B-Human 2019 – Complex Team Play Under Natural Lighting Conditions

Thomas Röfer[1,2(✉)], Tim Laue[2], Gerrit Felsch[2], Arne Hasselbring[2], Tim Haß[2], Jan Oppermann[2], Philip Reichenberg[2], and Nicole Schrader[2]

[1] Deutsches Forschungszentrum für Künstliche Intelligenz, Cyber-Physical Systems, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
thomas.roefer@dfki.de

[2] Universität Bremen, Fachbereich 3 – Mathematik und Informatik, Postfach 330 440, 28334 Bremen, Germany
{tlaue,s_uhei4h,arha,hasst,jan_opp,s_ksfo6n,nicole2}@uni-bremen.de

Abstract. In the RoboCup Standard Platform League 2019, the team B-Human won the main competition and, together with Berlin United - Nao-Team Humboldt, the Mixed Team Competition. For being successful in such a competitive environment, many sophisticated solutions for a variety of robotics tasks need to be found and integrated in a reliable and efficient manner. In this paper, we focus on three aspects that we consider as crucial for this year's success and that have not been published before: a system of neural networks for ball classification and position estimation, a new framework for behavior specification along with its application to passes and set plays, and a set of approaches for maintaining the reliability of our robot team throughout a game.

1 Introduction

B-Human is a joint RoboCup team of the University of Bremen and the German Research Center for Artificial Intelligence (DFKI) that continuously participates in the Standard Platform League since 2009. We participated in ten RoboCup German Open competitions, the RoboCup European Open 2016, and eleven RoboCups and only lost six official games. As a result, we won all German Open competitions except for one as well as the European Open competition, and the RoboCups 2009, 2010, 2011, 2013, 2016 and 2017. This year, we won both the main competition and, together with the team Berlin United – Nao Team Humboldt as team *B&B*, the Mixed Team Competition. We also reached second place in the technical challenges, i.e. the open research challenge and the directional whistle challenge. In this paper, we present three major aspects that we consider to have significantly contributed to this year's success: the ball detection, a new behavior structure, and the overall system reliability.

For some years, starting with the *Outdoor Competition* at RoboCup 2016, the Standard Platform Leagues aims for natural lighting conditions on the playing fields. This means possible changes of the ambient brightness during a game

S. Chalup et al. (Eds.): RoboCup 2019, LNAI 11531, pp. 646–657, 2019.
https://doi.org/10.1007/978-3-030-35699-6_52

Fig. 1. Playing under natural lighting conditions during a practice match in Bremen (left) and at one of the *outdoor fields* at RoboCup 2019 (right)

as well as the possibility of bright spots and dark shadows, caused by structures in the environment or by the robots themselves, as depicted in Fig. 1. As an increasing number of games under such conditions has been foreseeable (at RoboCup 2019, B-Human actually played most games close to huge windows, e.g. on the field shown in the right part of Fig. 1), we decided to increase our development efforts towards more flexible vision approaches and to test more under such difficult conditions. One currently very popular and successful approach for object detection in such environments is the application of *Convolutional Neural Networks* (CNN). At the RoboCup Symposium 2019, we presented our CNN-based robot detection [1], which has been used throughout the whole competition. In this paper's Sect. 2, our new CNN-based approach for ball classification is presented. Both approaches make use of our new library for fast neural network inference on NAO robots. Its implementation has been released as open-source [7] and was also presented at the RoboCup Symposium [8].

In 2018 and 2019, multiple Standard Platform League rule changes directly affected the implementation of the robots' behavior by introducing new set plays such as corner kicks and kick-ins. Their proper handling – when being in defense as well as when being in offense – together with reasonable transitions to the normal game behavior significantly increases the complexity of the overall behavior. As the previously used C-based Agent Behavior Specification Language (CABSL) [6] together with libraries turned out to be too inflexible to be used as the only means of specifying behaviors, a new behavior framework has been developed. It is presented in Sect. 3, along with some of the innovations we implemented this year.

One important aspect of robot football, which is perhaps underrated as it is not a clearly defined field of research, is the overall team reliability. Whenever a single robot breaks or becomes removed for a rule violation, the opponent team immediately gains a numerical advantage for some time. In Sect. 4, we describe multiple measures that we took to decrease the likelihood of such situations. Furthermore, we present and discuss the corresponding game statistics recorded at RoboCup 2019.

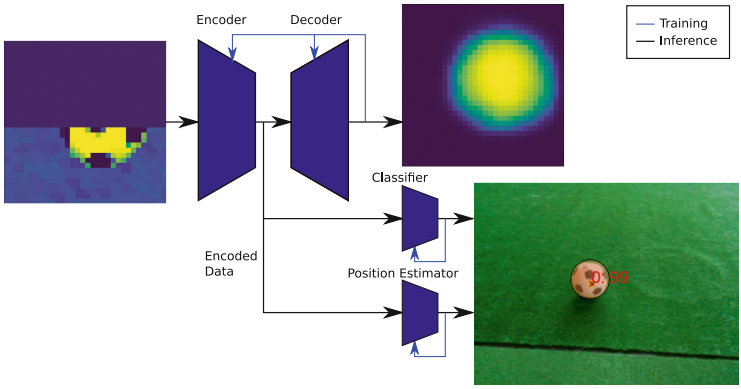

Fig. 2. System of neural networks for ball classification and position estimation. The encoder is a CNN, the classifier and the position estimator are DNNs. The architectures of the networks that are executed on the robot are shown in Table 1.

2 Ball Detection

Detecting a ball under natural lighting conditions means detecting it in very different conditions, be it in bright light, in completely shadow, or only partly covered by it. CNNs are currently a very popular approach for such problems. We have been using them for ball classification on extracted 32×32 patches since 2018 [4]. Instead of collecting more data and tuning hyperparameters, we aim to improve this approach by integrating additional knowledge.

2.1 Encoding Relevant Knowledge

This year, we used CNNs to estimate the exact *position* of the ball in a given patch. For this task, the labels of the existing dataset were extended by the position of the ball in the patch and the ball radius. If it is assumed a ball is always round, it is possible to create ball segmentation images from those labels. An example of such a segmentation image pair is shown in Fig. 3. A segmented image contains much more knowledge than a simple classification label. Some of this knowledge is relevant for ball classification and determining the ball position. If an encoder-decoder architecture is successfully trained to predict this segmentation from the original ball patches, this knowledge is also contained somewhere in the encoded features. Ideally, the features would comprise whether a ball is present in the image, which specific pixels carry the information that a ball is present, that a ball is round and connected even if a part is obscured, and the position and size of the ball.

Features containing this information could both be used for ball classification and position estimation. In the example shown in Fig. 3, this knowledge was incorporated by the autoencoder. Despite the input showing only one half of a ball, the segmented image shows a whole circle, where the ball would be.

Table 1. Architectures of the three neural networks for ball detection

(a) Encoder

Layer Type	Output Size
Input	32x32x1
Convolutional	32x32x8
Batch Normalization	32x32x8
ReLU Activation	32x32x8
Max Pooling	16x16x8
Convolutional	16x16x16
Batch Normalization	16x16x16
ReLU Activation	16x16x16
Max Pooling	8x8x16
Convolutional	8x8x16
Batch Normalization	8x8x16
ReLU Activation	8x8x16
Max Pooling	4x4x16
Convolutional	4x4x32
Batch Normalization	4x4x32
ReLU Activation	4x4x32
Max Pooling	2x2x32

(b) Ball Classifier

Layer Type	Output Size
Input	2x2x32
Flatten	128
Dense + Batch Norm + ReLU	32
Dense + Batch Norm + ReLU	64
Dense + Batch Norm + ReLU	16
Dense + Batch Norm + Sigmoid	1

(c) Position Estimator

Layer Type	Output Size
Input	2x2x32
Flatten	128
Dense + Batch Norm + ReLU	32
Dense + Batch Norm + ReLU	64
Dense + Batch Norm + ReLU	3

This hints on the ability to detect partly obscured balls, which is very important under natural lighting conditions.

Figure 2 shows the resulting system for ball classification and position estimation. After feature extraction by the encoder, ball classification and position estimation are done by two separate deep neural networks (DNN). The position is only estimated if the image was classified as containing a ball.

2.2 Results

For evaluation purposes, the performance of the new network was compared to the performance of the one we used during the German Open 2019. This was done by running both ball detectors on data recorded during a test game in front of a large window front and during the RoboCup 2019 final in Sydney.

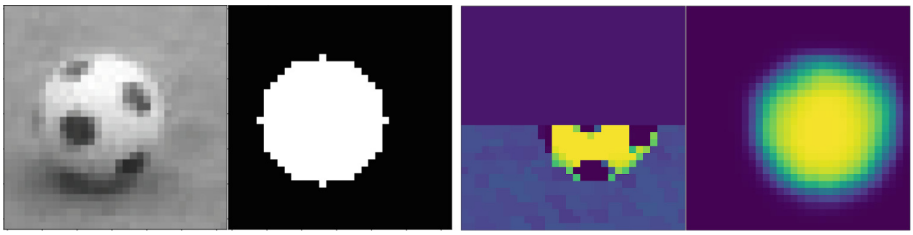

Fig. 3. Training ball segmentation. The left image shows a training example, the right image shows an exemplary inference result.

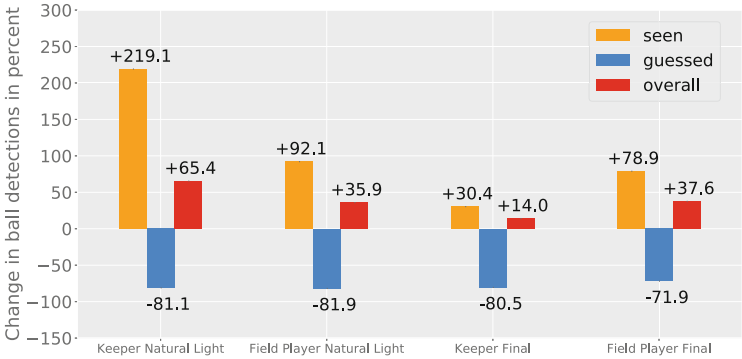

Fig. 4. Change in occurrence of different confidence levels by using the new network instead of the old one that was used at the RoboCup German Open 2019. The changes are in percent based on the detections of the old network.

For both games, the detections of the keeper and a field player from the first half were used. The evaluation was completely automated by considering ball detections as invalid that are not consistent with the team's belief of the actual ball position, which is a feature of the modeling we use in games. This method does not identify all false positives, but it makes it possible to compare both detectors on a relatively large dataset without manual labeling. We use two different confidence levels of detected balls, *seen* and *guessed*, that differ in the threshold that the classifier output must surpass.

As can be seen in Fig. 4, the new network detects overall more balls than the old one. Specifically, the number of actual sightings has risen, while there was a decrease in the number of times patches were classified as guessed. This effect could also be achieved by lowering the thresholds for the guessed and seen labels. Such an adjustment would also result in reduced precision. Figure 5 shows that this is not the case. For a reliable ball detection it is also important to be able to evaluate every frame provided by the NAOs cameras. Using our library for fast neural network inference [7], computation of the shared features requires 192 µs per evaluated patch on the NAO V6 and inference of the classifier 3.1 µs per patch. This is fast enough to allow processing of the 60 frames per second the NAO provides. Additional estimation of the ball position takes 2.5 µs per frame and is therefore almost insignificant compared to encoding multiple patches.

3 Behavior

In the B-Human software, the *behavior* is the component that decides about the actions to be taken with a given world model. Actions are then passed on to the motion system, which generates the actual joint angles and sends them to the controllers. The behavior of B-Human has always been a key part of our success, making use of the mostly correct and precise output from our state estimation and the extensive features and stability of our motion system.

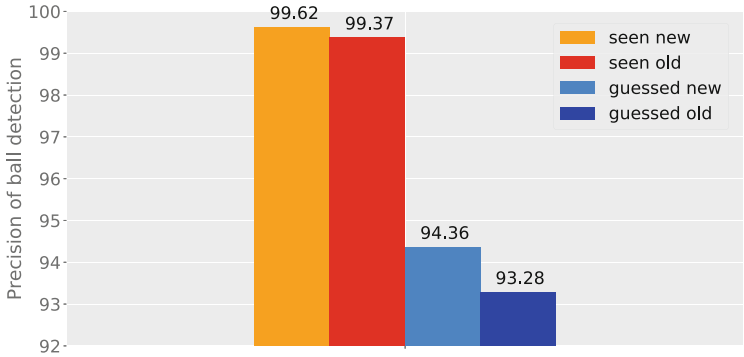

Fig. 5. Precision at different confidence levels of the new and the old network

However, handling uncertainty is still crucial in order to achieve stable decisions. We also often integrate ideas of other teams that we observe to be successful and try to optimize them.

3.1 Defining a Framework

Decomposition and hierarchy are necessary to specify behaviors for complex tasks such as playing soccer. From 2013 to 2017, B-Human used a single hierarchy of CABSL options to specify the behavior. This approach had some shortcomings: Adding or removing high level behaviors required modification of other options. This also meant that an option would often be in a different place than the conditions under which it would become active, which is not easy to maintain. Furthermore, some behaviors are simply not suitable to be modeled with finite state machines, need large calculations, or keep additional state. Some functionality was therefore outsourced to so-called *libraries*, which also spread closely related code across different places.

In 2018, we already started to move away from a single hierarchy of CABSL options (cf. [4, p. 34]). The main problem with the *behavior options* in 2018 was that they could not be passed any parameters, making them useless for behaviors such as walking to a point or doing a specific kick. On the other hand, this anonymity was a desired property to achieve exchangeability of behavior components. We realized that it might not be a good idea to try to fit all behavior levels in the same formalization, but instead split the behavior into two layers: one that would decide *what* the robot should do, where options could easily be added or removed, and another one that would realize *how* the robot fulfills this request.

We call our new framework the *Skills and Cards* system. Skills are separated into an interface declaration and an implementation, where the interface defines the signature of a skill. This allows us to develop multiple methods for the same task, just as representations can be provided by different modules in

the B-Human software (cf. [3]). Skills can call other skills and directly set output representations of the behavior, e.g. the *MotionRequest*. Cards, on the other hand, define a behavior together with the conditions under which they may run. They cannot have parameters and thus do not need separate interface declarations. Cards are organized in decks, which are priority-ordered lists. A so-called *dealer* can choose from them, currently selecting the first runnable card, where *runnability* is determined based on pre- and postconditions that a card specifies. Cards can contain dealers themselves, thereby forming hierarchies of cards.

The following example illustrates the distinction between skills and cards: The GoToBallAndKick skill takes a kick pose and a kick type as parameters. It does not decide whether and where to kick. In contrast, the KickAtGoalCard evaluates whether it is possible to do so and then calls the GoToBallAndKick skill with the appropriate parameters. Of course, the GoToBallAndKick skill is also called from other cards, such as passing to a teammate.

3.2 Passing into Space

For this year, we abandoned the idea of taking passes using specialized motions. Instead of kicking the ball directly at a teammate, we found that it would be much better to pass into space, acknowledging the inaccuracy of our kicks over long ranges. Furthermore, when passing upfield, the ball should end up between the receiving teammate and the opponent's goal because this reduces the time the receiver needs to walk around the ball after reaching it. Therefore, robots positioning for receiving a pass walk to the flanks of the opponent's half, oriented such that the ball can still be seen, but with a tendency towards the center of the field. This way, the passing robot can aim at an area next to its teammate, including a safety margin to minimize the probability that the ball actually ends up behind the receiver.

A particular instance of this kind of passing is the kick-off. While we are not the first team to implement a passing kick-off (the concrete variant with two robots entering the opponent's half is inspired by the team rUNSWift), we seem to be the only team that repeatably executes it with success. The kick-off occurred 10 times during the competition, and while a direct goal after two ball contacts could be scored only once, in all cases our robot had the second ball contact after the kick-off far inside the opponent's half. In some cases, only the opponent's goalkeeper could prevent an immediate goal, which is acceptable and still a strong opening. This also means that we are ready for potential rule changes which require two different robots touch the ball before a goal can be scored.

3.3 Taking Advantage of Set Plays

One of this year's changes in the SPL rule book [2] was the expansion of set plays with the introduction of kick-ins and corner kicks. If not handled properly, these situations could either allow the opponent to easily score a goal or deny a crucial goal in a game. On the highest level, our system distinguishes between

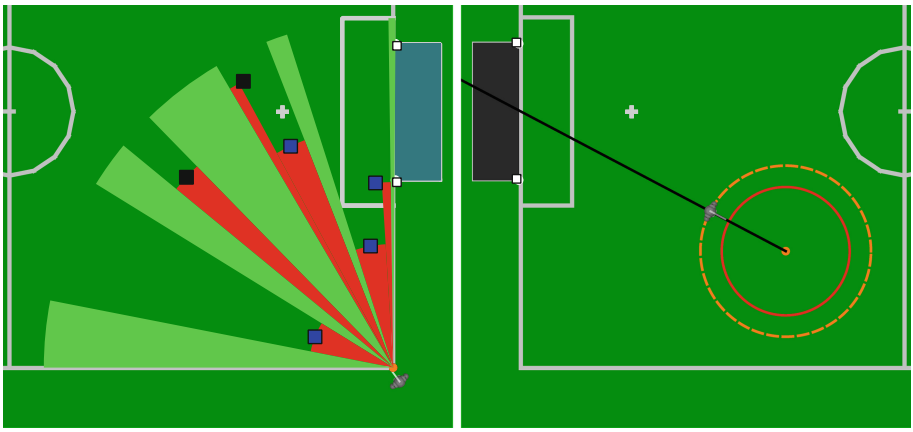

Fig. 6. Offensive and defensive free kicks. The left drawing shows the candidate kick direction intervals (green) and blocked sectors (red) during an offensive corner kick. The right drawing shows a defensive wall building scene. (Color figure online)

own (or *offensive*) and opponent (or *defensive*) free kicks. Both of them have their own deck, although they share most cards with the normal play deck.

The first card in the own free kick deck is a direct shot at the goal, which is the same that is used during normal play and executed as soon as possible. If this is not the case (e.g. due to obstacles blocking the goal or being too far from the goal), different variants exist for the different types of set plays, i.e. goal free kick, corner kick, kick-in and pushing free kick, that try to move the ball closer to the goal and ideally close to a teammate. For example, a striker executing the CornerKickToOwnRobotCard calculates all angular intervals around the ball that point inside the field of play and are not blocked by obstacles (regardless of their team affiliation), as shown in Fig. 6. Up to two of them are chosen, e.g. they must have a minimum width, and broadcast to its teammates. Simultaneously, up to two supporting robots can execute the WaitForCornerBallCard. They listen to the potential kick directions that the striker sends and position themselves such that the expected ball position after the pass is between them and the goal, according to the passing paradigm described above. As soon as the selected kick direction is stable or 15 s have passed, the kick is executed. Stability is determined by checking whether the candidate direction intervals calculated each frame intersect by a certain amount in successive frames for some time.

The behavior during an opponent's free kick, on the other hand, is always aimed at preventing a goal and does so in different ways depending on the possibilities available. All ball-playing cards are removed from this deck, however, there are three special cards in addition to the usual supporting behavior during normal play. Most importantly, one robot tries to form a wall between the own goal and the ball (cf. Fig. 6). The position is chosen just outside the clear area around the ball in an angle such that the own goal is covered as wide as possible. Keeping a robot

close to the ball has the additional purpose to regain ball possession quickly when the free kick is not executed properly. A special case is when the ball is in one of the own corners, such as in an opponent's corner kick. In this case, the robot does not stand between the ball and its own goal but walks to a position close to the sideline. The goal is then still covered by the defenders and the goalkeeper. If a free kick takes place directly in front of the own goal such that a wall between the ball and the own goal is no longer possible, an attempt is made to delay or prevent the opponent from taking the free kick. This is done by walking in front of the opponent which is nearest to the ball. This forces the opponent to walk around the blocking robot (which continuously adjusts its position) or even makes it lose track of the ball. As third option, if there are five robots or more in the defending team, one of them considers marking an opponent. Candidates are opponents that are closer to the own goal than the ball and probably not taking the free kick. The goal is to make the robot unattractive as pass receiver for the opponent or to gain ball possession if a pass is done nevertheless.

Finally, it is important to keep track of the ball during free kicks. At least during goal free kicks and corner kicks, the ball is replaced in a different location than it went out, which may be outside the field of view of all robots. The robots, however, know the type of free kick from the referee computer message. Therefore, they can specifically search the positions where the ball should have been put. Although we make a guess about the side (i.e. left or right) based on the observed ball trajectory, both possibilities are investigated with two robots, if available. For kick-ins, the ball is searched for at the position where it went out, if that has been observed.

4 Reliability

In human soccer, it is considered a huge disadvantage to play with fewer players than the opponent, usually due to a penalty. Unless the level of play is very different between the two opposing teams, this assumption is also valid for robot soccer. Therefore, we try to keep our robots on the field as long as possible, i.e. to avoid receiving penalties as much as possible. In the 2019 SPL Champions Cup competition, we generally succeeded in that goal—with one exception.

4.1 Obeying the Rules

Throughout the competition, our robots received no penalties for *leaving the field*, *illegal defender*, *illegal positioning*, and *illegal motion in set*. The absence of the first three of these penalties is a result to our reliable self-localization [5], the last one originates from our robust whistle detection [3, p. 86] that was improved this year. In fact, the only penalties the robots received were four pushing calls throughout the competition, a result of our good obstacle detection [1] and careful behavior near opponent players. No team received less. The only problem

that we had this year was that the operating system on the NAOs sometimes terminated our software as a response to terminal output, which resulted in 15 penalties for *inactive players*. It took us the first two games in the Champions Cup and one in the Mixed Team Competition to identify and solve this problem.

4.2 Avoiding Hardware Damage

Another reason for a low count in penalties is the avoidance of hardware damage. In general, the new version 6 of the NAO helped a lot with this, because it appears to be a lot more robust than its predecessors. In addition, our robots execute protective measures when they fall, i.e. they kneel down when falling backwards, turn the head straight and pitch it away from the fall direction, and lower the joint stiffness before impact. They determine over which edge of the support polygon they might fall and use an Unscented Kalman Filter to preview whether they actually will. This allows them to start the safety measures very early. An indicator that our fall protection works better than others is the cover of the loudspeakers in the head of the NAO V6. It is a little bit loose and it pops off quite easily when a robot falls, as could be observed during many games. However, our robots never lost this part, most likely due to the lower force with which the heads hit the ground.

4.3 Getting Up

In previous years, our robots had problems with getting up on the artificial grass the field is made of. After a robot fell, it tried different getup motions in a row, from fast and risky to slow and safe, until it could actually get up. This resulted in many failed attempts and sometimes also in *fallen robot* penalties. In addition, the get up motions had to be manually tuned for the carpet at each competition, sometimes even for individual robots.

The main problem with robustly getting up is that NAO's joints sometimes do not follow the commands they were given or at least not as fast as they should. This changes the overall trajectory that the body follows during the get up motion, which often results in the robot falling over again. In particular, this can happen when the NAO has to get its legs together from a wide crouch, which requires a lot of force. The key idea in our current approach is to observe whether each joint of the robot actually follows the commands it was given. If it lags too much behind, the error is distributed over other joints according to predefined weights. Although this cannot produce the same overall trajectory as the original motion, it takes away the load from the lagging joint, which can then catch up more easily to its target angles. As a result, we now only have a single get up motion for each fall direction of the robot, i.e. front and back, with no adjustments needed. All of our robots, even the old ones, get up in their first attempt in more than 96% of the cases if there are no obstacles in the vicinity that could prevent this.

Table 2. Average number of penalties per game (PPG) and average number of robots in play (RIP, also for 1st and 2nd half) for the three SPL competitions at RoboCup 2019

(a) Champions Cup

Team	PPG	RIP	1st	2nd
B-Human	2.7	4.81	4.65	4.98
Nao Devils Dortmund	4.0	4.78	4.76	4.79
TJArk	4.7	4.77	4.81	4.74
Berlin United	4.8	4.77	4.71	4.82
Nao-Team HTWK	5.1	4.67	4.81	4.50
HULKs	3.3	4.65	4.79	4.51
NomadZ	9.6	4.57	4.67	4.46
rUNSWift	9.9	4.50	4.58	4.42
Bembelbots	10.5	4.42	4.35	4.49
SPQR Team	8.9	4.38	4.38	4.39
Dutch Nao Team	12.4	4.20	4.18	4.21
UT Austin Villa	15.3	3.90	4.16	3.60
Average	7.6	4.53	4.57	4.49

(b) Challenge Shield

Team	PPG	RIP	1st	2nd
Camellia Dragons	10.5	4.44	4.66	4.22
SABANA Herons	10.5	4.44	4.50	4.37
NTU RoboPAL	9.9	4.36	4.51	4.21
Starkit	10.6	4.22	4.28	4.16
MiPal	16.2	4.07	4.04	4.10
Naova ETS	14.8	3.96	4.01	3.90
UPennalizers	20.3	2.85	3.16	2.57
RoboEireann	23.3	2.80	2.89	2.72
Average	14.5	3.89	4.01	3.78

(c) Mixed Team Competition

Team	PPG	RIP	1st	2nd
B&B	7.0	5.69	5.63	5.75
Team Team	8.0	5.56	5.70	5.42
Devil SMASH	8.0	5.52	5.56	5.48
SwiftArk	9.3	5.49	5.47	5.51
SPQR-Starkit	16.8	4.60	4.41	4.82
Average	9.8	5.37	5.35	5.39

4.4 Results

Table 2 shows the average penalties per game and the average number of robots that were on the field during actual play. It shows that B-Human was the team with the least penalties and with the most robots on the field. Also in the Mixed Team Competition, where teams play with six instead of five robots, B&B, our joint team with Berlin United, also was the best one in this regard. As can be seen, there is a general tendency to have fewer robots on the field in the second half, in particular in the Challenge Shield. This is mainly due to the incremental nature of the penalty time, i.e. according to the rule book, each penalized robot has to stay 10 s longer off the field than the previous one. However, some teams also start taking robots off the field to save them for the next game, when the current game has basically already been decided in their favor. This is, e.g., the case for the team HULKs, which had fewer robots on the field in the second half, although they have the second-lowest penalty count per game.

5 Conclusion and Future Work

In this paper, we described three – out of many – aspects that contributed to our success in the RoboCup 2019 competitions. While the overall reliability of

our robots is already quite close to the aim of having always the full number of robots on the field, without any penalties or fallen robots, the vision system as well as the team behavior, although they already provided very good results, can still be considered as work in progress.

Due to the much higher computing power of the NAO v6 robots, we are now able to run multiple deep neural networks for ball detection as well as for robot detection in parallel. However, these components are still embedded in our old, partially lighting-dependent vision system that still requires a few manual calibration steps before each game. Thus, currently ongoing research deals with the replacement of these parts, for instance by applying neural-network-based semantic segmentation.

This year's competitions showed that our new ability of playing passes after set plays as well as during the normal course of the game is definitely an advantage. However, so far, we only developed few variants for the respective situations, which is why our robots did not find a proper solution in some situations. As our new behavior framework allows an easy integration of new variants for passes, we are looking forward to an even more sophisticated team play at RoboCup 2020.

References

1. Poppinga, B., Laue, T.: JET-Net: real-time object detection for mobile robots. In: Chalup, S., et al. (eds.) RoboCup 2019: Robot World Cup XXIII. LNAI, pp. 227–240. Springer (2019)
2. RoboCup Technical Committee: RoboCup Standard Platform League (NAO) rule book (2019). https://spl.robocup.org/wp-content/uploads/downloads/Rules2019.pdf
3. Röfer, T., et al.: B-Human team report and code release 2017 (2017). http://www.b-human.de/downloads/publications/2017/coderelease2017.pdf
4. Röfer, T., et al.: B-Human team report and code release 2018 (2018). http://www.b-human.de/downloads/publications/2018/CodeRelease2018.pdf
5. Röfer, T., Laue, T., Richter-Klug, J.: B-Human 2016 – robust approaches for perception and state estimation under more natural conditions. In: Behnke, S., Sheh, R., Sarıel, S., Lee, D.D. (eds.) RoboCup 2016. LNCS (LNAI), vol. 9776, pp. 503–514. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68792-6_42
6. Röfer, T.: CABSL – C-based agent behavior specification language. In: Akiyama, H., Obst, O., Sammut, C., Tonidandel, F. (eds.) RoboCup 2017. LNCS (LNAI), vol. 11175, pp. 135–142. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00308-1_11
7. Thielke, F., Hasselbring, A.: CompiledNN: a JIT compiler for neural network inference (2019). https://github.com/bhuman/CompiledNN
8. Thielke, F., Hasselbring, A.: A JIT compiler for neural network inference. In: Chalup, S., et al. (eds.) RoboCup 2019: Robot World Cup XXIII. LNAI, pp. 448–456. Springer (2019)