B-Human

Team Description for RoboCup 2015

Thomas Röfer[1], Tim Laue[2], Jesse Richter-Klug[2], Jonas Stiensmeier[2], Maik Schünemann[2], Andreas Stolpmann[2], Alexander Stöwing[2], Felix Thielke[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 seven RoboCup German Open competitions and in six RoboCups. We always won the German Open and became world champion four times.

As in previous years, we released our code after the RoboCup 2014 together with a detailed description [4] on our website and on GitHub (`https://github.com/bhuman/BHumanCodeRelease`). Up to date, we know of 21 teams that based their RoboCup systems on one of our code releases (AUTMan Nao Team, Austrian Kangaroos, BURST, Camellia Dragons, Crude Scientists, Edinferno, JoiTech-SPL, NimbRo SPL, NTU Robot PAL, SPQR, UChile, Z-Knipsers) or used at least parts of it (Cerberus, MRL SPL, Nao Devils, Northern Bites, NUbots, RoboCanes, RoboEireann, TJArk, UT Austin Villa).

This team description paper is organized as follows: In the next section, we present all current team members, followed by the descriptions of our newest developments since the end of RoboCup 2014. As the new rules changed the color of the goals from yellow to white, i. e. from a rather special color to one that is very common in the environment of a RoboCup soccer field, the detection of goals had to be adapted (cf. Sect. 3.1) and the detection of new features on the field was added (cf. Sect. 3.2). We continued to automate the calibration of extrinsic camera parameters (cf. Sect. 3.3). Regarding the robots' motions, we improved the dynamic use of the arms (cf. Sect. 4.1) and we took measures to keep the joint temperatures low in potentially long play-off games (cf. Sect. 4.2). We also worked on our framework again (cf. Sect. 5.1), which is mainly interesting for teams that based their systems on our framework. In addition, we developed

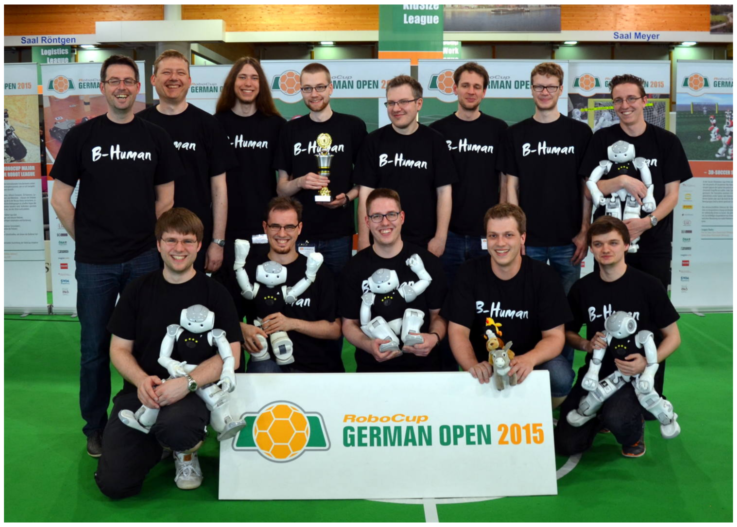


Fig. 1: The B-Human team after winning the RoboCup German Open 2015

a new tool for the league that monitors the communication between the robots on the field (cf. Sect. 5.2). Finally, this paper ends with a short summary (cf. Sect. 6).

2 Team Members

B-Human currently consists of the following people, most of whom are shown in Fig. 1 (left to right, top to bottom):

Team Leaders / Staff: Judith Müller[3], Tim Laue, Thomas Röfer.
Students: Felix Thielke, Jesse Richter-Klug, Jonas Stiensmeier, Arne Böckmann, Jan-Bernd Vosteen, Patrick Glaser, Florian Maaß, Alexis Tsogias, Alexander Stöwing, Andreas Stolpmann, Vitali Gutsch, Maik Schünemann[3].
Associated Researchers[3]: Udo Frese, Dennis Schüthe, Felix Wenk.

3 Vision

3.1 Detection of White Goals

The most important rule change from last year is the introduction of white goals. While yellow goals were relatively easy to detect based on color alone, white is much more common in the environment. For instance, at the RoboCup German Open 2015, white boards were standing around the field, including the background of the goals, and a white column was located next to the field. In

[3] Not shown in the photo.

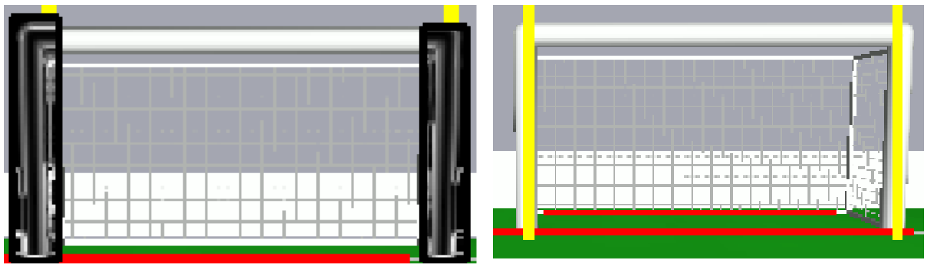


Fig. 2: On the left, the (simulated) image of a goal is shown, where the areas around the goalposts are convolved with the Sobel operator. The right (simulated) image shows detected goal with the identified left and right goalposts.

addition, the construction of the goal that holds the net can also be (and often is) white according to the rules. For the German Open 2015, we improved our goal detector for yellow goals, which uses a region based approach (cf. [3]). Last year's version was purely based on color segmentation, which could not succeed anymore, since the white goalposts could actually appear in front of other white surfaces in the camera image. Therefore, an edge detection was added to the goal detector as well as a lot of sanity checks. However, there still were quite a number of false positives, where goalposts were detected in other robots and the boards surrounding the field. In many cases, our self-localization module is able to filter wrong detections out, but this did cannot work in all cases.

Therefore, we are currently developing a new goal detector for the RoboCup 2015. The new approach searches for white regions directly below the field border, because the rules define what can be visible on the field (i. e. robots, goals, the ball, and referees wearing black or dark blue), but it is unknown what could be seen outside the field. If something white is found, it should either be a robot or a goalpost. The latter has an expected size that can be determined from the location of its foot point in the image and the goal size described in the rules. A sub image that covers the area of the expected goalpost is then convolved with the Sobel operator to create an edge map of the goalpost (cf. Fig. 2).

On this edge map, a Hough transform is used to detect lines. The accumulator array will only be filled with lines that have a certain angle in the image. The angle is calculated by estimating how a vertical line in reality would be seen in the image, which is not necessarily exactly vertical because of the camera perspective and the rolling shutter effect.

Afterwards, the detected lines are fitted onto a model of the goal to validate whether the region observed belongs to a goal. If two goalposts are found, the same procedure is then applied to the crossbar to validate the hypothesis.

3.2 Detection of Penalty Area and Penalty Mark

As the goal detection is not as reliable as it was before, our robots are now detecting other complex features on the field to compensate for that problem. These

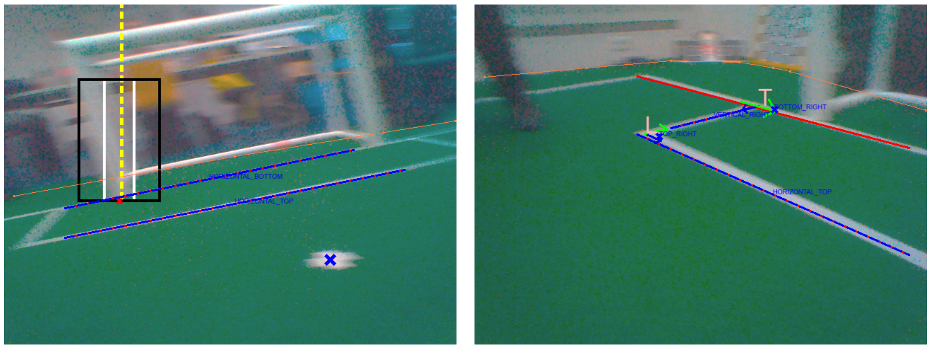


Fig. 3: Penalty mark detection and penalty area detection from lines (left) and from L and T intersections (right)

features are the penalty mark and the lines and intersections that constitute the penalty area. They are both valuable for the self-localization, because they only exist twice on the field, in contrast to most of the lines and intersections that our vision system has already been able to detect before.

On the left, Fig. 3 shows an image recorded by a robot during a practice match. The position of the penalty mark detected is marked by a blue X. The two parallel lines of the penalty area are labeled as a goal keeper would see them when standing in its goal.[4] The detection of the penalty mark builds upon early phases of the line detection in which points are identified that could potentially belong to a field line. Such points that were not joined with others to form a field line, which requires minimum distance, are candidates for the penalty mark. They are accepted as such if their surroundings in a distance bigger than the dimensions of a penalty mark contain (almost) no white.

The detection of the penalty area is achieved by analyzing the patterns of lines and intersections that were recognized. Two long parallel lines were recognized on the left of Fig. 3, which were in the expected distance from each other for the two main lines of the penalty area. Patterns of T and L intersections recognized are used to gather more information about the exact location of the penalty area. The orientation of such a pair of intersections that is connected by a line contains sufficient information to reconstruct the lines of the penalty area that are intersecting. On the right, Fig. 3 illustrates this situation.

3.3 Camera Calibration

The camera calibration usually consists of two steps: the intrinsic calibration to determine the focal length and the image center (we ignore distortions, because they are relatively small) and the extrinsic calibration to determine the camera's location in the robot. For the latter, we mainly focus on its relative

[4] Technically, the side from which the penalty area is observed cannot be determined when only detecting the two parallel lines, so the labeling is just a guess in this case.

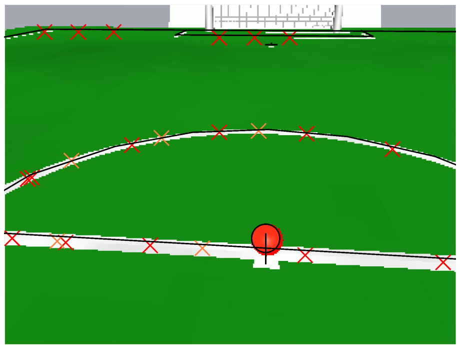


Fig. 4: A (simulated) image of the points on field lines selected. The red and orange crosses denote points selected for the upper and the lower camera respectively.

orientation to the ground plane. We perform an intrinsic calibration for each camera using the "GML C++ Camera Calibration Toolbox" [5] together with a compatible chessboard. The extrinsic camera calibration uses the soccer field as calibration pattern. Points on field lines are picked while the robot is looking around, matched to the closest lines according to a model of the field, and the parameters are then optimized using the Gauss-Newton method. The estimated parameters are both cameras' and the body's roll and pitch offsets, as well as the robot's position on the field to compensate for small deviations from the robot's pre-defined calibration position.

The points on field lines can be picked manually or automatically. The main improvements were performed in the automatic selection of points, which makes the automatic approach feasible now. The points on field lines are detected using B-Human's normal vision system [4]. However, to ensure a uniform density of points in all directions and distances, new points are only accepted if they are farther away than 30 cm from points that were already picked before, as shown in Fig. 4. In addition, they are only recorded when the head is not moving to avoid errors resulting from time differences between joint angle measurements and image taking and the rolling shutter effect. Although these effects are usually compensated for by B-Human's vision system, there is a certain amount of guessing involved, which should be avoided when calibrating the cameras.

Although the goal is to perform a fully automatic calibration process, it is always possible that erroneous points are detected, e. g. because it is impossible to get access to a completely empty field during the setup of an official competition, during which other objects, such as cables, are on the field and can be mistaken for field lines. In addition, some field lines further away might have too little contrast for our vision system to detect them. Therefore, the calibration module still offers the option to manually add and remove a few points.

Overall, the time it takes to calibrate the extrinsic camera parameters has significantly been improved.

4 Motion

4.1 Integration of Arm Motions

B-Human is one of the few teams that actively uses the robot's arms in soccer games, not only when getting up and when the goalkeeper dives. We use the arms as sensors to detect the contact with other robots. We also use them to hinder other robots from overtaking our robots, while, in contrast, our robots take them away when they are overtaking another robot. They also take an arm out of the way when they plan to steal the ball from an opponent by walking along in front of it or if the arm is potentially blocking the goalkeeper's view at the ball. As we use the arms more and more, they are now seen as a third kind of motion unit in addition to the body with its legs and the head. In fact, the two arms can be moved independently from each other, so they are actually two separate motion units.

The robot can operate in different motion modes, some of which need all motion units working together (e. g. getting up), while in others (e. g. walking), they can be controlled separately. And of course, there are transitions between these different motion modes that have to be executed smoothly to avoid unstabilizing the robot. In addition, although during walking the arms, the head, and the body can be controlled relatively independent from each other, the walking engine still needs to know which motions the other motion modules intend to perform to properly balance. The walking engine also prefers to control the arms during walking to let them swing slightly. Therefore, the walking engine is in control of the arms if they are not executing another motion.

As in the past, managing the different motion modes is split into three phases. In the first one, it is decided which motion modules are currently active and, in case of a transition between different motion modes, how strongly their output influences the actual joint angles that are set. In the second phase, all motion modules that are currently active compute their output joint angles. In a third phase, these joint angles are mixed together based on the weightings that were defined in the first phase.

4.2 Load Balancing

RoboCup games can take quite a while, in particular the play-offs, in which the clock is stopped when not in playing state. During such a game, the robots walk a lot, but they also stand a lot, e. g. when waiting for other robots to reach their kick-off positions in the ready state. This has two negative effects: On the one hand, there is a high power consumption, which can become critical during long games with weak rechargeable batteries. On the other hand, the joints are getting very hot[5], which results in weaker and less precise joint movements, and ultimately in the robots falling down more often.

[5] The joint temperature is an estimation done by NAO's operating system NAOqi and not an actual measurement.

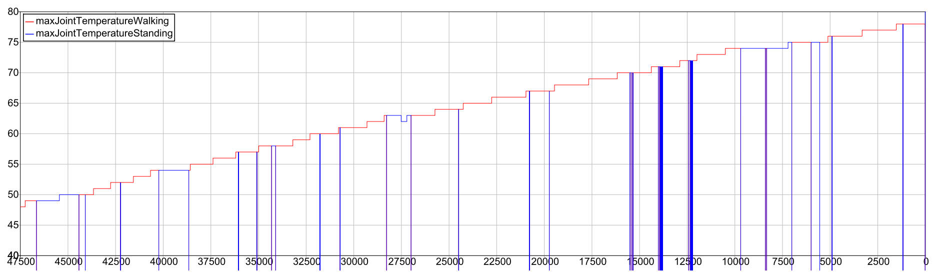


Fig. 5: The temperature in °C of the hottest joint of B-Human's robot "Kripke" in the second half of the final of the RoboCup German Open 2015. The robot also played the whole first half before. The time is shown as the number of vision frames until the end of the half, i. e. in $\frac{1}{60}$secs. The vertical blue lines mark the begin and end of phases in which the robot was standing.

We address this problem from two different angles. The team behavior is now more optimized to avoid that robots have to continuously walk around on the field. Instead, they try to stand still in more situations. But even when standing, the heat of the joints can quickly go up if the weight is not evenly distributed over both legs. Therefore, while standing, our robots are continuously detecting the cumulative electrical currents for each leg over a short amount of time and then lean the torso over towards the side the leg which had the smaller power consumption so far. Thereby, the robots are able to distribute the load more equally and to keep the temperature from increasing while the robot is standing.

As can be seen in Fig. 5, on a robot playing during the final of the RoboCup German Open 2015, the increase in joint temperature was always slowed down when the robot was standing.

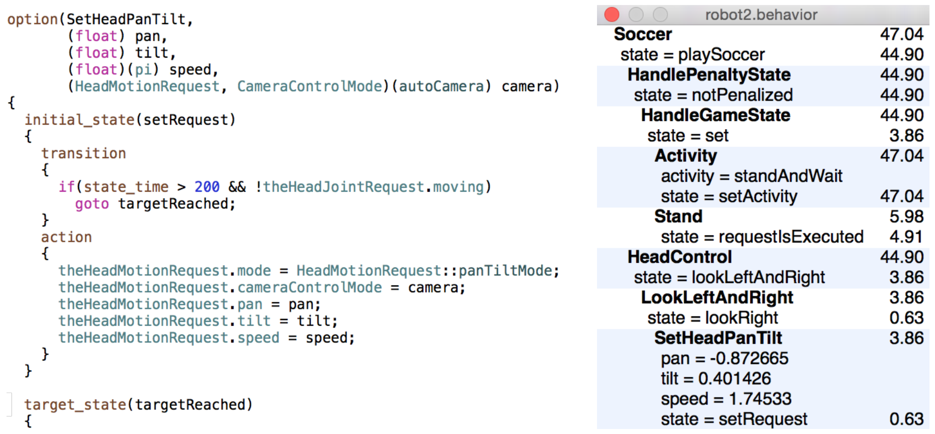
5 Infrastructure

5.1 B-Human Framework

We replaced our own mathematics library by *Eigen* [2] throughout the whole B-Human system. We kept some special methods that Eigen does not offer and integrated them through Eigen extensions. Since Eigen adds a considerable source code footprint to compilation, we use precompiled headers in our build system for all projects that include Eigen.

Our C-based *Agent Behavior Specification Language (CABSL)* [3] now uses the same syntax for specifying parameters as the streamable data types in our system. Thereby, the actual parameters of each option can be recorded in log files and they can also be shown in the behavior dialog (cf. Fig. 6). Please note that the source code shown in Fig. 6 is still understood by a C++ compiler and thereby by the editors of C++ IDEs.

To further enhance the usage of log files, we added the possibility for our modules to annotate individual frames of a log file with important information.



```
option(SetHeadPanTilt,
       (float) pan,
       (float) tilt,
       (float)(pi) speed,
       (HeadMotionRequest, CameraControlMode)(autoCamera) camera)
{
  initial_state(setRequest)
  {
    transition
    {
      if(state_time > 200 && !theHeadJointRequest.moving)
        goto targetReached;
    }
    action
    {
      theHeadMotionRequest.mode = HeadMotionRequest::panTiltMode;
      theHeadMotionRequest.cameraControlMode = camera;
      theHeadMotionRequest.pan = pan;
      theHeadMotionRequest.tilt = tilt;
      theHeadMotionRequest.speed = speed;
    }
  }

  target_state(targetReached)
  {
```

```
robot2.behavior
Soccer                              47.04
 state = playSoccer                 44.90
  HandlePenaltyState                44.90
   state = notPenalized             44.90
    HandleGameState                 44.90
     state = set                     3.86
      Activity                      47.04
       activity = standAndWait
       state = setActivity          47.04
      Stand                          5.98
       state = requestIsExecuted     4.91
  HeadControl                       44.90
   state = lookLeftAndRight          3.86
    LookLeftAndRight                 3.86
     state = lookRight               0.63
      SetHeadPanTilt                 3.86
       pan = -0.872665
       tilt = 0.401426
       speed = 1.74533
       state = setRequest            0.63
```

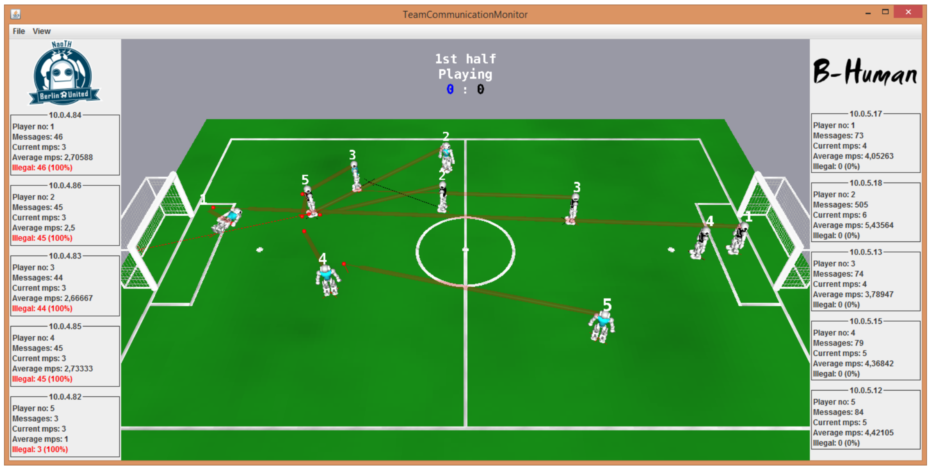
Fig. 6: The behavior view shows the actual parameters, e. g. of the option *SetHeadPanTilt* (left: source, right: view). The display of values equaling default parameters (specified in a second pair of parentheses) is suppressed. On the right of the view, the number of seconds an option or state is active is shown.

This is, for example, information about a change of game state, the execution of a kick, or other information that may help us to debug our code. Thereby, when replaying the log file, we may consult a list of those annotations to see whether specific events actually did happen during the game. In addition, if an annotation was recorded, we are able to directly jump to the corresponding frame of the log file to review the emergence and effects of the event without having to search through the whole log file.

Log files now also contain additional information that allows replaying them even after the specification of the data types recorded has changed (in most cases). Based on the specification of each data type stored in a log file and its current specification, the data is automatically translated during the replay if it has changed. Missing fields keep their default values. Removed fields are ignored.

5.2 Team Communication Monitor

With the start of the Drop-in Player Competition in 2014, also a standard communication protocol was introduced, i. e. the *SPLStandardMessage*, to allow robots of different teams to exchange information. While implementing the standard protocol correctly is a necessity for the Drop-in Player Competition to work, it is also an opportunity for teams to lower the amount of coding they have to do to develop debugging tools, because when all teams use the same communication protocol, the tools they wrote could also be shared more easily. Therefore, B-Human developed the *TeamCommunicationMonitor* (TCM) as a standard tool to monitor the network traffic during SPL games. This project is partially funded by the RoboCup Federation.



```
10.0.4.84
Player no: 1
Messages: 46
Current mps: 3
Average mps: 2,70588
Illegal: 46 (100%)

10.0.4.86
Player no: 2
Messages: 45
Current mps: 3
Average mps: 2,5
Illegal: 45 (100%)

10.0.4.83
Player no: 3
Messages: 44
Current mps: 3
Average mps: 2,66667
Illegal: 44 (100%)

10.0.4.85
Player no: 4
Messages: 45
Current mps: 3
Average mps: 2,73333
Illegal: 45 (100%)

10.0.4.82
Player no: 5
Messages: 3
Current mps: 3
Average mps: 1
Illegal: 3 (100%)
```

```
10.0.5.17
Player no: 1
Messages: 73
Current mps: 4
Average mps: 4,05263
Illegal: 0 (0%)

10.0.5.18
Player no: 2
Messages: 505
Current mps: 6
Average mps: 5,43564
Illegal: 0 (0%)

10.0.5.13
Player no: 3
Messages: 74
Current mps: 4
Average mps: 3,78947
Illegal: 0 (0%)

10.0.5.15
Player no: 4
Messages: 79
Current mps: 5
Average mps: 4,36842
Illegal: 0 (0%)

10.0.5.12
Player no: 5
Messages: 84
Current mps: 5
Average mps: 4,42105
Illegal: 0 (0%)
```

Fig. 7: The display of the TeamCommunicationMonitor during the semifinals of the RoboCup German Open 2015

The TCM visualizes all data currently sent by the robots, highlighting illegal messages. Furthermore, it uses the knowledge about the contents of the SPL-StandardMessages in order to visualize them in a 3-D view of the playing field. This makes it also suitable for teams to debug their network code. Visualized properties of robots are their position and orientation, their fallen and penalty states (the latter is received from the GameController), where they last saw the ball, and their player numbers. A screenshot of the TCM can be seen in Fig. 7.

In order to display all messages of robots sending packets, the TCM binds sockets to all UDP ports that may be used by a team as their team number, i. e. ports 10000 to 10099, and listens for any incoming packets. When a packet was received, it is parsed as an SPLStandardMessage, marking all fields as invalid that do not contain legal values according to the definition of the SPLStandard-Message. As the TCM only listens and does not send anything, it may run on multiple computers in the same network at once without any problems.

The TCM identifies robots using their IP address and assumes them to be sending on the port that matches their team number. The team number sent by the robots as part of the SPLStandardMessage is marked as invalid if it does not match the port on which the message was received. For each robot, the TCM holds an internal state containing the last received message as well as the timestamps of the most recently received messages in order to calculate the number of messages per second. A robot's state is discarded if no message was received from the robot for at least two seconds.

Displaying the 3-D view is done with OpenGL using the JOGL library[1]. The visualization subsystem is also extensible so teams can write plug-ins containing drawings that visualize data from the non-standardized part of their messages. We have written a plugin to display perceptions of goals, obstacles, and whistles as well as basic status information of the robots which our robots send via the non-standardized message part.

Besides just visualizing received messages, the TCM also stores all received messages both from robots and from the GameController in log files. These can later be replayed, allowing teams and organizers to check the communication of robots after the game is over.

The TCM is developed as part of the repository of the GameController (`https://github.com/bhuman/GameController`), which is the standard referee tool of the SPL and Humanoid leagues developed mostly by B-Human team members since 2012, and it shares parts of its code.

After having been successfully used at the RoboCup German Open 2015 to ensure valid messages from all teams during the Drop-in games, the TCM was publicly released in early June and will be installed on the referee PC on each SPL field at RoboCup 2015.

6 Summary

In this paper, we described a number of topics that we have implemented or we have been working on since RoboCup 2014. Most prominently, we adapted our existing goal detection to detecting white goals, which we successfully used at the RoboCup German Open 2015. However, we are currently working on a new approach, because there were still too many false positives. In addition, we added other feature detectors to be less dependent on the detection of goals, namely for penalty marks and penalty areas. We also improved existing parts of our system such as the procedure for camera calibration, the management of arm motions, dealing with joint heat, and the general infrastructure of our system.

Besides our annual code releases, B-Human maintains the GameController, which is now accompanied by the TeamCommunicationMonitor as another contribution of our team to the development of the Standard Platform League.

References

1. Gothel, S., Santina, R., Koutsolampros, P.: JOGL - Java binding for the OpenGL API. `http://jogamp.org/jogl/www/`
2. Guennebaud, G., Jacob, B., et al.: Eigen v3. `http://eigen.tuxfamily.org` (2010)
3. Röfer, T., Laue, T., Müller, J., Bartsch, M., Batram, M.J., Böckmann, A., Böschen, M., Kroker, M., Maaß, F., Münder, T., Steinbeck, M., Stolpmann, A., Taddiken, S., Tsogias, A., Wenk, F.: B-Human team report and code release 2013 (2013), only available online: `http://www.b-human.de/downloads/publications/2013/CodeRelease2013.pdf`
4. Röfer, T., Laue, T., Müller, J., Schüthe, D., Böckmann, A., Jenett, D., Koralewski, S., Maaß, F., Maier, E., Siemer, C., Tsogias, A., Vosteen, J.B.: B-Human team report and code release 2014 (2014), only available online: `http://www.b-human.de/downloads/publications/2014/CodeRelease2014.pdf`
5. Velizhev, A.: GML C++ camera calibration toolbox. `http://graphics.cs.msu.ru/en/node/909`