battery
14.8 V, 3.6 Ah

20 actuators
Networked Dynamixel
MX actuators
6 per leg (MX-106)
3 per arm (MX-64)
2 in the neck (MX-64)

Lightweight materials
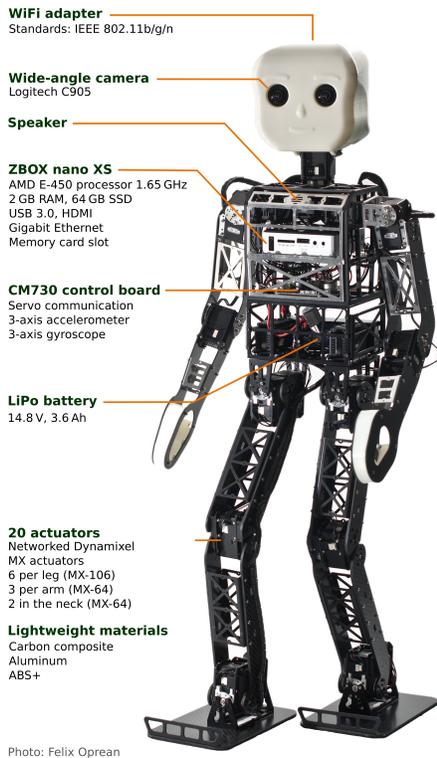Carbon composite
Aluminum
ABS+

Photo: Felix Oprean

Fig. 2. Diagram of the NimbRo-OP robot hardware.

though, stopping short of implementing ROS-compatible walking or manipulation tasks.

The widely-used Nao robot is a success story when it comes to ROS integration. Based on an initial port by researchers at Brown University, the Humanoid Robots Lab at the University of Freiburg has developed, and continues to maintain, a collection of ROS packages and stacks that wrap the NaoQI API and make it available in ROS. The UChile Robotics Team has published an extensive tutorial on how to build, install and run ROS natively on the latest version of the Nao [6]. This was believed to be the first initiative of its kind for the Nao robot.

III. Hardware Overview

A diagram of the NimbRo-OP that highlights all of the main hardware features is shown in Figure 2. The robot is 95 cm tall and weighs a total of 6.6 kg with the battery. This combination of height and weight was chosen so as to allow the robot to be driven using only a single actuator per joint. This decreases the cost and complexity of the design, and allows the robot to exhibit quite large ranges of motion in its 20 degrees of freedom. Robotis Dynamixel MX series actuators are used for all of the joints, of which there are two in the neck, three in each arm and six in each leg. All of the Dynamixel servos are chained together and addressed using a single one-wire TTL serial bus.

At the heart of the robot is a Zotac Zbox nano XS PC, featuring a dual-core AMD E-450 1.65 GHz processor with 2 GB RAM and a 64 GB solid state disk, and various communication interfaces including USB 3.0, HDMI and Gigabit Ethernet ports. A USB WiFi adapter allows also for wireless

networking. Data connections to the robot automatically switch between TCP/IP and UDP depending on whether the connection is wireless or not. The camera in the NimbRo-OP's right eye, a Logitech C905 model fitted with a wide-angle fisheye lens, is also connected to the PC via USB. The extremely wide field of view is convenient for keeping multiple objects in view at the same time.

A CM730 control board from Robotis, as used in their DARwIn-OP robot, sits at the approximate center of mass of the robot in the trunk. This board incorporates a reprogrammable 32-bit ARM Cortex CPU and an inertial measurement unit, featuring both a three-axis accelerometer and a three-axis gyroscope. Currently this CPU is simply used to relay the sensory data to the computer and interface with the servos. Three buttons and an array of multicolor LED lights are connected to the CM730, and are available for custom use. As a non-standard addition, a three-axis compass has been mounted in the trunk and interfaced to the CM730. This allows the directional heading of the robot to be estimated. Appropriate software for the anytime recalibration of the compass has been developed. More detailed descriptions of the NimbRo-OP hardware can be found in [7] and [8].

IV. Software Architecture

The NimbRo-OP software architecture makes use of ROS's native ability to split a software solution into multiple independent processes called nodes. Each node handles a certain aspect of the overall control task, and can be written, compiled and launched separately from the other nodes if required. The nodes are coupled using the ROS communication infrastructure and exchange information in a direct peer-to-peer manner via ROS topics, service calls, shared memory, and more. A decoupled architecture of this type is advantageous in that it can be implemented in a very modular way, with relatively high independence between the various elements of the system. Another advantage is that it naturally can be split along the process boundaries and run on multiple cores, in a fashion dynamically managed by the operating system scheduler.

The NimbRo-OP software framework is implemented almost exclusively in C++, with only miscellaneous use of additional scripting to perform tasks such as setting up a computer for initial use of the framework, compiling the package documentation, and providing framework-specific command line tools for everyday use. XML-based launch scripts exist for all of the main tasks and configurations of the framework, and are executed using the native ROS launch tool. The framework runs on a Linux operating system, nominally Ubuntu. This system constraint is predominantly imposed by the compatibility requirements of ROS itself.

A schematic of the NimbRo-OP ROS framework software architecture is shown in Figure 3. The figure is generally arranged such that the perception and state estimation tasks appear on the left, the lower-level sensor and motion control tasks (i.e. Motion Control Layer; see Section V) appear on the right, and the higher level planning and control tasks appear in the middle.

At the heart of the lower level control section is the *Robot Control* node, which contains the main real-time control loop that communicates with the sensors and actuators, the camera
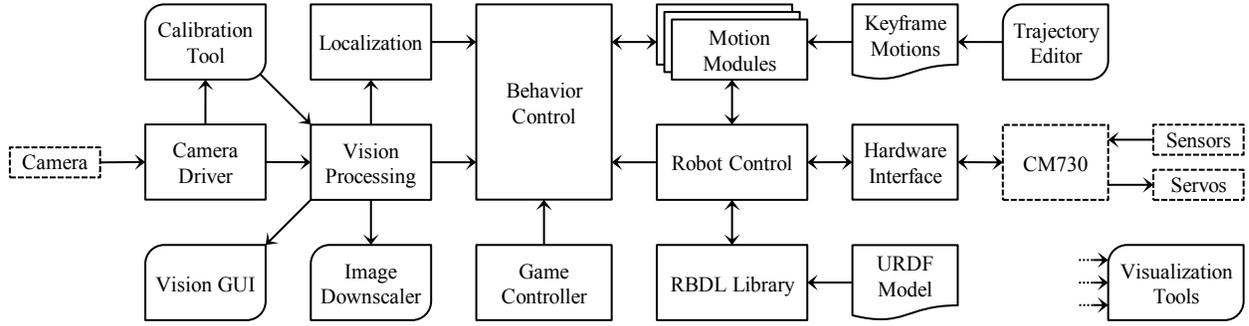
Fig. 3. Diagram of the NimbRo-OP ROS framework software architecture. Boxes with a dashed outline represent hardware elements, boxes with rounded corners represent auxiliary software tools that are not necessarily required for execution of the framework, and the arrows show the flow of information. Boxes with a wavy lower edge represent data files. The visualization tools use information from throughout the entire architecture.
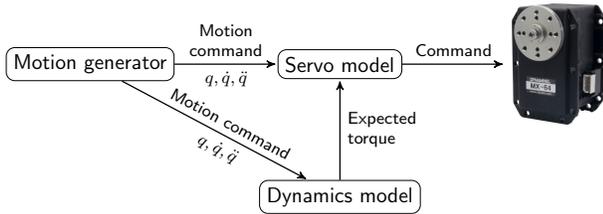

Fig. 4. Schematic of the servo control architecture.

Fig. 5. Screenshot of the trajectory editor, used to generate and edit the keyframe motions used by the motion player.

excluded. This communication occurs via the CM730 control board, and an exchangeable software component referred to as the *hardware interface*. The Robot Control node makes use of the RBDL library [9] for various kinematic and dynamic calculations, which in turn requires a model of the NimbRo-OP in Unified Robot Description Format (URDF) format, loaded from disk. Even though the servos are driven in the Robot Control node, the coordination of the actual robot motions are left to the *motion modules*. One of these modules, a keyframe motion player, can execute motion trajectories that are read from file. In control of the motion modules and responsible for all higher level planning is the *behavior control* node. In the case of robot soccer, this reads game state data from the *Game Controller* node.

A summary of the vision and localization nodes can be seen on the left-hand side of Figure 3. Images from the camera are made available via the camera driver, and forwarded for processing by the color calibration tool and vision processing node. Further tools include the vision graphical user interface, and the image downscaler, described in Section VI-A along with the remainder of the vision system. The detections from the vision processing are read by the behavior control node, and are used to drive the robot localization.

V. MOTION CONTROL LAYER

A. Robot Control Node

As indicated in Figure 3, the Robot Control node is the core of the Motion Control Layer. For the layers that come above it, it serves to abstract away lower level details such as the internal robot model, kinematics and dynamics. To preserve generality and portability, the Robot Control node is extended via a plugin scheme to implement the hardware interface. Two hardware interfaces have been developed within
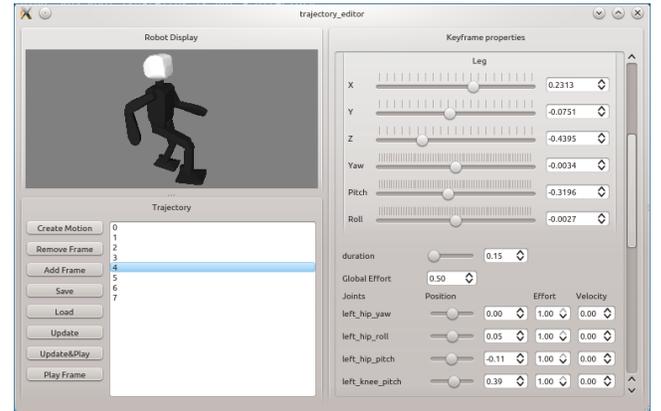
the scope of the NimbRo-OP project, one to implement the communication with the actual robot, and one to simulate such communications with an adjustably noisy and time-delayed model. The latter, referred to as the *dummy interface*, is advantageous for development in that code can be tested without risking damage to the robot.

Robot Control largely uses the RBDL library [9] for its kinematic and dynamic calculations. In particular, the RBDL Newton-Euler inverse dynamics algorithm is used to estimate the joint torques required for a particular motion. These estimates are passed to a compliant servo controller, based on a detailed model of the servos [10], that ensures that the robot remains compliant while still following the target trajectories. This servo control architecture is shown in Figure 4.

To avoid latencies, missed cycles and/or jitter, which can lead to undesired robot behavior, the Robot Control loop was designed to be as CPU and resource efficient as possible. Real-time performance of this node is paramount, so it is given real-time privileges and uses the Linux high-precision *timerfd* API to ensure that timing constraints are met. The timer period is nominally 8 ms, giving rise to a nominal 125 Hz update rate.

B. Motion Modules

In addition to the plugin scheme implemented by Robot Control to determine the robot hardware interface, a second

Fig. 6. A snapshot from the NimbRo-OP camera showing the detected features and the distortion induced by the fisheye lens.

Fig. 7. Color calibration tool used to generate YUV color lookup tables.

plugin scheme was implemented in the same node to import the actual motion code in a modular way. A plugin of this second type is referred to as a *motion module*. As opposed to being hard-coded, the definition of which motion modules to load is provided at launch time, allowing for greater flexibility and avoiding unnecessary recompilations.

The four main motion modules are the gait, head control, fall protection and motion player modules. The *gait module* implements an omnidirectional walking gait for the NimbRo-OP that is based on a central pattern generator. This gait has been used by team NimbRo for numerous other humanoid robots [11], and was ported in this case for use in ROS and the NimbRo-OP. The *head control module* is responsible for the motions of the neck servos, and ensures that the target gaze directions commanded by the behavior control layer are effectuated. The *fall protection module* monitors the robot's state and triggers the fall protection override when it detects that a fall is imminent, defined as a threshold on the robot's attitude estimate. The override causes the servos to relax and enter a free-wheeling mode, reducing the risk of damage to the gears. Prone and supine get-up motions are used by the robot to recover once the fall concludes [12]. The final module, the *motion player module*, is a keyframe motion player that uses a nonlinear keyframe interpolation approach to achieve smooth playback, and with great control. Keyframes are specified in terms of both joint positions and velocities, and are chained together to form the required motion(s) using timestamps. The interpolation scheme is based on quadratic splines and takes velocity and acceleration limits into account. An editor for these keyframe motions was created to facilitate their use, and is referred to as the *trajectory editor*. A screen shot of the trajectory editor is shown in Figure 5.

VI. PERCEPTIONS AND STATE ESTIMATION

A. Vision System

The visual perception of a robot's surroundings is a crucial element in many control systems, and in the case of the NimbRo-OP, is a necessary prerequisite for playing soccer. The NimbRo-OP is equipped with a Logitech C905 camera, custom fitted with a wide-angle fisheye lens to increase the field of view to approximately 180 degrees. This allows the robot to observe a large proportion of its environment at any one
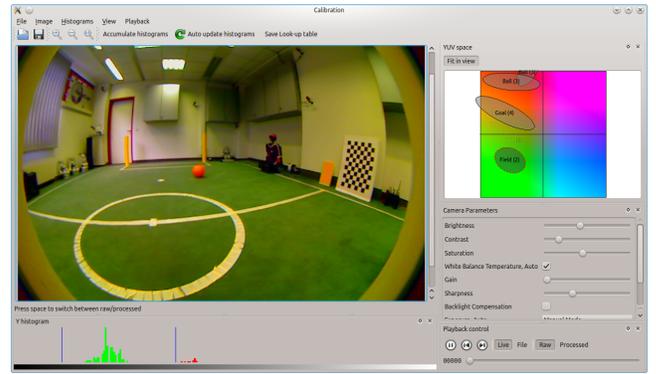
time, and reduces the need for gaze control strategies to locate objects. The Linux UVC camera driver is used to perform the image acquisition via the `camera_v4l2` plugin. Refer to Figure 3 for a schematic of the vision software architecture. The images are retrieved in YUYV format at a resolution of 800×600, and at a rate of 24 fps. Camera parameters such as brightness, contrast and saturation can be adjusted using a custom color calibration tool, shown in Figure 7. This tool is also used to tune the color lookup tables that are required for the subsequent color segmentation step (part of the vision processing), which separates the image into orange, yellow, green, white and black regions, and simultaneously subsamples the image by a factor of four. The output of this step is a collection of subsampled binary images, where each image corresponds to the pixel detections of a particular color. For the purposes of robot soccer, these subsampled images are then used to detect the extent of the field, and the required features of the environment such as obstacles, line crossings, balls and goals. Finally, the pixel coordinates of the object detections are undistorted into 3D Euclidean vectors that estimate the location of the detected objects in egocentric world coordinates. A sample image illustrating various object detections is shown in Figure 6.

Two additional tools were written for the vision system. The first is a vision GUI that can be used to inspect the raw, segmented and subsampled images, while the second is an image downscaling node that is used to throttle and compress images for publishing over WiFi. This allows images produced by the running robot to be viewed on a remote computer.

B. Localization

The task of robot state estimation can be broken down into several components. The pose of the robot—in terms of the configuration of all the body parts relative to one another—is determined from the joint encoders via a kinematic model of the robot and the corresponding forward kinematics. The encoders are located directly on the joint axles, so they already incorporate any servo non-linearities, in particular backlash, into their returned measurements. The robot torso attitude is estimated based on 3D accelerometer and gyroscope measurements by means of an angle estimation filter. This has been demonstrated to provide a low noise, low latency and drift-free attitude estimate. What remains is the requirement for an estimate of the global position of the robot in its environment,

in this case a soccer field. For this purpose, a particle filter is used that processes the landmark detections from the vision node, namely the detected white line crossings and goal posts, as well as the compass readings. The implementation of localization on the NimbRo-OP was ported from the work in [13], and functions satisfactorily, but is currently only in a beta stage. Future work includes the consideration of the landmark orientations and the actual white line segments, as well as general performance tuning of the particle filter.

VII. CONFIGURATION SERVER

Due to the large number of parameters in the NimbRo-OP software, it is highly unpractical or even infeasible to configure and tune the performance of the robot if these parameters are incorporated in the source code as hard-coded constants. Modifying hard-coded constants requires recompilation and redeployment to the robot, potentially a very time-consuming process, and does not allow for runtime changes to parameters. To address this issue, the *configuration server* was developed. Implemented as a ROS node, the configuration server is a centralized manager and storage location for system parameters. It is similar in idea to the native ROS parameter server, but with many crucial improvements that make it suitable for use in the NimbRo-OP software framework. In particular, much effort was put into ensuring that the runtime reconfigurability of the parameters is well supported, and that parameters can be systematically shared between nodes. The existing `dynamic_reconfigure` ROS package, which extends the ROS parameter server, was insufficient for the task as it did not allow the reconfigurable parameters to be shared globally between the various nodes of the system. The parameters are stored in a hierarchical structure in the server, with local copies of the relevant parameters being kept in each of the nodes for performance reasons. The appropriate nodes are notified via service calls whenever a parameter is changed on the server. Appropriate functionality exists to be able to load and save the entire parameter hierarchy. The default parameter values on system startup are taken from a well-defined configuration file, stored on each robot.

VIII. BEHAVIOR CONTROL

A. Behavior Frameworks

In order to perform tasks such as the playing of soccer, the NimbRo-OP needs to be able to be programmed with complex agent behaviors. Here, the term *behavior* is used to refer to an observable and coordinated pattern of activity. As the behaviors constitute the higher-level control of the robot, it is generally desired that such behaviors operate on a suitably abstracted view of the environment, allowing the lower-level layers, such as the Motion Control Layer described in Section V, to take care of the specifics of the walking and servo control aspects for example. Two generic cross-platform C++ frameworks were developed for the implementation of behaviors on the NimbRo-OP robot, the State Controller Library and the Behavior Control Framework [14]. The former is a state-based approach and generalizes the concept of finite state machines to multi-action planning derivatives thereof, while the latter is a behavior-based approach that exploits a hierarchical structure and the concept of inhibitions to allow for dynamic transitioning.

B. Soccer Behaviors

The soccer behaviors of the NimbRo-OP are based on the Behavior Control Framework, with the State Controller Library being used in a supporting role to implement finite state machines within the individual behaviors if and as required. As the lower-level motions are already handled by the Motion Control Layer, the soccer behaviors were split into only three hierarchical *behavior layers*, the soccer layer, the control layer, and the ROS interface layer. Each non-interface layer contains a pool of robot behaviors that compete dynamically via user-defined inhibitions for control of the robot. The control layer contains skill-oriented behaviors such as *Dribble*, *Kick* and *Go Behind Ball*, the execution of which are resolved at runtime based on the game state and the perceptions of the robot. The soccer layer on the other hand, contains behaviors of even greater abstraction, including in particular *Play Soccer*. The purpose of the soccer layer is to accommodate the inclusion of additional behaviors, that can address any auxiliary aspects of the soccer game, such as the technical challenges at RoboCup. The ROS interface layer is a special behavior layer with no child behaviors. Instead, it handles the communication of the input and output data of the behavior node over the ROS topics and services. As this allows the implementation of the remaining two behavior layers to be ROS-independent, this makes the behaviors more portable to other systems.

In terms of behaviors, the NimbRo-OP is currently capable of searching for the ball, walking behind it while taking into account any perceived obstacles, dribbling it towards the target half of the field, and kicking the ball towards the goal. Additional behaviors of the control layer that execute in parallel to this include a game control behavior that processes the signals of the RoboCup game controller during a match, and a head control behavior, which ensures that the robot's camera is always pointing in a suitable direction during the game. It is to be noted however, that despite all that has been achieved in terms of behavior control so far, there will be more advanced behaviors to come in the future software releases.

IX. SYSTEM VISUALIZATION

Visualizations are indispensable tools at all stages of the code development cycle, in particular for the purposes of debugging and testing. They can also relay valuable state information to the operator during normal runtime situations. The visualization tools for the NimbRo-OP are all implemented as widgets for the ROS-standard RQT graphical user interface (GUI), and can be seen in Figure 8. The most fundamental of these widgets is the RViz plugin, which provides a highly customizable 3D visualization environment within which the robot can be displayed. The robot is shown in either its current or desired pose, as specified by its joint angles, and at the global location it currently thinks it is at. A virtual soccer field environment can be enabled to augment the view. Many additional features can be displayed in the 3D virtual environment, including custom point markers, line traces, footstep locations, servo torques, link coordinate frames, and object detections such as ball, goal, obstacle and landmark detections. These allow the user to get a quick visual insight into the state of the robot, and detect any obvious problems or discrepancies.

The diagnostics widget, the leftmost widget in Figure 8, displays the robot's current battery voltage level and maximum

servo temperature. It also displays the current motion state, as defined in Robot Control and the motion modules, and allows the user to fade the servos in and out. The walk control widget, located above RViz in the figure, allows the user to issue manual gait commands and otherwise control the gait module, provided that the currently executing behavior code is not writing values of its own. This is useful for instance, for the testing of the Motion Control Layer. The parameter tuner widget allows the user to view, organize, modify, load and save parameters of the configuration server. Numeric parameters are represented and can be modified using sliders and spin boxes, while checkboxes are used for boolean parameters. This permits the user to manually access the configuration server and modify aspects of the system at runtime, either for testing or tuning purposes.

The final visualization that can be seen in Figure 8 is the plotter widget. This is a plugin that can be used to trace and plot the time history of any nominated variables throughout the entire framework code. The node in which the variable appears just needs to publish its value to the plotter, where it is automatically placed into a circular buffer and made available for display to the user. The plotter variables are stored in a tree hierarchy, which is presented to the user so that a selection of variables to plot can be made. An important feature of the plotter is that the mouse cursor can be used to view the robot's state at a past time. This affects what is shown in the 3D visualization. The RQT tool had to be modified in order to allow the negative time changes that are observed when the user scrolls back in time. This extension to RQT is encapsulated in the so-called *timewarp* package. Finally, at any point in time the plotted data can be flushed to a ROS bag using the save feature of the widget.

X. CONCLUSION

In this paper a ROS-based software framework was presented for the NimbRo-OP robot, a humanoid open platform in development by team NimbRo of the University of Bonn. This is the second software release to be undertaken for this robot, but the first to feature the ROS middleware. All dependencies and inclusions of the original Robotis software framework have been rewritten and/or removed. The decision to release the framework open source was so as to support collaboration and community-based improvements.

The NimbRo-OP ROS framework has been demonstrated publicly on numerous occasions, including at ICRA 2013 and RoboCup 2013, where it scored its first official soccer competition goal. The framework was also demonstrated in an earlier form at the RoboCup German Open 2013, where the NimbRo-OP received the HARTING Open-Source Award. Work on the NimbRo-OP is still an ongoing project however, so accordingly the ROS framework presented in this paper is expected to remain subject to further revisions and additions. Future work on the framework includes the incorporation of a multi-hypothesis Kalman filter for object filtering and tracking, the integration of a physics-based simulator for testing, and the development of more advanced soccer behaviors. Additional goals include the development of a more advanced gait, and the tuning of the localization for improved robustness.
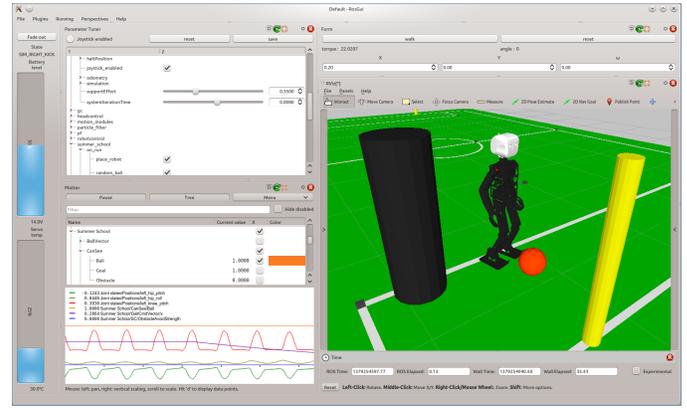

Fig. 8. Screenshot of the RQT visualization tool, showing the RViz, walk control, plotter, parameter tuner and diagnostics widgets.

REFERENCES

[1] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of NAO humanoid," in *Int. Conf. on Robotics and Autom. (ICRA)*, 2009.

[2] I. Ha, Y. Tamura, H. Asama, J. Han, and D. Hong, "Development of open humanoid platform DARwIn-OP," in *SICE Annual Conf.*, 2011.

[3] NimbRo. (2013, Oct) NimbRo-OP ROS Software Release. [Online]. Available: https://github.com/NimbRo/nimbro-op-ros

[4] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[5] ROS on the HUBO Humanoid Robot. [Online]. Available: http://wiki.ros.org/Robots/HUBO

[6] U. R. Team. Ros Fuerte cross-compiling and installation for the NAO v4. [Online]. Available: http://www.ros.org/wiki/nao/Tutorials/Cross-Compiling_NAO-V4

[7] M. Schwarz, J. Pastrana, P. Allgeuer, M. Schreiber, S. Schueller, M. Missura, and S. Behnke, "Humanoid TeenSize Open Platform NimbRo-OP," in *Proceedings of 17th RoboCup International Symposium*, Eindhoven, Netherlands, 2013.

[8] M. Schwarz, M. Schreiber, S. Schueller, M. Missura, and S. Behnke, "NimbRo-OP Humanoid TeenSize Open Platform," in *7th Workshop on Humanoid Soccer Robots at Humanoid Conference, Osaka*, 2012.

[9] M. Felis. (2013, August) Rigid Body Dynamics Library. [Online]. Available: http://rbdl.bitbucket.org/

[10] M. Schwarz and S. Behnke, "Compliant robot behavior using servo actuator models identified by iterative learning control," in *Proceedings of 17th RoboCup Int. Symposium*, Eindhoven, Netherlands, 2013.

[11] S. Behnke, "Online trajectory generation for omnidirectional biped walking," in *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, Orlando, USA, 2006.

[12] J. Stückler, J. Schwenk, and S. Behnke, "Getting back on two feet: Reliable standing-up routines for a humanoid robot," in *Int. Conf. on Intelligent Autonomous Systems (IAS-9)*, Tokyo, 2006, pp. 676–685.

[13] H. Schulz and S. Behnke, "Utilizing the structure of field lines for efficient soccer robot localization," *Advanced Robotics*, vol. 26, no. 14, pp. 1603–1621, 2012.

[14] P. Allgeuer and S. Behnke, "Hierarchical and state-based architectures for robot planning and control," in *Proceedings of the 8th Workshop on Humanoid Soccer Robots at Humanoids Conference*, Atlanta, 2013.