B-Human

Team Description for RoboCup 2017

Thomas Röfer[1,2], Tim Laue[2], Andre Mühlenbrock[2]

[1] Deutsches Forschungszentrum für Künstliche Intelligenz,
Cyber-Physical Systems, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
[2] Universität Bremen, Fachbereich 3 – Mathematik und Informatik,
Postfach 330 440, 28334 Bremen, Germany

1 Introduction

B-Human is a joint RoboCup team of the University of Bremen and the German Research Center for Artificial Intelligence (DFKI). The team was founded in 2006 as a team in the Humanoid League, but switched to participating in the Standard Platform League in 2009. Since then, we participated in eight RoboCup German Open competitions, the RoboCup European Open 2016, and in eight RoboCups. We always won the German/European Open and became world champion five times.

This team description paper is organized as follows: In Sect. 2, we describe our new detection of the orientation of other robots. In Sect. 3, we explain the improvements we made to UNSW Australia's walking engine to integrate it into our system. Section 4 describes the approach of our new fall detection. Section 5 introduces an important change we made to our software framework. In Sect. 6, we describe our efforts to improve the infrastructure of the league this year. Finally, Sect. 7 summarizes this paper.

1.1 Team Members

B-Human consists of the following people, most of whom are shown in Fig. 1:

Team Leaders / Staff: Tim Laue, Thomas Röfer.
Students: Yannick Bülter, Daniel Krause, Jonas Kuball, Lam Duy Le, Andre Mühlenbrock, André Osse, Alicia Susan Pagel, Bernd Poppinga, Lukas Post, Markus Prinzler, Enno Röhrig, Jurij Schmidt, René Schröder, Markus Strehling.
Alumni: Florian Maaß, Judith Müller, Jesse Richter-Klug, Andreas Stolpmann, Alexander Stöwing, Felix Thielke, Alexis Tsogias.
Associated Researcher: Udo Frese.

Fig. 1. The B-Human team after winning the RoboCup German Open 2017

1.2 Publications Since RoboCup 2016

As in previous years, we released our code after the RoboCup 2016 together with a detailed description [6] on our website and on GitHub (`https://github.com/bhuman/BHumanCodeRelease`). Up to date, we know of 23 teams that based their RoboCup systems on one of our code releases (AUTMan Nao Team, Austrian Kangaroos, BURST, Camellia Dragons, Crude Scientists, Edinferno, GraceBand, JoiTech-SPL, Luxembourg United, Nao Devils, NimbRo SPL, NTU Robot PAL, SPQR, UChile, Z-Knipsers/nomadZ) or used at least parts of it (Cerberus, MRL SPL, Northern Bites, NUbots, RoboCanes, RoboEireann, TJArk, UT Austin Villa).

We also described our approaches to image processing under varying lighting conditions, detecting the black and white ball, and combining simple field features such as lines to more complex ones (e. g. oriented center circle, corners, penalty areas) and their use for self-localization in our 2016 Champion Paper [4]. Our C-based Agent Behavior Specification Language (CABSL) is used by several teams in the Standard Platform League, usually if they base their systems on our code release anyway. To simplify its use in other environments, we now also provide a standalone release with an extensive example [5].

2 Detecting the Orientation of Other Robots

Knowing the positions of the other players on a football pitch is a crucial aspect for playing successfully. However, knowing not only the position but also the

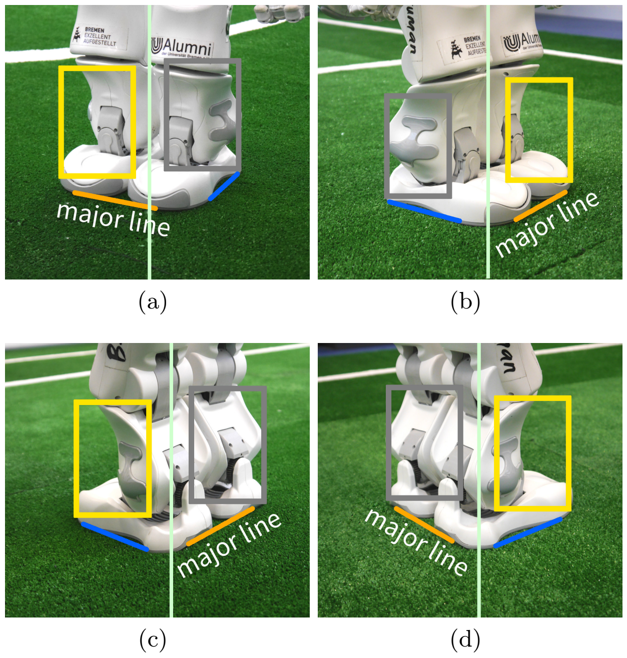


Fig. 2. Major features for orientation detection: The location of the *major line* (orange) and the area that contains more green pixels (yellow). (a/b) The major line and the yellow marked area are on the same side when the robot is facing forwards and (c/d) they are on a different side when the robot is facing backwards. The blue line depicts the *minor line*.

orientation of another player provides certain tactical advantages. We developed an approach that uses the alignment of robot feet to compute orientations, as this feature turns out to be quite robust, in contrast to a robot's upper body, which might have a complex appearance due to moving arms and a jersey of almost arbitrary style.

The approach consists of two steps which both analyze the region around a previously recognized robot's feet. First, we search for two lines – the so-called *major line* and *minor line* – in this region. The major line is defined from toe to toe and from heel to heel while the minor line is defined as a side line of a foot. These two lines already allow calculating an orientation in the range of $[-90°, 90°)$. To obtain the complete orientation in the range of $[-180°, 180°)$, we determine whether the robot is facing forwards or backwards by analyzing two areas in the color classified image over the feet. An example is shown in Fig. 2. Both steps turn out to work quite reliably. However, by focussing on the feet, the approach is not able to handle lying robots or robots that have a ball directly at their feet.

The major benefits of our approach are its ability to determine orientations even over distances of more than two meters, which is necessary as robots might walk quite fast, as well as its computational efficiency, which is important as the available computing time has always to be shared with many other components. During several experiments, the longest execution time was 0.53 ms.

3 Integration of UNSW Australia's Walking Engine

In 2016, B-Human participated in the Outdoor Competition. Although we still became the runner-up, it was quite obvious that the walk that we have used since 2012 is not well-suited for artificial grass. Another walk that was very successful in recent years is the one developed by Bernhard Hengst [1] for the team UNSW Australia/rUNSWift. It won the RoboCup 2014 and 2015 and reached the final in 2016 as part of UT Austin Villa's system. It also appeared to work quite well during the Outdoor Competition.

The Walk2014, as the walk is called, is based on three major ideas:

1. As in all walks, the body swings from from left to right and back during walking to shift away the weight from the swing foot, allowing it to be lifted above the ground. In contrast to many other walks, this is achieved without any roll movements in the leg's joints (unless walking sideways), i. e. the swing foot is just lifted from the ground and set back down again, which is enough to keep the torso in a pendulum-like swinging motion.
2. As a result, there is a clearly measurable event, when the weight of the robot is transferred from the support foot to the swing foot, which then becomes the new support foot. This event is detected by the pressure sensors under the feet of the robot. In the moment this weight shift is detected, the previous step is finished and the new step begins. This means that a transition between a step and its successor is not based on a model, but on a measurement, which makes the walk quite robust to external disturbances.
3. During each step, the body is balanced in forward direction by the support foot. The approach is very simple but effective: the lowpass-filtered measurements of the pitch gyroscope are scaled by a factor and directly added to the pitch ankle joint of the support foot. As a result, the faster the torso turns forward, the more the support foot presses against this motion.

Improvements. In contrast to some other teams that integrated the Walk2014, we only use parts of its core implementation from the class `Walk2014Generator`. We have removed everything not related to walking from that class, refactored the code, converted it into a B-Human module, integrated B-Human data types, and integrated our in-walk kicks. We also replaced the inverse kinematics by our own, which results in a different rotational behavior. In the original implementation, the inverse kinematics is first solved for the legs ignoring any yaw rotation of the feet. Then, the yaw rotation is added as a rotation around the hip yaw-pitch joint (which is not entirely correct). Afterwards, an iterative algorithm is used to return the feet to the positions they had before the rotation was added. Our inverse kinematics simply computes the joint angles for the 6-D positions of both feet, equally distributing the error introduced by the linked hip yaw-pitch joint to both feet. The computation is not only more precise, it is also a lot faster, as no iteration is involved. We now also use linear trajectories for forward, sideways, and rotational motions of the support foot.

One of the core ideas of the Walk2014 is that a step ends when the weight transfer happens and the next step starts from where the last step ended. The latter was only fully implemented in the Walk2014 for forward and rotational movement, but not for sideways movement and the height of the swing foot, which may not have been set down completely before the weight has already shifted. We implemented this, which results in less disturbances in situations in which a step did not behave as expected anyway. We also implemented a mode to reach a certain relative position on the field, always considering which motion is unavoidable, since both legs always have to swing back to their zero position before the robot can come to a full stop.

We use different gyroscope balancing parameters for forward and backward rotation, because the feet are shorter at the back and stability benefits from slightly more aggressive balancing in that direction. We also found that a lot of differences between different robots can be compensated by varying the balancing parameters. Typically, older robots need more aggressive balancing.

The Walk2014 assumes a fixed position of the center of mass in the torso. Since our robots sometimes take their arms behind their back to better avoid obstacles, this assumption is not valid for them. Therefore, we introduced a dynamic center of mass computation into the walk, which considers the impact of the arms' positions. We also changed the acceleration of the robot to a linear model. In addition, if the weight transition between the support leg and the swing leg is not recognized, the current speed is reset to zero, resulting in the robot accelerating again in a controlled manner afterwards. We also detect if the weight transition is not recognized repeatedly, which can happen if the robot leans against a static object, e. g. a goalpost. In that case, it starts moving sideways away from the obstacle, which frees it from the deadlock. We also recognized that robots sometimes get stuck in a lower swing frequency. This is also detected and ended through a short stand phase.

4 Fall Detection

It is important to detect when a robot falls, when it reached the ground, and in which direction it fell on the ground. Fall detection has two phases. The first one is to determine that the robot is currently falling down. The second one is to detect that the robot has reached the ground. It is important to recognize that the robot is falling to prepare it for hitting the ground. In case of our team, the head's yaw joint is turned back to its center position and the pitch joint is turned away from the fall direction. After that, the stiffness of all joints is significantly

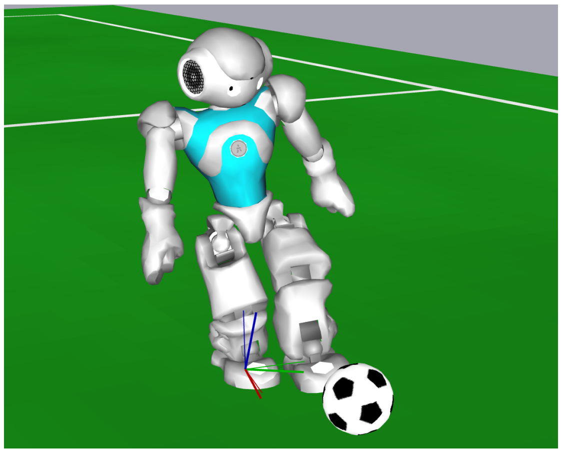


Fig. 3. The coordinate system of the support foot (thin) and the estimated coordinate system of the field plane (thick).

lowered. After the robot has hit the ground, a getup motion is started. So far, both falling and hitting the ground were simply detected through thresholds of the torso's orientation in yaw and pitch directions. This lead to several special cases for motions that changed the torso's orientation intentionally. The main innovation of our new approach is not to use the torso's orientation anymore, but the orientation of the support foot relative to the ground (cf. Fig. 3). The support foot is the foot that is extended more in the direction of gravity than the other foot. This allows detecting falling rather early independent from the motion that is currently executed. In addition, returning to the upright state can also be determined without dedicated feedback from the getup procedure by simply observing how far the torso is above the ground, i. e. the length of the support leg in the direction of gravity.

5 Framework Changes

A robot control program usually consists of several modules, each performing a certain task, e. g. image processing, self-localization, or walking. Modules require a certain input and produce a certain output (so-called *representations*). Therefore, they have to be executed in a specific order to make the whole system work. So far, the representations in the module framework introduced in [3] have only forwarded data from a module to other modules. Starting in 2017, representations can now also provide *functionality*, i. e. they can contain functions, which have an implementation that is provided by a module.

So far, all data in a representation had to be computed before it was used. This lead to a certain overhead, because the data has to be computed in advance without knowing whether it will actually be required in the current situation. In addition, there is a lot of data that cannot be computed in advance, because its computation depends on external values. For instance, a path planner must know the target to which it has to plan a path before it can be executed. However, in many situations a path planner is not needed at all, because the motion is generated reactively.

Functions in representations allow modules that require the representation to execute code of the module that provided that representation at any time and they allow to pass parameters to that implementation. For instance, while the behavior control of the 2016 B-Human system still contained many so-called libraries that could compute all kinds of information on demand as a fixed part, they are now simply representations that are provided by regular B-Human modules that implement the logic behind the interface. These modules can be switched to other implementations as all other B-Human modules can, giving a greater flexibility when improving the functionality of the system.

These functions are based on *std::function* from the C++ runtime library and the assigned implementations are usually lambda expressions.

6 Infrastructure for the League

As in previous years, we improved the infrastructure of the league. This year, this effort was financially supported by the RoboCup Federation as a project for league development.

6.1 GameController

Logging. The *TeamCommunicationMonitor* supports logging the team communication, which is an important feature to document what happens during a game. For instance, both the winners of 2016's outdoor competition Nao Devils [2] as well as the winners of the main competition B-Human [4] analyzed these logs in their publications. However, it appeared that it is quite difficult to ensure that the TeamCommunicationMonitor is running during all games on up to six different fields. Although it can continuously run in the background, only all but one logs from the RoboCup European Open 2016[3] and around half of the logs from the RoboCup 2016 [4] were successfully recorded. Therefore, the TeamCommunicationMonitor's logging feature was also integrated into the GameController, as it will always be running during official games. In contrast, the TeamCommunicationMonitor only writes log files if no GameController is running on the same machine.

Penalty Shootout. At RoboCup 2016, there were quite a number of penalty shootouts in the SPL. This is usually a risk for the competition schedule, as such games often take longer than the time allocated for them. Therefore, the technical committee decided to streamline the process of penalty shootouts. Teams

[3] spl.robocup.org/.../RoboCupEuropeanOpen2016TeamCommunicationLogs.zip
[4] spl.robocup.org/.../RoboCup2016TeamCommunicationLogs.zip

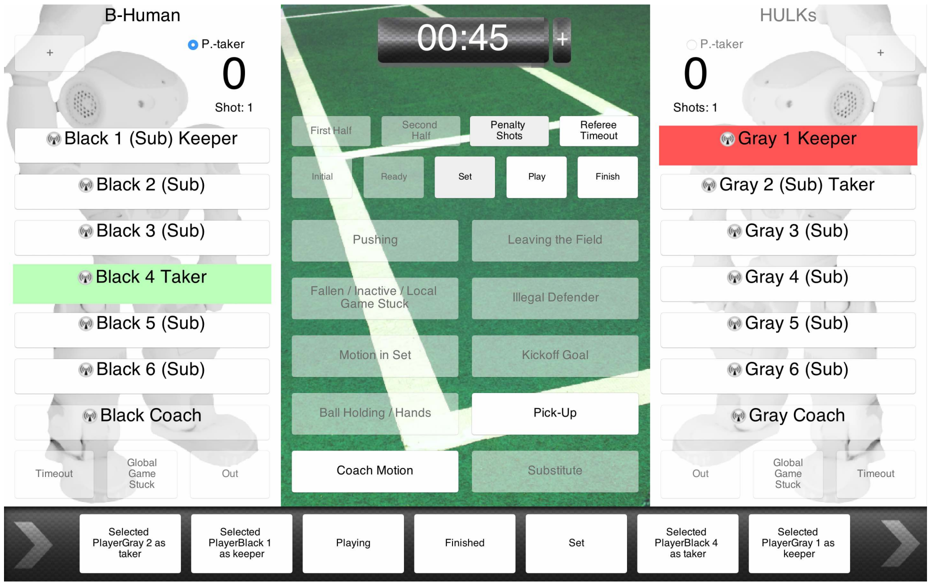


Fig. 4. The GameController during a penalty shootout. The current penalty taker is marked in green. The current goal keeper is marked in red. All other robots are substitutes. The interface also shows the robots used in the previous shot (Black 1 and Gray 2), which are the default ones for the next shot.

should now provide their penalty taker, their goalkeeper, and possible substitutes at the beginning of the penalty shootout to the referees who will have to handle the whole procedure on their own. However, until 2016, the GameController handled a penalty shoutout as an event between two teams without caring about individual robots. This means that teams had to switch off all robots that were not actually playing. The new GameController (cf. Fig. 4) allows selecting individual robots and penalizes all other robots (as substitutes). This is easier for the teams, because penalization is also part of the normal game process and should already have been implemented correctly.

Mixed Team Competition. In 2017, the SPL will have its first mixed team competition, but no Drop-in Player Competition anymore. Therefore, the support for the Drop-in Player Competition was replaced by a mode for the Mixed Team Competition. For now, the mode is identical with normal games, but with six players per team. In the future, it should also support 10 vs. 10 games.

6.2 GameStateVisualizer Mode

The TeamCommunicationMonitor visualizes the standardized part of the communication between the robots in a 3-D view. It has become quite popular in the

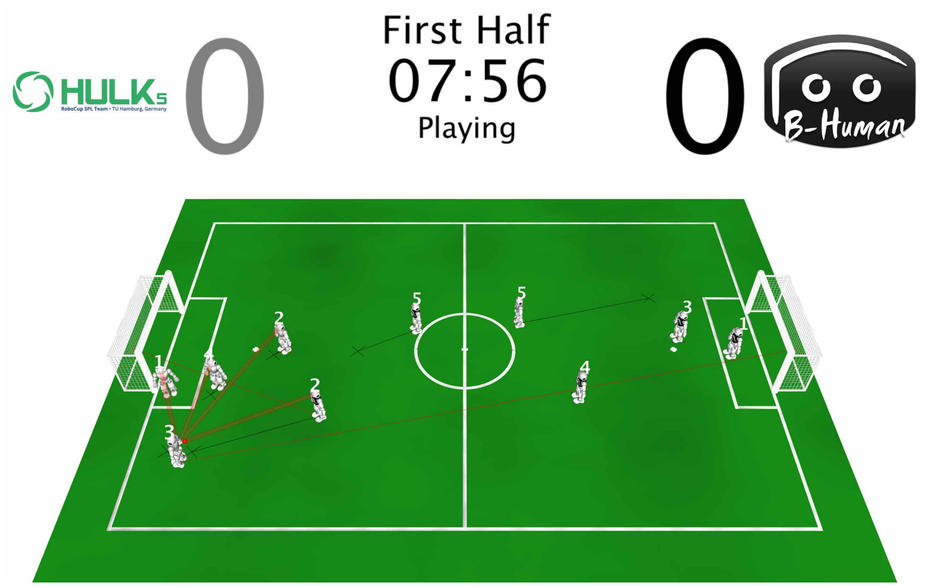


Fig. 5. The new GameStateVisualizer

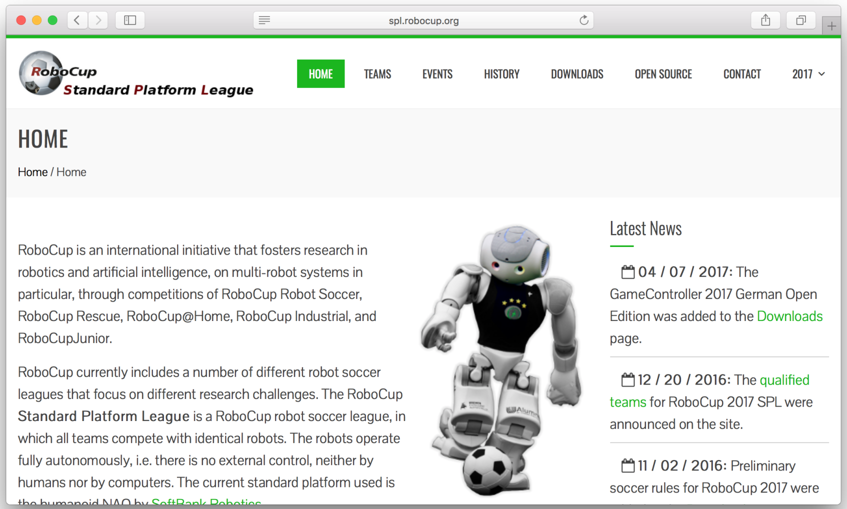
league by now. At RoboCup 2016, several teams preferred to see it on the official external screen of the GameController PC next to the previous GameStateVisualizer. The previous GameStateVisualizer displayed compact information about the current games to the audience in front of a background that follows the theme of the general event and is usually displayed on the external screen. Displaying both programs next to each other resulted in a cluttered, unprofessionally looking screen. Therefore, the rather simple functionality of the GameStateVisualizer was integrated into the TeamCommunicationMonitor as a separate mode. As a result, the audience can see both, the general game information as well as what the robots are "thinking" (cf. Fig. 5).

6.3 Website

Andre Mühlenbrock transferred the SPL website to spl.robocup.org (cf. Fig. 6).

7 Summary

Similar to most previous years, we incrementally improved our system by replacing certain components that did not seem to be competitive anymore for upcoming competitions. Some of our new features, in particular the walking engine, have already been used at the RoboCup German Open 2017 and contributed to our success in that competition. Thus, we are looking forward to a successful participation in the RoboCup 2017 in Nagoya!



RoboCup is an international initiative that fosters research in robotics and artificial intelligence, on multi-robot systems in particular, through competitions of RoboCup Robot Soccer, RoboCup Rescue, RoboCup@Home, RoboCup Industrial, and RoboCupJunior.

RoboCup currently includes a number of different robot soccer leagues that focus on different research challenges. The RoboCup Standard Platform League is a RoboCup robot soccer league, in which all teams compete with identical robots. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers. The current standard platform used is the humanoid NAO by SoftBank Robotics

04 / 07 / 2017: The GameController 2017 German Open Edition was added to the Downloads page.

12 / 20 / 2016: The qualified teams for RoboCup 2017 SPL were announced on the site.

11 / 02 / 2016: Preliminary soccer rules for RoboCup 2017 were

Fig. 6. The new website of the Standard Platform League

References

1. Hengst, B.: rUNSWift Walk2014 report. Tech. rep., School of Computer Science & Engineering University of New South Wales, Sydney 2052, Australia (2014), `http://cgi.cse.unsw.edu.au/~robocup/2014ChampionTeamPaperReports/20140930-Bernhard.Hengst-Walk2014Report.pdf`
2. Hofmann, M., Urbann, O., Schwarz, I., Rensen, F., Moos, A.: Playing robot soccer outdoor. In: Lofaro, D.M., BaekKyu Cho, K., Behnke, S., Lee, D.D., Lau, N. (eds.) The 11th Workshop on Humanoid Soccer Robots at 16th IEEE-RAS International Conference on Humanoid Robots (2016)
3. Röfer, T., Brose, J., Göhring, D., Jüngel, M., Laue, T., Risler, M.: GermanTeam 2007. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007: Robot Soccer World Cup XI Preproceedings. RoboCup Federation, Atlanta, GA, USA (2007)
4. Röfer, T., Laue, T., Richter-Klug, J.: B-Human 2016 – robust approaches for perception and state estimation under more natural conditions. In: Behnke, S., Lee, D.D., Sariel, S., Sheh, R. (eds.) RoboCup 2016: Robot Soccer World Cup XX. Lecture Notes in Artificial Intelligence (2017), to appear
5. Röfer, T.: CABSL – C-based agent behavior specification language (2017), `https://github.com/bhuman/CABSL`
6. Röfer, T., Laue, T., Kuball, J., Lübken, A., Maaß, F., Müller, J., Post, L., Richter-Klug, J., Schulz, P., Stolpmann, A., Stöwing, A., Thielke, F.: B-Human team report and code release 2016 (2016), only available online: `https://github.com/bhuman/BHumanCodeRelease/raw/master/CodeRelease2016.pdf`