FUmanoids Code Release 2012

Daniel Seifert and Raúl Rojas

Institut für Informatik, Arbeitsgruppe Intelligente Systeme und Robotik,
Freie Universität Berlin, Arnimallee 7, 14195 Berlin, Germany
{dseifert,rojas}@inf.fu-berlin.de
http://www.fumanoids.de

Abstract. Code re-use and sharing between teams are important factors in the advancement of a RoboCup league. This paper presents the code release of the Humanoid League KidSize team *Berlin United - FUmanoids*. We describe the underlying frameworks and their design principles. Abstraction of hardware and independence of the platform allow the use for different robot types and usage scenarios.

1 Introduction

In this paper we describe the software release of the Humanoid League (HL) KidSize team *Berlin United - FUmanoids*. The first section covers the motivation for code releases in RoboCup, as well as a short history of our team. Section 2 discusses the current code release and its main features, as well as plans for the next code release. Relevant design decisions, the overall architecture and the architecture of the underlying frameworks are presented in section 3. Section 4 covers the debug and analysis tool *FUremote*. Finally, section 5 provides an overview of the use of the release and offers some conclusions.

1.1 Motivation for Code Release

Interoperability between teams in a RoboCup league is an important factor for the advancement of the league. Being able to start with a working system allows new participants to quickly focus on their areas of research. And having the possibility to integrate another team's code allows teams to accelerate their progress without having to reinvent the wheel.

The benefits of interoperability are evident in the Standard Platform League (SPL), where each team has the same robots and several teams regularly provide full releases of their code. A notable example is B-Human, who won first place in 2009-2011. They provide their source code on a yearly basis including an extensive team report [10,11]. As a result of being able to share code easily, the SPL was able to strongly push their rules towards the 2050 goal by increasing team size, reducing color-dependency and enlarging their playing field [9].

On the other hand, teams in the HL usually build their own robots, or at least modify existing platforms. Significant time is spent on hardware development and a team's software is usually incompatible with other teams' robots due to

S. Behnke et al. (Eds.): RoboCup 2013, LNAI 8371, pp. 600–607, 2014.

the close ties to the specific hardware platform. This severely slows down progress in the HL. Only in the last two years have the release of the DARwIn-OP [2] and the recent release of the NimbRo-OP [13] provided means to build upon the experience of established teams more easily.

1.2 Berlin United - FUmanoids

Berlin United - FUmanoids has been participating in the Humanoid KidSize League since 2007. Since 2010, the team has cooperated with the SPL team *Nao Team Humboldt*, resulting in the multi-league joint-research group *Berlin United*.

In the first year of the team's existence, a commercially available robot platform was used. In 2008 this platform was slightly modified and finally completely replaced by a fully custom-built robot in 2009 [8]. Since then, a variety of robot models have been designed and successfully[1] used in RoboCup competitions.

Over time, it became obvious how important code sharing is and how little has happened in the Humanoid League in this regards. As a result, we decided to release our code after RoboCup 2011. The first release was published on the team's website in December 2011, thus being one of the first and few code releases in the league. In December 2012, the updated code from RoboCup 2012 was released [1].

2 Current State and Outlook

Our release includes the sources for *FUmanoid* and *FUremote*, a Java application handling data display, debugging and many additional tasks (section 4).

FUmanoid is the robot control software which runs on the robots. At its heart is a modular framework developed by *Berlin United - NaoTH* [5]. It is incorporated as part of the *Berlin United Framework*, consisting of hardware and operating system abstractions as well as a number of service classes. In the current code release [1] the *Berlin United Framework* has not yet been separated. However a standalone version is available online [1]. Utilizing both frameworks, modules separated into cognition and motion blocks are responsible for handling robot specific tasks.

Section 3 highlights the technical aspects of the *FUmanoid* source code. It is the result of several years of development which gradually changed the code from being highly specific for the team's robots to a more generic approach. This has allowed the *Berlin United - Racing Team*[2] to use the software as the basis for their autonomous RC car. The *Berlin United Framework* will also be used in two other robots at our working group, namely in a quadcopter and in a robot participating in the DLR's SpaceBot Cup, supporting inter-project exchange of code and knowledge.

[1] The team won 3rd place in 2007, was runner-up in 2009 and 2010, and placed 4th in 2011.

[2] The *Racing Team* is a student project participating in the CaroloCup competition, where an autonomous RC car has to drive on a modelled street and park itself.

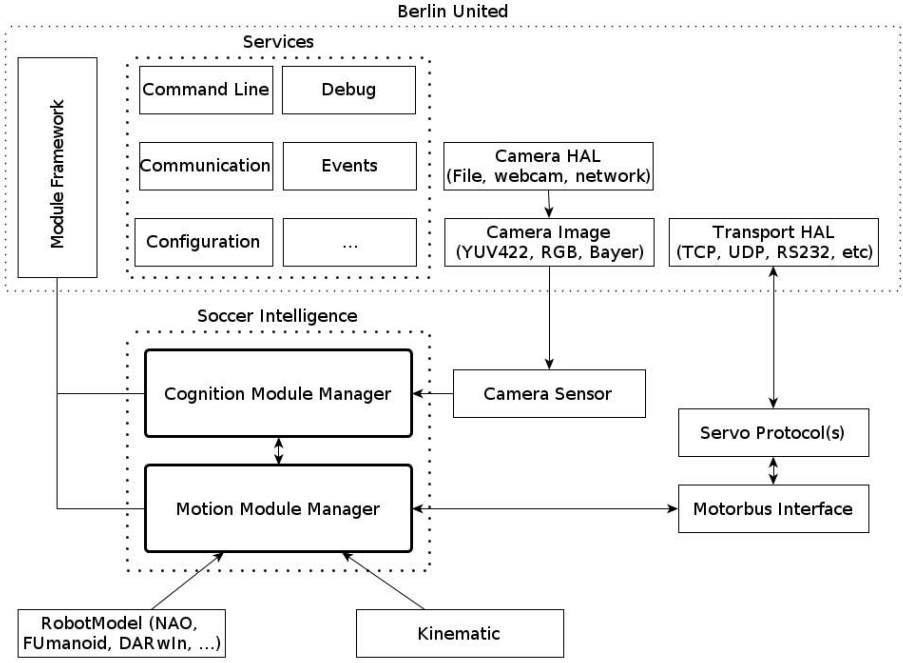

Fig. 1. Overall structure of the control software

For the next release, we are actively working on additional hardware abstractions and increased functionality of the framework. Development of a generic actuator interface is almost finished. It will add full support for the SPL's NAO robot as well as the DARwIn-OP robot to our software. Work has also started on proper documentation outside the source code to aid new students to get a better overall grasp of the system as well as to allow new projects to easily adapt the source code to their needs.

3 Robot Control Architecture

The robot control software *FUmanoid* presented in this section consists of multiple parts. Figure 1 gives an overview of the important components.

3.1 Module Framework

The *NaoTH Module Framework* [5,6] is a clean standalone implementation of a blackboard architecture. The historical roots go back to the GermanTeam's module framework [12], which results in some similarities of the concepts.

The blackboard holds and manages instances of objects, which are *representations* of data that *modules* can either *require* (read) or *provide* (write). Read

and write access to the blackboard objects is granted based solely on the declared requirements of the module. The resulting dependency chain permits to automatically calculate an execution order of the modules. As modules are independent from each other they can be exchanged easily, which supports to have concurrent solutions for the same problem.

The modules are executed serially and access data through their module manager's dedicated blackboard only. Thus, concurrency issues are avoided without the overhead of using critical sections throughout the code.

The framework simplifies the development process, especially for new students joining the team. With this approach, no extensive knowledge of the whole software stack is required. Instead work can focus on a single, contained module.

3.2 Multi-platform and Multi-sensor Support

A core design principle of the code in the last few years was the requirement to support different hardware platforms and sensors. This principle arose naturally out of the yearly changing hardware of the team. Additionally it became necessary to support testing the software outside the robots to accelerate development and to improve software quality.

Rather than providing a platform interface which acts as a hardware abstraction layer (HAL) to a pre-defined platform (e.g. a certain robot model), our software is heavily based on the concept of exchangeable input and output devices, i.e. sensors and actuators. Instead of a single HAL, there are specific hardware abstraction interfaces for specific types of sensors and actuators. Being able to easily combine these different sensors allows to quickly adjust to changes in the platform or to assemble new platforms based on pre-existing components. Furthermore, sensor information can be retrieved from multiple sources[3]. Representations can be pre-filled with ground truth data aquired via e.g. simulators[4].

Currently the software only supports robots running the Linux operating system. As CPU-specific functionality is avoided, the code works on both ARM and x86 based CPUs. Due to the multi-sensor approach the code also runs seamlessly on Linux desktop computers. However, supporting other common operating systems would be easy. All Linux-specific components are in wrapper classes which can be easily extended if required.

3.3 Berlin United Framework

The *Berlin United Framework* incorporates various service classes as well as the module framework discussed in 3.1. All functionality included in this framework is independent of the type of robot used, and as such can be used as the basis for

[3] One example would be to mix sensors from an actual robot and a simulated one.

[4] The code release includes support for our multi-level sensor emulator *Sim** [3,7], which is not part of the code release due to limited resources for its continued development. However, support for the Simspark simulator has been added and will be available in the next release of the code.

one's own robot software. Standard hardware functionality is included but not mandatory to use. The service classes offer all common functionality a mobile platform requires in an easy to use way to simplify development. For example:

- A dedicated thread handles incoming and outgoing **communication**, supporting multiple remote clients. All communication is based on Protobuf[5], which is an efficient [14] data serialization format that can be extended easily and accessed with a variety of computer languages. Modules are able to register for specific incoming Protobuf messages.
- **Configuration** parameters can be defined in the code and stored and retrieved from a Protobuf-serialized file or via network (ref. section 4), as well as set via the command line.
- An extensive **debug** interface allows the registration of debug messages of several types, including text, table, 2D plot and various image representations. These can be selectively enabled. While disabled, run-time overhead is negligible. Debug output, including images, can be streamed to several remote instances, allowing multiple users to debug or inspect different aspects of the robot simultaneously.
- **Log files** can be created, consisting of all or select representations from each execution cycle. These log files can be used in subsequent runs to populate representations, allowing to debug modules or compare algorithms against each other for the same input data set.

3.4 Motion and Cognition

So far, we discussed the generic capabilities included in our code release. However the code release also includes the HL specific soccer code of our team. We separated our modules into a cognition and a motion block (figure 2).

The *Cognition* module manager is executed when a new camera image is received and thus runs at most 30 times per second. At first the image processing modules run and retrieve information like the position of the ball, the goal and the obstacles from the image, as well as line segments. This data is then classified, e.g. field lines and field line features (crossings, etc) are extracted. Various modelling modules try to model the world and finally behavior modules based on XABSL [4] decide on the actions the robot should take. This sequence follows the Sense-Think-Act paradigm.

The *Motion* module manager runs at 100 Hz. It is responsible for hardware control, primarily interfacing with sensors (except camera) and actuators. An execution cycle starts with reading out sensors and actuators and managing the various information to handle sensor fusion and related tasks, e.g. calculation of camera height based on the robot's kinematics. Depending on the type of motion requested by the cognition, modules are selected which perform keyframe motions like standing up, or which calculate and set the walking trajectories.

Communication between the two module blocks is achieved by event notifications. At the end of each manager's cycle, specific representations are sent and

[5] `https://code.google.com/p/protobuf`

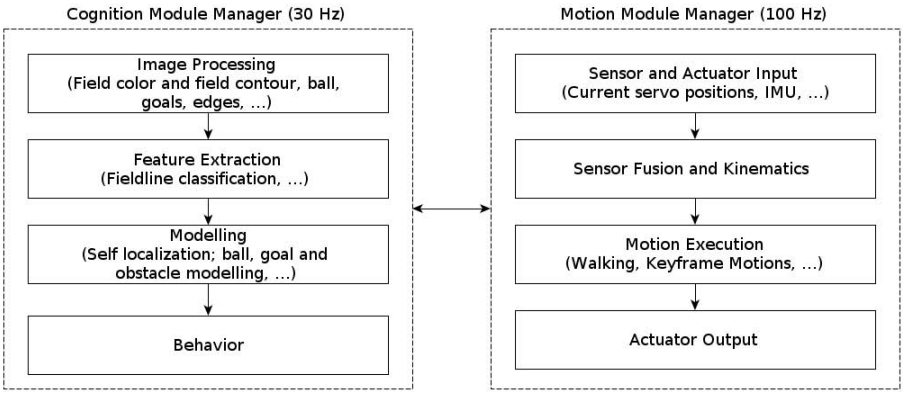

Fig. 2. Core Tasks in the Module Managers

the other manager injects them into its own representations at the next execution cycle. This is a simple yet effective method of sharing data and distributing commands with negligible overhead.

4 Remote Control and Debugging

The code release also includes our debug and control tool *FUremote*. It is based on Eclipse RCP and provides most of its features by means of plugins which allows to switch out features or extend functionality easily, for example when the software is used by other projects.

For the most part, *FUremote* is developed in a way that is agnostic to the specific hardware and environment being used. Plugins retrieve the relevant information from the robots, e.g. the list of debug options, the configuration keys or the available camera settings. This makes it a versatile tool supporting changes to the robot's hardware and software.

Figure 3 shows a possible setup of the tool. On the top-left, the robots currently online are shown including relevant status information like battery level. Below, the Debug plugin lists the available debug options, four of which are active and displaying information in various ways.

Other notable plugins available allow to configure the camera settings, modify configuration parameters, change the robot's game state, visualize the behaviour status of the robot, interact with the servos and record and preview keyframe motions.

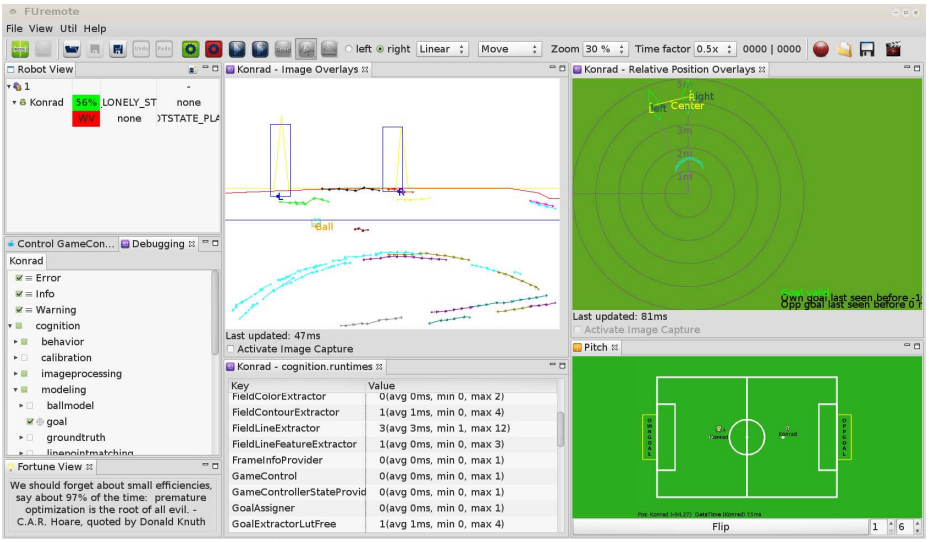

Fig. 3. Screenshot of the *FUremote* tool

5 Conclusion

We presented our code release consisting of the *FUremote* tool, as well as *FUmanoid* and its frameworks. The *Berlin United Framework* is actively used in our humanoid RoboCup team as well as in an autonomous RC car. It will also be used for a quadcopter and a robot participating in the SpaceBot Cup. Its platform-independence and modularity makes it suitable for a variety of robots within and outside the RoboCup community. We expect increased impact when the NAO und DARwIn-OP support is fully integrated in the next code release.

The code release can be downloaded at `http://www.fumanoids.de/code`.

Acknowledgements. The code released is based on many years of development by the members and students of the *Intelligent Systems and Robotics* group at Freie Universität Berlin. More information about the team and its members, as well as publications and details to the robots is available online at `http://www.fumanoids.de`.

References

1. FUmanoids: FUmanoids Code Release (2012), `http://www.fumanoids.de/publications/coderelease`
2. Ha, I., Tamura, Y., Asama, H., Han, J., Hong, D.: Development of open humanoid platform DARwIn-OP. In: Proceedings of SICE Annual Conference, SICE 2011, pp. 2178–2181 (2011)

3. Heinrich, S.: Development of a Multi-Level Sensor Emulator for Humanoid Robots. Diploma thesis, Freie Universität Berlin, Institut Mathematik und Informatik, Deutschland (2012)
4. Loetzsch, M., Risler, M., Jungel, M.: Xabsl - a pragmatic approach to behavior engineering. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5124–5129 (2006)
5. Mellmann, H., Xu, Y., Krause, T.: Common Platform Architecture - A Simple and Clean Architecture for Participation in SPL and Simulation 3D (2011), http://www.naoth.de/projects/multi-platform-robot-controller
6. Mellmann, H., Xu, Y., Krause, T., Holzhauer, F.: NaoTH Software Architecture for an Autonomous Agent. In: Proceedings of the International Workshop on Standards and Common Platforms for Robotics (SCPR 2010), pp. 316–327 (2010)
7. Mielke, S.: Kamerabildrekonstruktion der Simulationsumgebung für humanoide Fußballroboter. Diploma thesis, Freie Universität Berlin, Institut für Mathematik und Informatik, Deutschland (2012)
8. Mobalegh, H.: Development of an Autonomous Humanoid Robot Team. Ph.D. thesis, Freie Universität Berlin (2011)
9. RoboCup SPL Technical Committee: RoboCup Standard Platform League (Nao) Rule Book (2013), http://www.tzi.de/spl/pub/Website/Downloads/Rules2013.pdf
10. Röfer, T., Laue, T., Müller, J., Bartsch, M., Batram, M.J., Böckmann, A., Lehmann, N., Maß, F., Münder, T., Steinbeck, M., Stolpmann, A., Taddiken, S., Wieschendorf, R., Zitzmann, D.: B-Human Team Report and Code Release 2012 (2012), http://www.b-human.de/wp-content/uploads/2012/11/CodeRelease2012.pdf
11. Röfer, T., Laue, T., Müller, J., Fabisch, A., Feldpausch, F., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Humann, A., Honsel, D., Kastner, P., Kastner, T., Könemann, C., Markowsky, B., Riemann, O.J.L., Wenk, F.: B-Human Team Report and Code Release 2011 (2011), http://www.b-human.de/downloads/bhuman11_coderelease.pdf
12. Röfer, T., Brose, J., Göhring, D., Jüngel, M., Laue, T., Risler, M.: GermanTeam 2007 - The German national RoboCup team. In: RoboCup 2007: Robot Soccer World Cup XI Preproceedings, RoboCup Federation (2007)
13. Schwarz, M., Schreiber, M., Schueller, S., Missura, M., Behnke, S.: NimbRo-OP Humanoid TeenSize Open Platform. In: IEEE-RAS International Conference on Humanoid Robots Proceedings of 7th Workshop on Humanoid Soccer Robots (2012)
14. Sumaray, A., Makki, S.K.: A comparison of data serialization formats for optimal efficiency on a mobile platform. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC 2012, pp. 48:1–48:6. ACM, New York (2012), http://doi.acm.org/10.1145/2184751.2184810