B-Human 2013: Ensuring Stable Game Performance*

Thomas Röfer[1], Tim Laue[1], Arne Böckmann[2], Judith Müller[2], and Alexis Tsogias[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

Abstract. The aim of a soccer game is to score more goals than the opponent does. The chances of doing so increase when all the own players are continuously present on the field. In the RoboCup Standard Platform League (SPL), the only reasons for players not to participate in a game are that they either have been penalized or they have not been ready for the match in the first place. This paper presents some of the methods the 2013 SPL world champion team B-Human uses to ensure a stable and continuous game performance. These include overcoming the weak obstacle avoidance of our 2012 system, reacting to a rule change by an improved detection of the field boundary, better support for analyzing games by logging images in real time, and quality management through procedures the human team members follow when participating in a competition.

1 Introduction

B-Human participates in the RoboCup Standard Platform League. It is a joint team of the Universität Bremen and the German Research Center for Artificial Intelligence (DFKI) and consists of numerous undergraduate students as well as three researchers. Since 2009, we won every world championship in the SPL except for 2012, where we became the runner-up. The main goal for 2013 was to win back the championship title. To do that, we had to analyze why we failed in 2012. Some of the reasons were external: the vibrating field construction and the bad wireless network introduced a high degree of noise into the games. They simply all looked bad, which made it hard to detect problems that were still present in our system. Thanks to the setup in Eindhoven, these problems were not present in 2013, at least not after UDP packet buffering was switched off in the wireless access points.

One of the major weaknesses we had in 2012 was the detection of other robots or obstacles in general [1] , which did not become apparent before the semifinal.

* The authors would like to thank all B-Human team members for providing the software base for this work.

S. Behnke et al. (Eds.): RoboCup 2013, LNAI 8371, pp. 80–91, 2014.

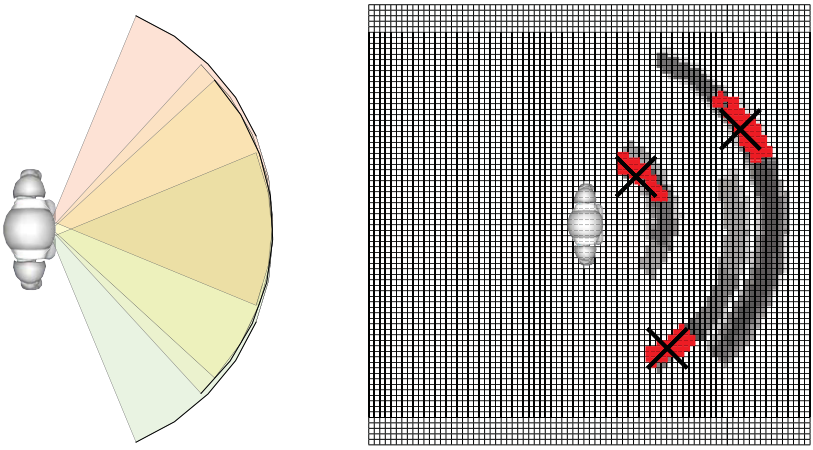
The visual robot perception relied on the detection of the colored waistbands. Since they were rather small, it was limited to short ranges. In addition, our opponents in the semifinal and final played with arms behind the robot's back, which made the bands hard to detect when approaching such a robot from behind. The sonar-based obstacle detection struggled with the new ("silver") ultrasound sensors of the NAO V4 robots that we had received one week prior to the competition. It did not work well with the previously used method for acquiring the sonar measurements. As a result of both weaknesses, our robots committed an unusually high number of "player pushing" offenses. They sometimes also did not switch to their tackling behavior, which resulted in a number of "ball holding" penalties. As a result, our robots were often taken off the field in the last two games. Sometimes, only the goalkeeper was left. This was a huge advantage for our opponents.

For 2013, one of the main goals was to keep our robots on the field, i. e. to avoid receiving penalties. This goal was achieved while still scoring by far the most goals of all teams. According to the log files of the GameController [2], our team only had an average of 3.75 penalties per game. This was the lowest value in the whole competition. The average value of all teams was 11.06.

This paper focuses on a few examples of approaches, how this was achieved. To avoid "player pushing" and "ball holding" penalties, the detection of obstacles was significantly improved, using both ultrasound and vision. This is described in Sect. 2. In response to a rule change that allowed goals on neighboring fields to be visible, an improved detection of the field boundary was implemented, which is presented in Sect. 3. In the actual competition, it was more useful to avoid "leaving the field" penalties, because no neighboring goals were visible, but orange logos were placed next to the field that could have been mistaken for the ball in some situations. If a robot still misbehaves, it is often hard to determine what the actual reason for an error was. A new logging technique that includes both camera images of the NAO and runs in real time, simplifies debugging a lot because it gives the programmer the context needed to understand what happened. This method is described in Sect. 4. Some penalties such as "fallen robot" and "request for pickup" can be avoided by careful selection and preparation of the robots that play and by a general quality management. The organizational procedures that we use to ensure the best performance of our robots are outlined in Sect. 5.

2 Obstacle Detection

Obstacle detection is required for navigation on the field, but also for tackling. The NAO can perceive obstacles in various ways: visually [3,4,5], through ultrasound [6], with its foot bumpers, and with its arms [1]. We use all of these methods. However, the ultrasound detection method employed in 2012 did not work very well [1] and the previous approach of detecting robots visually relied on the colored waistbands, which only worked at short range and could not be applied anymore in 2013, because the waistbands were replaced by jerseys. For



(a) (b)

Fig. 1. Ultrasound obstacle detection. a) Overlapping measuring cones. b) Obstacle grid that shows that sometimes even three robots can be distinguished. Clustered cells are shown in red, resulting obstacle positions are depicted as crosses.

2013, the ultrasound-based obstacle detection was reimplemented and the previous visual robot detection was replaced by a visual obstacle detection. Since the new jerseys were not available before the competition, none of our methods is able to determine the team color of the objects perceived; they are simply treated as anonymous obstacles.

2.1 Ultrasound-Based

The NAO is equipped with two ultrasound transmitters (one on each side) and two receivers (again one on both sides), which results in four possible combinations of transmitter and receiver used for a measurement. We model the areas covered by these different measurement types as cones with an opening angle of 90° and an origin at the position between the transmitter and the receiver used (cf. Fig. 1a). Since NAO's torso is upright in all situations in which we rely on ultrasound measurements, the whole modeling process is done in 2-D. We also limit the distances measured to 1.2 m, because experiments indicated that larger measurements sometimes result from the floor. The environment of the robot is modeled as an approximately 2.7 m × 2.7 m grid of 80 × 80 cells. The robot is always centered in the grid, i. e. the contents of the grid are shifted when the robot moves. Instead of rotating the grid, the rotation of the robot relative to the grid is maintained. The measuring cones are not only very wide, they also largely overlap. To exploit the information that, e. g., one sensor sees a certain obstacle, but another sensor with a partially overlapping measuring area does not, the cells in the grid are ring buffers. These ring buffers store the last 16 measurements that concerned each particular cell, i. e. whether a cell was

measured as free or as occupied. With this approach, the state of a cell always reflects recent measurements. Experiments have shown that the best results are achieved when cells are considered as part of an obstacle if at least 10 of the last 16 measurements have measured them as occupied. All cells above the threshold are clustered. For each cluster, an estimated obstacle position is calculated as the average position of all cells weighted by the number of times each cell was measured as occupied.

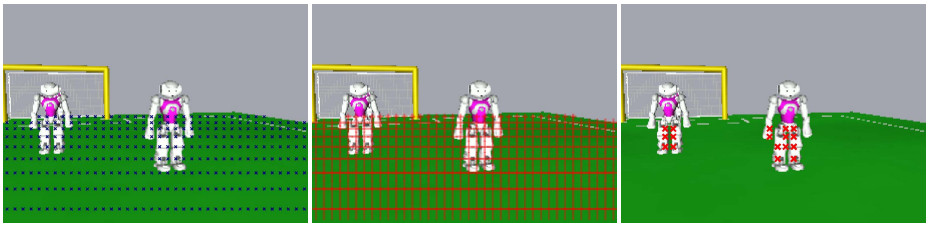
Since the ultrasound sensors have a minimum distance that they can measure, we distinguish between two kinds of measurements: close measurements and normal measurements. Close measurements are entered into the grid by strengthening all obstacles in the measuring cone that are closer than the measured distance, i. e. cells that are already above the obstacle threshold are considered to have been measured again as occupied. For normal measurements, the area up to the measured distance is entered into the grid as free. For both kinds of measurements, an arc with a thickness of 100 mm is marked as occupied in the distance measured. If the sensor received additional echoes, the area up to the next measurement is again assumed to be free. This sometimes allows narrowing down the position of an obstacle on one side, even if a second, closer obstacle is detected on the other side as well – or even on the same side (cf. Fig. 1b). Since the regular sensor updates only change cells that are currently in the measuring area of a sensor, an empty measurement is added to all cells of the grid every two seconds. Thereby, the robot forgets old obstacles. However, they stay long enough in the grid to allow the robot to surround them even when it cannot measure them anymore, because the sensors point away from them.

2.2 Vision-Based

The visual obstacle detection is based on the idea that obstacles are big non-green areas in the image. It is split into an initial detection stage and a refinement stage. The initial detection generates a list of spots in image coordinates that might be anywhere inside a possible obstacle. Afterwards, the refinement step filters these spots, using several sanity checks. In addition, it stitches obstacles together that span both camera images.

Initial Detection. The initial detection is based on the assumption that each non-green area that is bigger than a field line has to be an obstacle. The whole image is searched for such regions. To be able to search the image in a reasonable time, not every pixel is checked. Instead, a grid is utilized (cf. Fig. 2a). The vertical distance between two grid points is variable. It is calculated by projecting points to the image plane that are precisely 100 mm apart in the field plane. The horizontal distance between two grid points is fixed. To further reduce the runtime, pixels above the field boundary (cf. Sect. 3) are not checked.

For each grid point it is determined whether the point lies inside a non-green region or not. This is achieved by checking whether more than $k\%$ (currently $k = 93$) of the neighborhood around each point is not green. The neighborhood



(a) (b) (c)

Fig. 2. For each blue spot in (a), it is checked whether it contains an obstacle or not. The highlighted pixels in (b) show the neighborhoods around each grid point. (c) shows the detected possible obstacle spots.

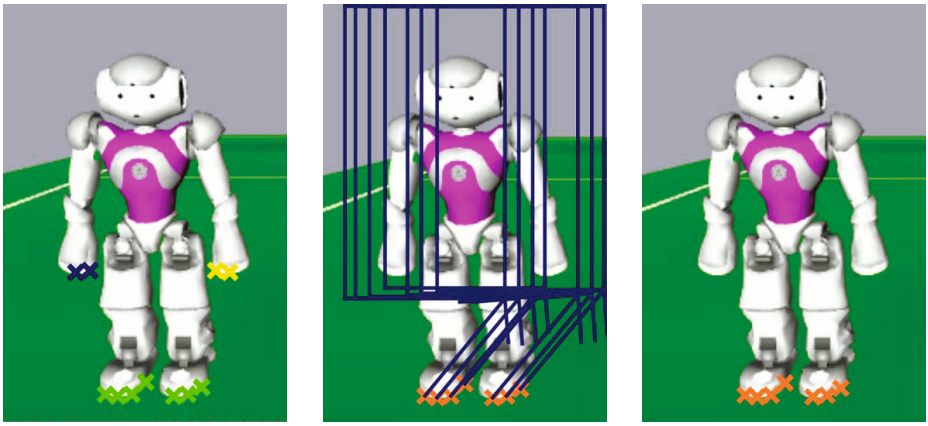
is a cross-shaped region that consists of the $2n$ neighboring pixels in vertical and horizontal direction (cf. Fig. 2b). n is the field line width at the given grid point, i. e. half the vertical distance between the grid point and its neighbor. The cross shape has been chosen due to its relative rotation invariance and ease of implementation.

Since the horizontal distance between two grid points is fixed, but the width of a neighborhood depends on the distance of the grid point, the neighborhoods of two adjacent grid points overlap. To avoid checking pixels for their greenness multiple times in the overlapping areas, a one-dimensional sum image is calculated for each grid row beforehand. The greenness of a certain region can then be calculated by subtracting the corresponding start and end pixels in the sum image. All grid points that lie inside non-green regions are considered to be possible obstacles.

Filtering. The possible obstacle spots show the general location of obstacle regions in the image. However, they usually do not show the foot point of an obstacle and neither do they show its size. In addition, they may contain false positives, in particular at field line junctions. Therefore, additional filtering is required. The filtering process consists of several steps:

Height check. The first step of the filtering process is to determine the height and the foot point of the obstacle at each possible spot. This is done by scanning the image vertically in both directions starting at each possible obstacle spot until either the horizon, i. e. the plane parallel to the field on the height of the camera, or a continuous patch of green is found. This way, the height can be determined well as long as the camera is more or less parallel to the ground. Since the NAO does not need to tilt its head and the body is kept upright while walking, this is usually the case.

If the height is less than a robot's width at the foot point, the obstacle spot will be discarded. This effectively removes most of the false positives. However, due to the fact that the environment around a RoboCup field is usually not green,



(a) (b) (c)

Fig. 3. Removal of false obstacle spots at the hands of other robots. The colored crosses in (a) are different obstacle spot clusters. The blue and the yellow clusters are false positives and should be removed. (b) shows the calculated ban regions and (c) shows the result.

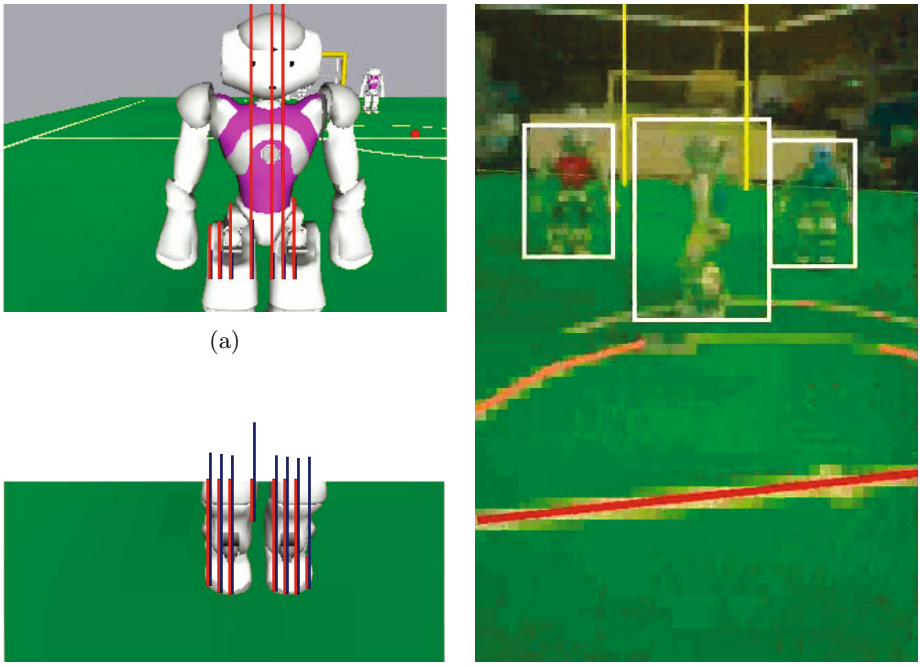
false positives close to the field border cannot be removed using only a height criterion. Still, since robots are not allowed to leave the field, false positives at the field border do not matter.

Hand removal. Depending on their posture, the hands of other robots may be detected as obstacle spots. Technically, those spots are not false positives since they are part of another robot. However, the foot point of such obstacles is not on the field plane. Thus it is impossible to calculate the distance to the obstacle. These obstacles are removed using the assumption that the lowest (in image coordinates) obstacle spot belongs to the foot of a robot. Starting from the lowest spot, an area that should contain the robot's hands is calculated for each spot. All spots inside that area are removed. Figure 3 illustrates this process.

Obstacle Stitching. When obstacles are close, they span both the lower and upper camera image. The foot point of such obstacles is in the lower image; however, the lower image does usually not show enough of the obstacle to pass the height check. In this case, the visible height is calculated and the intersection of obstacle and image border is converted to relative field coordinates and buffered.

When the next image from the upper camera is processed, the buffered points are moved (still in relative field coordinates) according to the odometry and projected back into the upper camera image. Here, the height check that was started in the lower image is continued (cf. Fig. 4a, b).

Clustering. Finally, the remaining obstacle spots are clustered and the center of mass is calculated for each cluster.



(a) (b) (c)

Fig. 4. (a, b) Obstacle stitching. Red lines show the measured height while blue lines indicate the minimal expected height. The image from the lower camera (b) does not contain enough of the obstacle to pass the height check; therefore, the height check is continued in the upper camera image (a). (c) Images of both cameras taken with the real time logger. The image quality is greatly reduced, but everything is still visible. The images are overlaid with perceptions that were also logged. See a video of the log file at `http://www.youtube.com/watch?v=rv3KpBRSTVw`.

3 Detecting the Field Boundary

The 2013 rules state that if fields are further away from each other than 3 m, a barrier between them can be omitted. This means that robots can see goals on other fields that look exactly like the goals on their own field. In addition, these goals can even be closer than a goal on their field. Therefore, it is very important to know where the own field ends and to ignore everything that is visible outside. Our old approach to determine the field boundary was to scan the image downwards to find the first green of sufficient size. This could now be easily on another field. Thus, a new approach was required. It searches vertical scanlines, starting from the bottom of the image going upwards. Simply using the first non-green pixel for the boundary is not an option, since pixels on field lines and all other objects on the field would comply with such a criterion. In addition, separating two or more fields would not be possible and noise could lead to false positives.

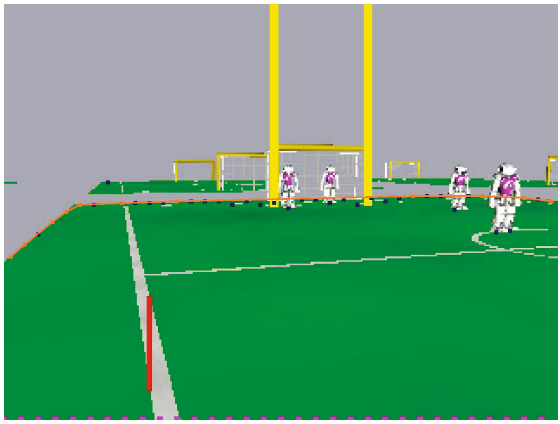


Fig. 5. Boundary spots (blue) and the estimated field boundary (orange) in a simulated environment with multiple fields

Our approach builds a score for each scanline while the pixels are scanned. For each green pixel a reward is added to the score. For each non-green pixel a penalty is subtracted from it. The pixel where the score is the highest is then selected as boundary spot for the corresponding scanline. The rewards and penalties are modified depending on the distance of the pixel from the robot when projected to the field. Field lines tend to have more green pixels above them in the image. As a result, they are easily skipped, but the gaps between the fields tend to be small in comparison to the next field when they are seen from a greater distance. So for non-green pixels with a distance greater than 3.5 meters, a higher penalty is used. This also helps with noise at the border of the own field. Figure 5 shows that most of the spots are placed on the actual boundary using this method.

Since other robots on the image also produce false positives below the actual boundary, the general idea is to calculate the upper convex hull from all spots. However, such a convex hull is prone to outliers in upward direction. Therefore, an approach similar to the one described by Thomas Reinhardt [7, section 3.5.4] is used. Several hulls are calculated successively from the spots by removing the point of the hull with the highest vertical distances to its direct neighbors. These hulls are then evaluated by comparing them to the spots. The best hull is the one with the most spots in near proximity. This one is selected as the field boundary. An example can be seen in Fig. 5.

4 Real-Time Logging with Images

Logging in real time during matches is, in general, nothing new to B-Human. However, until 2013, it was impossible to log images during actual games. Without images, log files are only of limited use for debugging, because the frame of reference for the programmer is missing. He or she can never be 100% certain about the situation or even whether looking at the correct situation at all.

The debugging of any perception-related mistake is close to impossible without images.

4.1 Logging Architecture

The main bottleneck when logging is the disk, i. e. a micro SD card in case of the NAO. Accessing the disk takes a very long time compared to everything else. The old logging implementation simply avoided this bottleneck by never writing to the disk during a game. Everything was buffered in memory until the game was over. However, when logging images, this approach is no longer feasible, because the robots would run out of memory very quickly. Therefore, we introduced a new non-real time process that continuously takes the oldest frames from the buffer and writes them to the disk. Due to the non-real time priority, data is only written when there is some spare time left. This way, the writing does not influence the real time property of the more important processes. In combination with a large buffer (600MB) and heavy image downscaling (cf. Sect. 4.2), this approach allows real time logging of images.

In addition, to further reduce the disk accesses, the data is compressed using Google's library Snappy [8] before it is written to the disk. Compressing the data and writing it to disk afterwards takes less time than to directly store the uncompressed data.

4.2 Image Compression

Compressing the images is done in two steps in order to get images that are as small as possible in minimal time.

At first, the images are scaled down by averaging blocks of 8×8 pixels. Since this is a very time consuming process, SSE instructions are used. In addition, it is very important not to process block after block since this impedes caching, resulting in great performance losses. Instead, we process line after line of the image buffering the results for eight lines until the average can be computed.

The second step reduces the number of representable colors and removes some unused space from the pixels. The cameras produce images in the YCbCr 4:2:2 format. This means that there are two luma channels for two chroma channels (the data has the form $\{Y_1, Cb, Y_2, Cr\}$). We only use one of the luma channels stored in each pixel since it would cost additional processing time to recreate the full resolution image. Thus one byte can be removed per pixel for the compressed image. Another byte can be gained by reducing the amount of bits each channel is allowed to use. In our case we use 6 bits for the luma and 5 bits for each chroma channel. Together, this results in another reduction of space by the factor of 2. This step might seem costly but it only takes 0.1 ms since it is done on a downscaled image.

The resulting compressed images are 128 times smaller than the original ones. Although a lot of detail is lost in the images, it is still possible to see and assess the situations the robot was in (see Fig. 4c).

5 Team Organization

The strict and careful organization of the tasks of the human team members is of significant importance for every match. This is definitely not a scientific issue but an inevitable necessity if one wants to make sure that all sophisticated algorithms and tactics are actually carried out effectively by the robots. There exist a number of mistakes that can be made and that ruin the performance of single robots or even the whole team and thereby render all scientific research useless. Common examples are using the wrong color calibration (making all robots blind or perceive false positives), mis-configured wireless settings (disabling communication among teammates), playing with buggy, untested code (perhaps causing crashes or deadlocks), and playing with damaged or insufficiently calibrated robots (letting the robot fall very often). In a perfect game, all five robots are well-configured, not damaged, and demonstrate their capabilities on the field for the whole time. Each robot that is not on the pitch – whether it is damaged, has not been prepared in time, or being penalized due to a rule violation such as Pushing (cf. Sect. 2) – weakens the overall team performance.

Therefore, our team has established a number of workflows that contribute to the goal of playing in this desired way. These procedures are the same in *all* matches, independent of the opponent team (no difference between The Invisibles and HTWK Leipzig) or the importance of the match (no difference between first round robin game and final). We consider this as very important as it trains all teammates and keeps concentration high.

5.1 Before a Game

To robustly walk at high speed and to perceive objects with high precision, continuously checking and calibrating all our robots is necessary. During a competition, multiple persons each spend many hours a day to make sure that we always have enough robots that can reliably play soccer. For this purpose, individual configurations are maintained, containing the maximum possible walk speeds and gait heights.

On the software side, we almost never start to play an official match in which we use software that has never played a match before. This could have been a test match, whether against another team or against an empty field, or the same code as in the official match before. This is necessary to avoid any negatively surprising and unintended side-effects of code changes.

Almost exactly 30 minutes before a match starts, we always perform a *test kickoff* of the whole team. All robots walk to their kickoff positions, the test game is started, and the robots play until they score a goal. By performing this test, we ensure that software and configuration files have been deployed correctly and all robots are fully operational. In case of any problem, there is still enough time to solve it without the need to hurry. If everything works fine, all robots are turned off and remain at the field.

All robots are again turned on several minutes before the start of the game (or the half). Until the actual beginning, they remain connected to a computer that

runs a monitoring tool. This enables us to react on unlikely but not impossible last minute problems such as a non-booting robot.

5.2 During a Game

During a game, the possible actions of the human team members are very limited: robots that have been picked-up can be fixed or replaced, a time-out or the half-time break can be used to change the configuration of the whole team.

In any way, these actions have to be carried out carefully and based on sound facts. As we are a huge team and many tasks are carried out by different persons, chaos needs to be avoided (situations such as somebody spontaneously shouting *"Time-Out!"* or disagreement about picking-up a certain robot). Therefore, we always have the same dedicated person that talks to the referee and one additional dedicated person that finally decides about all actions.

If the opponent team allows us to monitor our robots' communication, we always do. Our network packages include the so-called *robot health* that contains information about things such as battery power, joint temperatures, CPU temperature, or the frame rates of our two main processes. This allows us to identify potential problems before the robot starts to underperform and to discuss and schedule a replacement. During every game, a protocol is hand-written, including important facts for a later discussion but also statistics about which robot has fallen how many times. This allows us to watch out for potential hardware problems during the game and perhaps request a pick-up.

For every game, at least one already booted replacement robot is available at the field. This robot is always connected to a computer so that setting the player number and starting the B-Human software can be done immediately, making the robot ready to play in less than 30 seconds.

5.3 After a Game

After each game – test match or real one – the whole team discusses the match that was just played. The discussion is guided by the protocol that was written during the game. Tasks are assigned to individual team members. For analyzing certain mistakes, the log files (cf. Sect. 4) of the robots that misbehaved are watched to better understand the reason of an error.

6 Conclusions

At RoboCup 2013, B-Human's robots scored an average of 0.48 goals per minute of actual game time [2]. One reason for this success was that our robots were on the field most of the time. For instance, the new obstacle detection methods resulted in measurable improvements: While we do not know the number of "player pushing" penalties we received in 2012 (it certainly was high), we only received eight this year, in eight games. All other quarter-finalists committed more pushing offenses. Overall, B-Human was the team that received the least

penalties per game, which surely helped our success. In our opinion, our workflows significantly contributed to the comparatively small number of inactive robots, fallen robots, and requests for pick-up. As a consequence of our team's reliability, we have been able to play most of the tournament with a full set of well-working robots that showed a convincing performance.

During fall 2013, a large part of the software that we used during RoboCup 2013 will be released along with a detailed description of all components, including those described in Sections 2–4.

References

1. Röfer, T., Laue, T., Müller, J., Bartsch, M., Batram, M.J., Böckmann, A., Lehmann, N., Maaß, F., Münder, T., Steinbeck, M., Stolpmann, A., Taddiken, S., Wieschendorf, R., Zitzmann, D.: B-Human team report and code release 2012 (2012), Only available online:
   http://www.b-human.de/wp-content/uploads/2012/11/CodeRelease2012.pdf
2. Röfer, T., Bartsch, M.: Statistics of GameController logs (2013),
   http://www.tzi.de/spl/pub/Website/Results2013/RoboCup2013Statistics.pdf
3. Engel, M.: Anwendung von maschinellem Lernen zur echtzeitfähigen und kalibrierungsfreien Erkennung von humanoiden Fußballrobotern. Bachlor's thesis, HTWK Leipzig (2012)
4. Metzler, S., Nieuwenhuisen, M., Behnke, S.: Learning visual obstacle detection using color histogram features. In: Röfer, T., Mayer, N.M., Savage, J., Saranlı, U. (eds.) RoboCup 2011. LNCS, vol. 7416, pp. 149–161. Springer, Heidelberg (2012)
5. 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), Only available online:
   http://www.b-human.de/downloads/bhuman11_coderelease.pdf
6. Aquino, A., Akatsuka, C., Mushonga, T., Zhang, Y., He, Y., Lee, D.D.: The University of Pennsylvania RoboCup 2011 SPL Nao soccer team. In: Röfer, T., Meyer, N.M., Savage, J., Saranli, U. (eds.) RoboCup 2011: Robot Soccer World Cup XV Preproceedings, RoboCup Federation (2011)
7. Reinhardt, T.: Kalibrierungsfreie Bildverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfußball. Master's thesis, HTWK Leipzig (2011)
8. Google: Snappy – a fast compressor/decompressor (2013),
   http://code.google.com/p/snappy/