Team Report and Code Release 2022

Thomas Röfer[1,2], Tim Laue[2],
Finn Marvin Ewers[2], Enrico Göhrs[2], Michelle Gusev[2], Arne Hasselbring[1],
Jo Lienhoop[2], Ayleen Lührsen[2], Yannik Meinken[2], Philip Reichenberg[3],
Laurens Schiefelbein[2], Florian Scholz[2], Sina Schreiber[2], Simon Werner[2]

[1] Deutsches Forschungszentrum für Künstliche Intelligenz,
Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
[2] Universität Bremen, Fachbereich 3, Postfach 330440, 28334 Bremen, Germany
[3] JUST ADD AI GmbH, Konsul-Smidt-Straße 8p, 28217 Bremen, Germany

Revision: November 1, 2022

Contents

Chapter 1

Introduction

B-Human is a joint RoboCup team of the Universität 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 (SPL) in 2009. Since then, B-Human has won twelve European competitions and has become RoboCup world champion nine times. The 2022 team consisted of the following people (most of them are shown in Fig. 1.1):

Students: Lars Bredereke, Crispin Eichler, Finn Ewers, Jan Fiedler, Enrico Göhrs, Michelle Gusev, Lennart Heinbokel, Alexander Kaiser, Jo Lienhoop, Ayleen Lührsen, Yannik Meinken, Haktan Önay, Laurens Schiefelbein, Florian Scholz, Sina Schreiber, Timon Stahlbock, Simon Werner, Maximilian Wick

Active Alumni: Philip Reichenberg

Leaders: Tim Laue, Thomas Röfer

Associated Researchers: Udo Frese, Arne Hasselbring

This document provides a survey of this year's code release on GitHub[1], continuing the tradition of annual releases that was started several years ago. The description is split into two parts. The more technical information can be found in our Wiki[2]. This document focuses on the new approaches we introduced this year and our specific solutions to the challenges posed by

[1] `https://github.com/bhuman/BHumanCodeRelease/tree/coderelease2022`
[2] `https://wiki.b-human.de/coderelease2022`

Figure 1.1: The members of B-Human in 2022

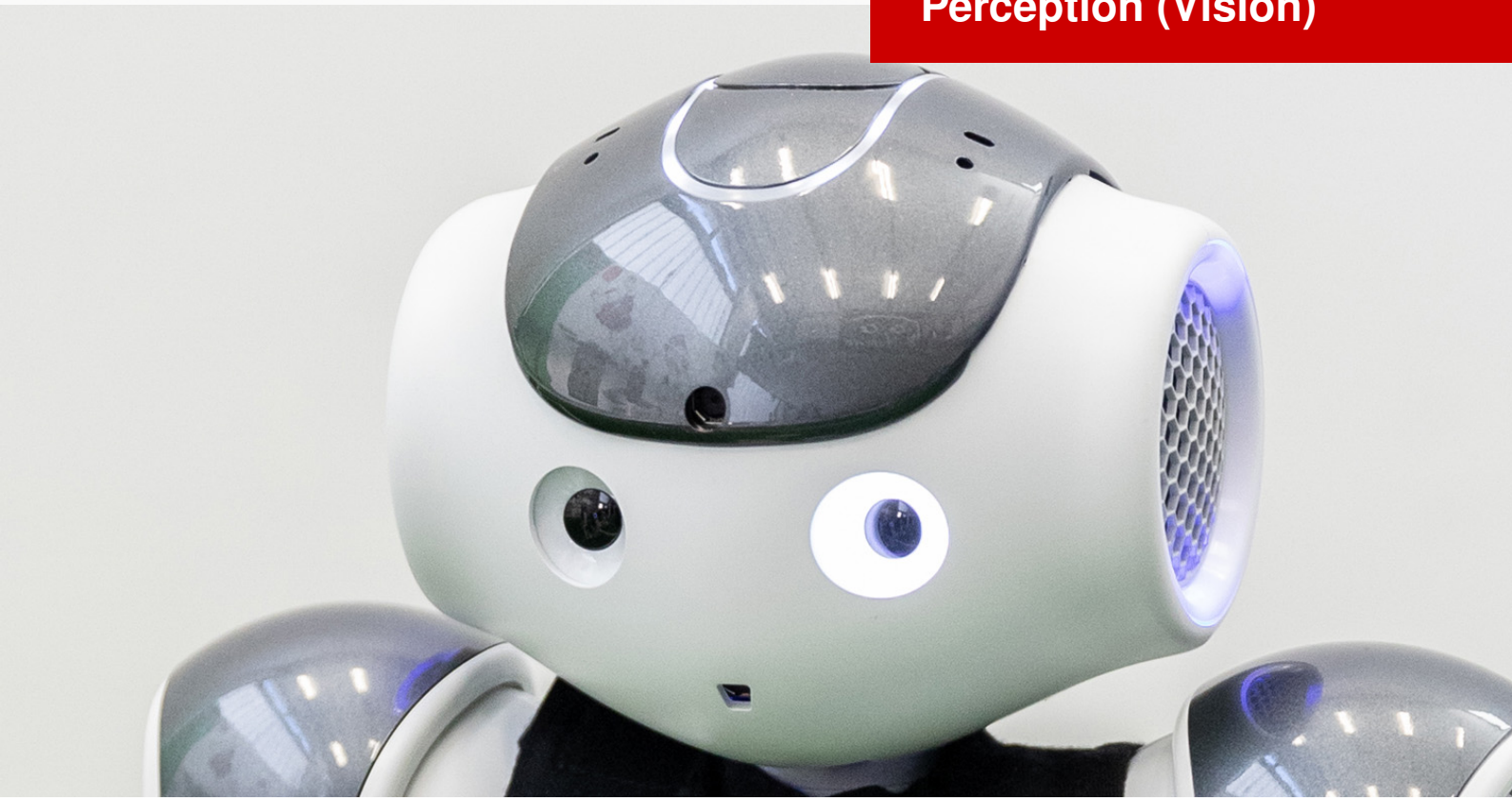
RoboCup 2022. It is not a description of the complete system. Topics that are neither described here nor in the Wiki can probably be found in the description of our 2019 code release [14].

This year, we participated in two competitive events: The German Open Replacement Event (GORE) and the RoboCup 2022. In lack of the official RoboCup German Open in 2022, similar to 2021, the GORE was organized by the SPL community. It was held in Hamburg at the Chamber of Commerce, but this time once again in person for most teams. A modified version of the Swiss-System Tournament was once again used, followed by a knock out stage. After 8 games with a total of 58:0 goals, we won this competition. The RoboCup 2022 itself was held in the Bangkok International Trade & Exhibition Centre in Thailand. By winning all 7 games in the main SPL competition with a total of 48:0 goals and 3 of the 4 technical challenges, our team also won the RoboCup 2022.

In the following chapters, the changes of different parts of the software are described. This starts in Chapter 2 with the introduction of three more convolutional neural networks to our vision system. Chapter 3 mainly describes how we handle the limitation of the communication and the announcement of goals by whistling, which were both rule changes introduced this year. Chapter 4 presents the progress we made in robot motion and the estimation of the torso orientation, joint play, and odometry. In Chapter 5, roles, skills, and modules are described that are used in the behavior control of the robots. Chapter 6 follows up with our approaches to this year's technical challenges.

Chapter 2

Perception (Vision)



The reliable detection of objects in various lighting conditions remains to be a challenging problem for autonomous robots playing soccer. This year's efforts to improve the robots' perception performance focused on moving from classical image processing methods to newer, machine learning based approaches. More precisely, we implemented several convolutional neural networks (CNNs) for the detection of the robots' jersey color as well as the field's penalty marks and line intersections.

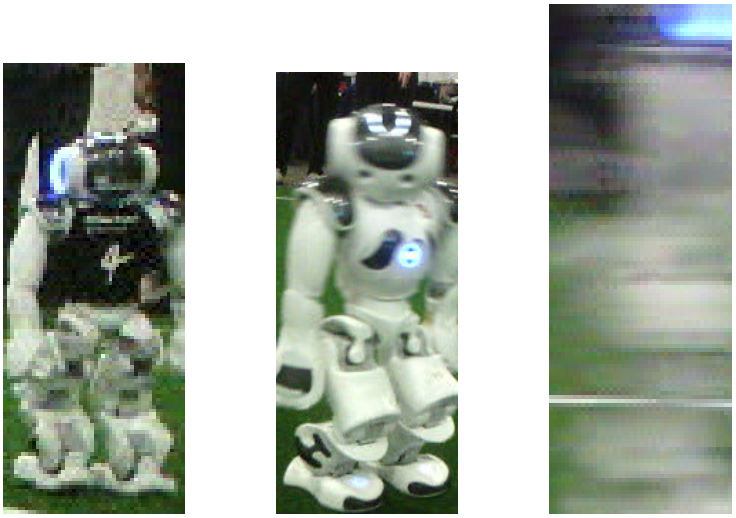
2.1 Jersey Detection

Detecting and classifying the jersey color of robots is a crucial part of the perception layer, since this information is used in the modeling layer to determine the team of a robot when estimating the current world state.

The current approach uses cutouts of robots provided by the `PlayersDeeptector` which are then classified using a CNN by the `JerseyClassifierProvider2022`. The obtained results are then used to determine the team of the robot seen in the image.

2.1.1 Preprocessing

The `PlayersDeeptector` provides an `ObstaclesImagePercept` which contains data about the detected obstacle in the current camera frame. The `ObstaclesImagePercept` holds information about the position and distance of the obstacle. This information is used to cut out the exact area of the image where the robot is located. Since the CNN used is trained on images with a



(a) Teammate (b) Opponent (c) Unknown

Figure 2.1: Cutouts of robots used for training the CNN.

resolution of 20×40 pixels, the cutout is resized using bilinear interpolation to match the input size of the CNN.

2.1.2 Classification

We use a CNN to classify whether a previously acquired cutout of an obstacle belongs to our team. The network returns the probability distribution over the possible three outcomes **Teammate** (Fig. 2.1a), **Opponent** (Fig. 2.1b), and **Unknown** (Fig. 2.1c). The architecture is shown in Table 2.1.

2.1.3 Evaluation

A total of 3590 images was used to train the CNN. As seen in the confusion matrix in Fig. 2.2 the model tends to classify most of the cutouts as teammates. This could be problematic for the parts of the robot behavior that heavily rely on the estimated world model. For instance, most of the actual opponents being classified as teammates could lead to failed passing attempts (see Section 5.3.2).

2.2 Penalty Mark Classifier

Being a rather prominent attribute of a soccer field, the penalty mark can be an important feature for self-localization of soccer playing robots. Thus, identifying the penalty mark as often and reliable as possible is a relevant part of the robots' perception.

In the past, we used a module that checked given candidate regions obtained from images of the upper and lower camera by trying to match the contour of the spot in different orientations and checking for a green surrounding. This approach worked pretty well and produced only a small number of false positives. However, some regions that obviously included penalty marks were discarded, e. g. due to a lack of contrast of the line. Moreover, spots that were more than 2 m away weren't considered at all, since in most cases they couldn't be identified due

Layer Type	Output Size	Param #
Input	$40 \times 20 \times 3$	
Convolutional	$38 \times 18 \times 8$	216
Batch Normalization	$38 \times 18 \times 8$	24
ReLU	$38 \times 18 \times 8$	0
Convolutional	$36 \times 16 \times 16$	1152
Max Pooling	$18 \times 8 \times 16$	0
Batch Normalization	$18 \times 8 \times 16$	48
ReLU	$18 \times 8 \times 16$	0
Convolutional	$16 \times 6 \times 8$	1160
Max Pooling	$8 \times 3 \times 8$	0
Flatten	192	0
Dense	16	3088
Dense	3	51
Total params: 5739 Trainable params: 5691 Non-trainable params: 48		

Table 2.1: Architecture of the jersey classifier

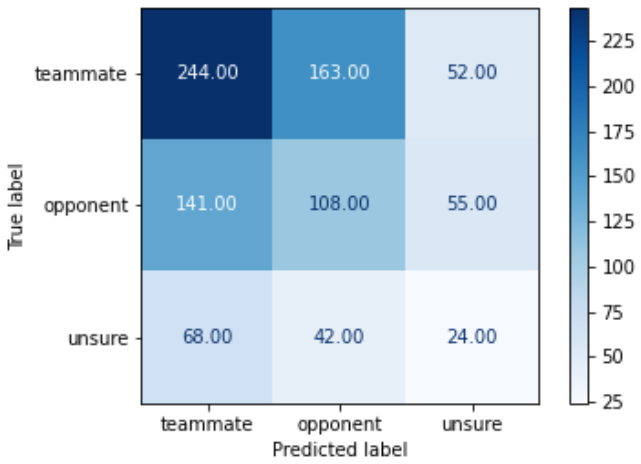


Figure 2.2: Confusion matrix of the jersey detector

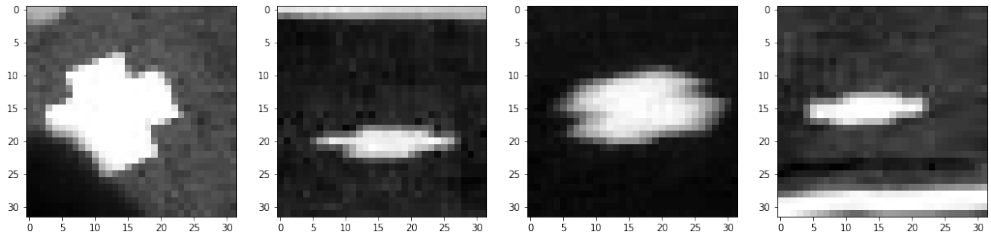


Figure 2.3: Examples of correctly classified regions during the German Open 2022 containing penalty marks.

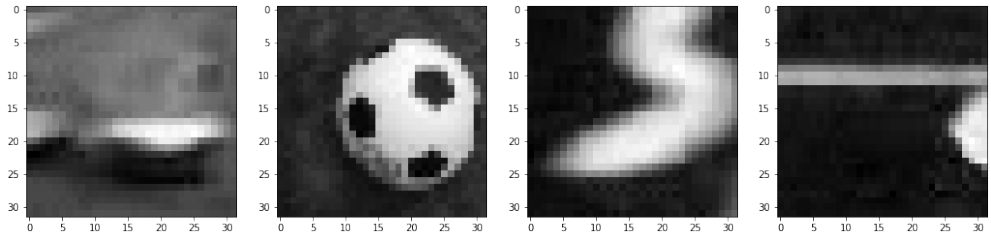


Figure 2.4: Examples of correctly classified regions during the German Open 2022 containing no penalty marks.

to the change of contour in the distance. Hence, our goal was to implement a penalty mark classifier capable of identifying more spots overall, while not generating substantially more false positives. For that purpose, we chose an approach using a CNN for classifying the candidate regions.

2.2.1 Finding Candidate Regions

The candidate regions continue to be provided by the `PenaltyMarkRegionsProvider`, which is described in detail in the 2019 code release [14, p. 69]. The main alteration is now considering regions in the image that are up to 3 m away, as opposed to 2 m in the previous version. Every chosen region is then cropped and scaled, resulting in a 32 x 32 grayscale image, which can be used as input for the neural network. In every image from the upper and lower camera, a maximum of 3 candidates is considered.

2.2.2 Classification

We use a single CNN for the binary classification of the candidates belonging either to class “Spot” or “No Spot”. The network’s architecture is described in Table 2.2.

The network was trained on over 40000 images obtained from logs of B-Human games between 2019 and 2021 with various lighting conditions and different applications of lines on the field. The center of the spot is assumed to be the center of the provided region which is not always completely accurate but close enough in most cases. In the special case of a penalty shootout, we assume the position of the ball to also be the position of the penalty spot, since the ball must be located on the penalty spot, blocking the view of the mark itself.

2.2.3 Evaluation

The performance of the classifier was tested and compared to the previous module using logs of a test game. Overall, 16401 spots were detected by the new Penalty Mark Classifier during the game, 5 of which were false positives. For example, during one half of the game, a single robot detected 844 spots (2 false positives) using the old module with candidates up to 2 m away. When considering spots up to 3 m away, 855 spots (3 false positives) were detected. Using the new Classifier, 1094 penalty marks were detected (1 false positive) in the same time.

2.3 Intersections Classifier

Part of the self-localization is the usage of line intersections as landmarks. Not only is the position of an intersection important but also the type of intersection. Until now, we used a traditional computer vision approach to locate and identify intersections by calculating where perceived field lines coincide. This method proved to be very reliable regarding the detection of intersections. The classification however only provides reliable results for intersections that are not beyond 3 m. Because of this, the goal was to implement a CNN that classifies all intersections that are further than 3 m away.

2.3.1 Candidate Regions

The candidate regions are provided by the `IntersectionsProvider`, which implements the previously mentioned traditional computer vision approach to detect line intersections. The functionality of this module is described in detail in the 2019 code release [14, pp. 66–68]. A candidate consists of a 32 x 32 grayscale image of the intersection which will be input into the neural network where it will be classified.

Layer Type	Output Size	Param #
Input	$32 \times 32 \times 1$	
Convolutional	$30 \times 30 \times 8$	80
Batch Normalization	$30 \times 30 \times 8$	32
Convolutional	$28 \times 28 \times 16$	1168
Max Pooling	$14 \times 14 \times 16$	0
Batch Normalization	$14 \times 14 \times 16$	64
Convolutional	$12 \times 12 \times 8$	1160
Max Pooling	$6 \times 6 \times 8$	0
Flatten	288	0
Dense	64	18496
Dense	2	130
Total params: 21130 Trainable params: 21082 Non-trainable params: 48		

Table 2.2: Architecture of the Penalty Mark Classifier

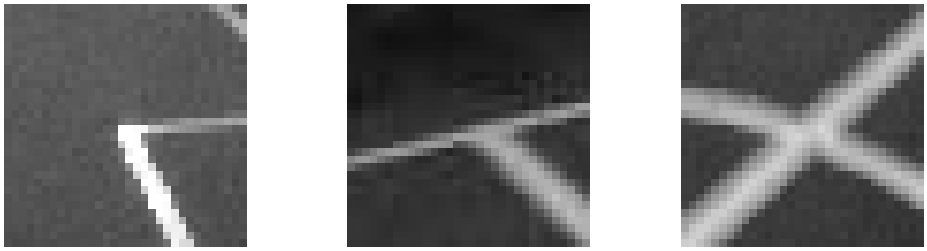


Figure 2.5: Example candidates for each of the intersection types.

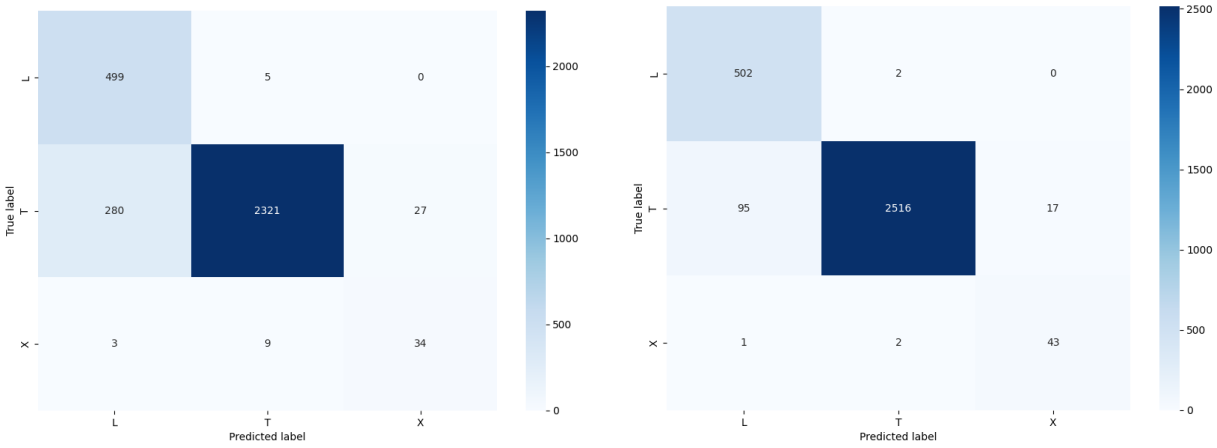
Layer Type	Output Size	Param #
Input	$32 \times 32 \times 1$	
Convolutional	$30 \times 30 \times 32$	80
Batch Normalization	$30 \times 30 \times 32$	32
ReLU Activation	$30 \times 30 \times 32$	0
Convolutional	$28 \times 28 \times 16$	1168
Batch Normalization	$28 \times 28 \times 16$	64
ReLU Activation	$28 \times 28 \times 16$	0
Max Pooling	$14 \times 14 \times 16$	0
Convolutional	$12 \times 12 \times 16$	2320
Batch Normalization	$12 \times 12 \times 16$	64
ReLU Activation	$12 \times 12 \times 16$	0
Max Pooling	$6 \times 6 \times 16$	0
Convolutional	$4 \times 4 \times 8$	4640
Batch Normalization	$4 \times 4 \times 8$	128
ReLU Activation	$4 \times 4 \times 8$	0
Max Pooling	$2 \times 2 \times 8$	0
Flatten	32	0
Dense	64	8256
Dense	4	260
Total params: 17012 Trainable params: 16868 Non-trainable params: 144		

Table 2.3: Architecture of the Intersections Classifier

2.3.2 Classification

A single CNN is used for the classification of candidate regions. The network has four outputs: `L`, `T`, `X` and `NONE`. The first three classes represent the three different intersection types on the field. An `L`-intersection, for example, can be a corner of the field. A candidate will be classified as `NONE`, if no intersection can be seen on the image. The most common case of this is when the `IntersectionsProvider` recognizes an `L`-intersection in the center circle of the field.

Close to 8000 images were used to train the model. These images were extracted from logs of B-Human games from 2019 and 2021 under different lighting conditions.



(a) Old method of classification

(b) Classification with CNN

Figure 2.6: Confusion matrices of both classification methods.

2.3.3 Evaluation

The classifier was tested against the old `IntersectionsProvider` which does not utilize a neural network to classify intersections. A total of 3178 images were used to evaluate both methods of classification.

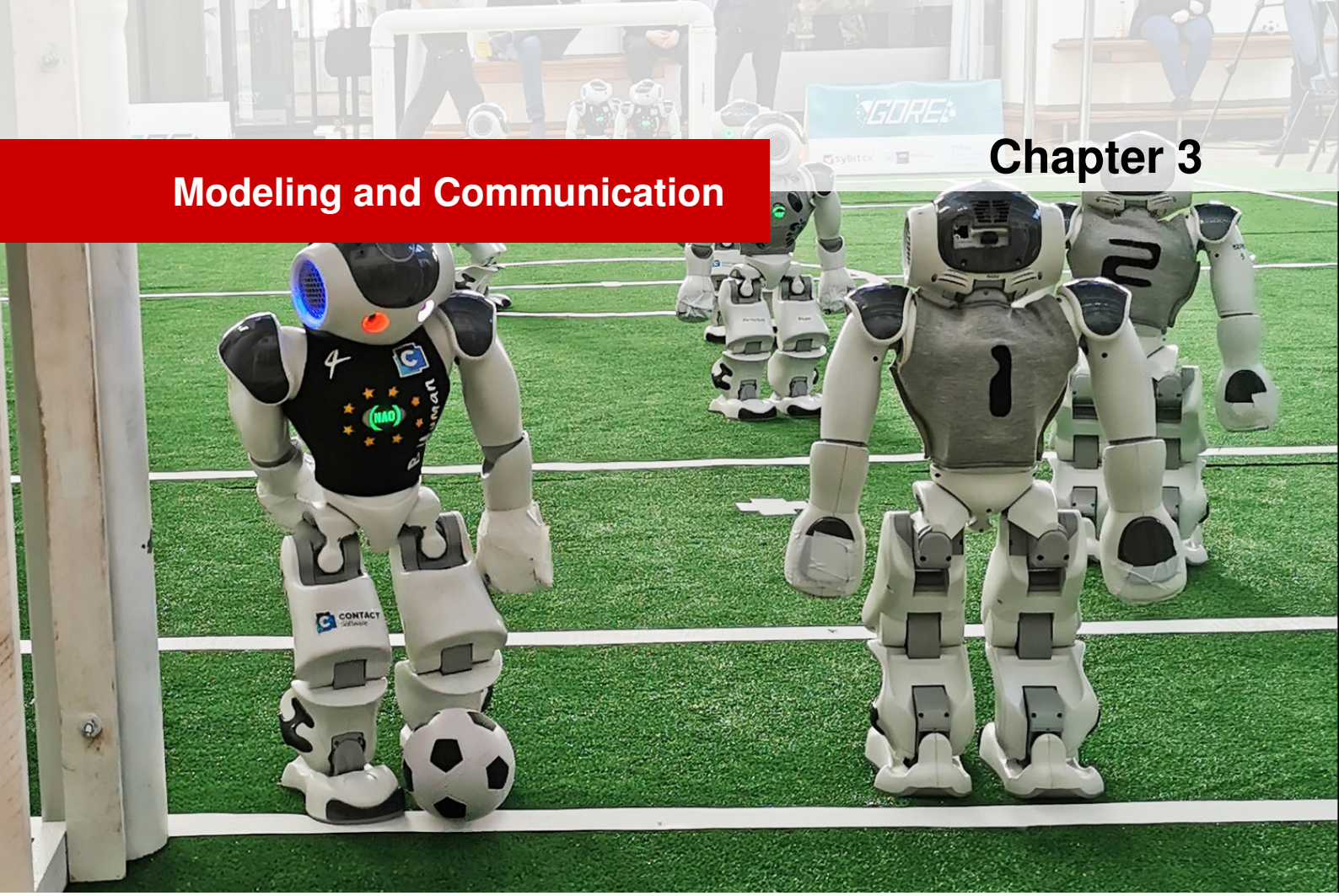
As can be seen in Fig. 2.6, the new classifier turns out to be an improvement to the old one. Especially when it comes to classifying `T` and `L`-intersections.

One noticeable characteristic of this test set is the uneven distribution of intersection types. `X`-intersections, for one, are very underrepresented. This is because the `IntersectionsProvider` excludes lines known to be part of the center circle when calculating intersections.

`T`-intersections on the other hand are very overrepresented. They have been recognized five times more often than `L`-intersections for this test set, even though there are 2 more `L`-intersections on the field. This is because the robots spend a considerable amount of time staying in their own half looking towards the opponent half and searching for the ball. In this position the only intersections that can be seen are the `T`-intersection at the center field line. `L`-intersections can only be recognized when walking to a penalty area. Once the robot is there, most `L`-intersection are within 3 m and thus not considered for additional classification.

Chapter 3

Modeling and Communication



For a proper action selection, each robot tries to accurately estimate the current state of its environment, the so-called world model. This task consists of multiple subtasks, some of which – such as the self-localization and the ball tracking – have remained almost unchanged compared to previous years. However, one new module has been added to detect if a goal has been awarded by the referee (see Section 3.1). Most models are based on perceptions provided by different vision components but some also rely on information communicated by teammates to establish a common world model within the team. Due to recent rule changes that heavily limit the amount of shared information, our communication approach was redesigned (see Section 3.2) and some of our cooperative models had to be adapted regarding their implementation as we well as their purpose (see Section 3.3).

3.1 Goal Prediction

A goal is signaled by a whistle to inform the robots to return to their own half. As normal gameplay is stopped during this phase, it is very important to avoid false positives. Otherwise, the robots would think a goal was shot and the ball could be ignored, while the opponent team could freely play the ball to score a real goal. Therefore, we verify the information of the whistle based on the position of the ball. We compute the distance from the ball to the nearest goal and decide based on that whether a goal for either team is plausible. The maximum distance that is considered as in goal depends on how far the ball is away from the individual robot, as for a robot near the ball it would be worse if he wrongly turned away and walked back to its kick-off position. Additionally, one would assume that such a robot close to the ball has better knowledge of the actual ball position than a robot that is far away from the ball.

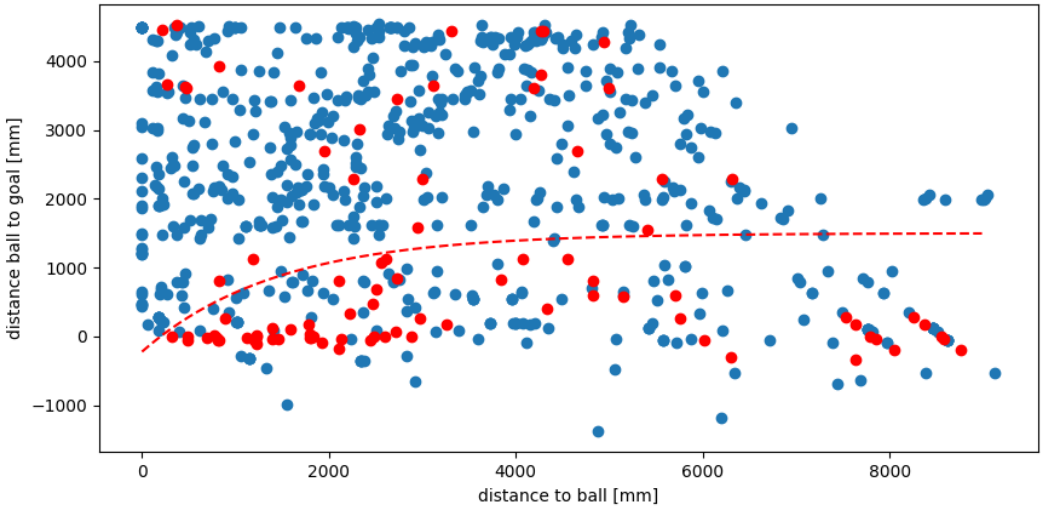


Figure 3.1: Sampled data and threshold function. The horizontal coordinate represents the distance of the robot to the ball and the vertical coordinate the distance from the ball to the nearest goal. Blue dots were sampled during normal play and red dots at the event of a scored goal. The red line is the threshold function.

To decide for a boundary, we sampled the distance of the ball to the goal and the robot based on the perceived world state of the robot during a simulated game, as seen in Fig. 3.1. We sampled these in the event of an actual goal (red dots) as well as every 10 seconds during normal play (blue dots). With this data, we formulated a function based on the distance between the ball to the robot and the goal to decide when a scored goal is plausible. The area below the dashed red line is accepted as a goal. The data from the simulation should be sufficient as the relevant errors of the ball position are most likely caused when the robot did not see the ball himself and has to rely on the maybe old data from his teammates.

3.2 Communication

In this year, the major rule change [4, section 2.5.2] was to limit the maximum number of messages robots can broadcast to their teammates to 1200 per game for the whole team. This is significantly less than the single message each robot was allowed to send at RoboCup 2019 per second. For instance, the 2022 final lasted around 25 min. According to the 2019 rules, the robots would have been allowed to send 7500 messages, but now have to make do with only 16 % of that amount.

The GameController application keeps track of the number of messages each team sent. It broadcasts information about the state of the game to the robots twice per second. The remaining message budget and the remaining game time are part of that information, allowing the teams to monitor their budget use.

Although the main intention of this rule change is that robots should rely more on their own perceptions rather than sharing them regularly, our general approach is that our robots still exchange mostly the same information as before, but limit the communication based on relevance and use a new way to exchange timestamps between the robots.

3.2.1 Limiting Message Exchange

The module `TeamMessageHandler` uses a multi-step approach to sending messages. Firstly, each robot waits at least one second after it sent the previous message before it sends the next one. On the one hand, this limits the maximum sending frequency to its 2019 counterpart. On the other hand, it ensures that the message sent was already counted by the GameController and is reflected in its latest broadcast.

The second step is to make sure that the message budget is never exceeded, because this would nullify any goals scored according to the rules, which would prevent our robots from winning the game. There are two different approaches based on the importance of the information that should be sent.

3.2.1.1 Priority Messages

If the whistle was just detected, the message should be sent as soon as possible, because teammates would interpret a missing message as the whistle not being heard, impeding the majority decision of the team whether the referee actually whistled. Therefore, such messages are sent immediately as long as there is still a budget left (keeping a small reserve), i. e. if the following condition is true:

$$\beta_{\text{remaining}} > \beta_{\text{reserve}} \tag{3.1}$$

$\beta_{\text{remaining}}$ is the message budget remaining as broadcast by the GameController. β_{reserve} is the number of messages we want to keep at the end of the game. This must be at least $\#robots - 1$ messages, because all robots could send a message at the same time and thereby reduce the budget by $\#robots$ at once.

3.2.1.2 Normal Messages

For all other information, a sliding budget is used based on the remaining game time and the remaining message budget using the following condition (ignoring some edge cases here):

$$\frac{\beta_{\text{remaining}} - \beta_{\text{reserve}}}{t_{\text{remaining}} - t_{\text{lookahead}}} > \frac{\beta_{\text{game}} - \beta_{\text{reserve}}}{t_{\text{game}}} \tag{3.2}$$

β_{game} is the overall message budget for a game, i. e. 1200. $t_{\text{remaining}}$ is the remaining time in the game. $t_{\text{lookahead}}$ is a time period into the future, of which the message budget can already be used. It allows to send a certain number of messages at the beginning of the game that are basically borrowed from the end of the game. This was set to 5 s. t_{game} is the overall time of a game. However, the overall duration of a game is unknown, because there are certain periods in a game in which the clock is stopped. This is at least the time until the first kick-off in each half. In playoff matches, the clock is also stopped between each goal and the next kick-off. Therefore, our robots do not send normal messages during such times, not even in preliminary matches. This allows us to set $t_{\text{game}} = 1200\,\text{s}$.

3.2.1.3 Message Relevance

The approach so far would basically result in each robot sending a message every five seconds, but also adapting dynamically to phases without messages during preliminary games (in which the clock keeps running) and messages not sent by robots that are penalized for a while.

Table 3.1: How long sent messages were delayed due to budget limits in the final of the German Open 2022.

Player	Half	#Msg	Delay / Msg
1	1st	63	2.32 s
	2nd	84	1.48 s
2	1st	126	1.51 s
	2nd	107	2.09 s
3	1st	161	1.58 s
	2nd	162	2.13 s
4	1st	141	1.89 s
	2nd	129	1.91 s
5	1st	94	1.97 s
	2nd	129	2.33 s
Game		1196	1.91 s

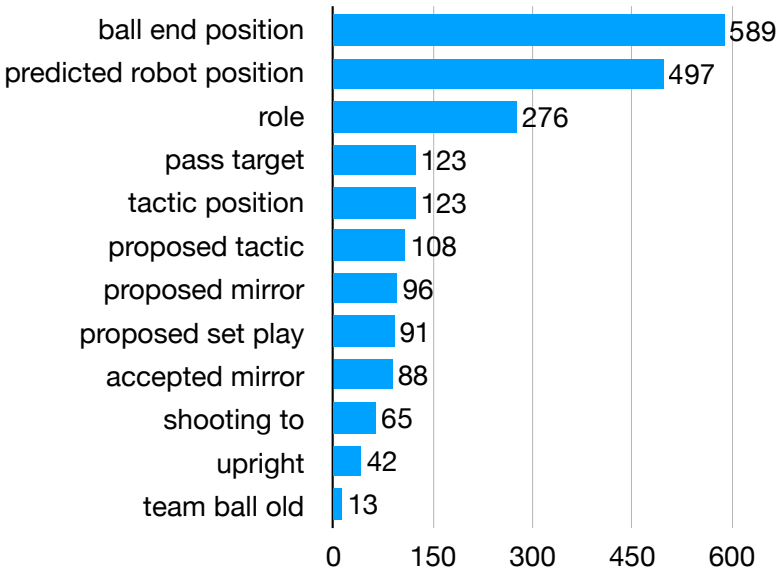


Figure 3.2: Number of times a change was a reason for sending a message in the final of the German Open 2022.

However, whether a normal message is actually sent also depends on its relevance, i. e. whether it would contain information different enough from the previous message sent.

How “different enough” is defined depends on the kind of information. Behavior information such as the current role, tactic, set play, or pass target consists of discrete values, i. e. if one changes, it should be sent to the teammates. Position information such as the robot’s own position, the ball position, or the kick target position consists of continuous values. Here, “different enough” means that it has changed significantly enough from its state sent previously both from the perspective of the sending robot as well as from the perspective of at least one of the receiving teammates. The further away a position is, the less precise it is usually observed. Also, the further away it is, the less important is its precision, because, for instance, kicking over long distances is also imprecise. As a result, the deviation accepted before a message is sent depends on the distance to the position in question. Another opportunity to save messages is to communicate information that allows to predict future positions for a while. The ball position is sent together with its speed. Using a friction model, this allows to predict the position where the ball will come to a halt. Robots also communicate how fast they are walking and where they intend to go. Thereby, teammates can predict their current position. Actually, the criterion for sending a message is whether the information sent previously would still suffice to predict the current position.

3.2.1.4 Results

We analyzed the team communication of the RoboCup German Open 2022 final, which was very similar to the final at RoboCup 2022 (same teams, same score). On average, each message was delayed by 1.91 s (see Table 3.1). This is better than the expected average delay of 2.5 s, when just sending every 5 seconds during the actual playing time. Figure 3.2 shows the reasons for sending messages.

3.2.2 Clock Synchronization

B-Human uses timestamps throughout the system to describe when something happened or will happen (only in some cases, the difference to the current time is additionally provided for convenience). When these timestamps are communicated to other players in a team message, there must be some estimate of the offset of the robots' clocks so that the receiver can interpret the timestamp relative to its own clock. Previously, B-Human embedded an NTP-like protocol into team messages which bounds the clock offset error by half the round-trip time spent on the network, while being most precise if both ways take the same time. Its disadvantage with a limited message budget is that it requires a team message (containing an NTP request) to be sent before a clock offset can be estimated, such that timestamps in the first message received from a specific robot (after the software has been started) cannot be used. It does not help much that messages in the *Initial* state are not counted, since robots that are substituted or rebooted during the game start in an unsynchronized state again. Also, we specifically want to avoid sending messages during the *Initial* state due to the risk that they arrive at the GameController just after the switch to *Ready* and are therefore counted anyway, and we do not want to waste our budget this way.

Instead, we now employ a Reference Broadcast Synchronization [8] based scheme in which GameController packets serve as reference broadcasts. The GameController application sends a packet roughly twice a second, which includes an incrementing 8 bit counter. Therefore, within approximately a two-minute interval, the packet number uniquely identifies a packet and, assuming a single wireless access point and ignoring differences in propagation time to each robot, defines an event that all robots can measure at the same time. The only significant delay that can be different for each robot is the time it takes until the timestamp of the local clock for the receive event is taken. We minimize this by fetching the timestamp from the kernel using the `ioctl(SIOCGSTAMPNS)` call.

To actually calculate the clock offsets, each sent team message contains the packet number of the most recent GameController packet and the local timestamp when this was received. Upon reception of a team message, the receiver looks up the packet number in a buffer of recent GameController packets to find its own local timestamp that corresponds to that packet. The difference between these is the estimate of the clock offset. If the GameController packet number is not known to the receiver, e. g., because it was lost,[1] the previous offset is kept. In the worst case, no offset was known before and the message has to be dropped because the timestamps are invalid. A known offset is also invalidated if the timestamp of the sender is outside estimated bounds, as this indicates that the robot was restarted in the meantime. However, a new offset can be estimated immediately from the received message if the referenced GameController packet number is known, which would not have been possible with NTP. We have measured the resulting system to estimate clock offsets of pairs of robots with millisecond precision, which is what we generally use to represent timestamps.

3.3 Cooperative Models

B-Human has always used the ball and robot information communicated by the teammates to fuse it with local perceptions and thereby create more comprehensive world models. Given the old communication scheme, the receipt of one message per second from each teammate provided a quite dense stream of information and even allowed to fuse ball velocities. However, due to the communication changes described above, the applied approaches needed to be

[1] We think it is impossible for a team message to "overtake" a GameController packet.

adapted regarding their implementations as well as their purposes.

3.3.1 Ball

As described in Section 3.2.1.3, any information about the ball is only communicated, if it is considered as *relevant*, i. e. if a robot computes a ball position that significantly differs from its last estimate or from the last communicated estimate. Thus, in most situations only one robot might communicate a ball position or sometimes even no robot at all. The latter, for instance, happens regularly in periods of time after the ball was placed at a free kick position and the team positions for the execution of the kick. These circumstances render a common ball model more or less useless.

In the current B-Human system, the robots rely on their own local `BallModel` whenever possible. Only if the local model is not considered to be valid, e. g. if the ball has not been seen for some time, the new `TeammatesBallModel` is used. It is computed by the `TeammatesBallModelProvider`, which buffers all recently communicated ball information by the teammates (but not the own ball observations). Technically, these different ball observations are clustered, weighted, and fused. However, as the weighting approach considers the age of any observation, the `TeammatesBallModel` actually represents the most recently communicated ball information in most situations.

3.3.2 Teammates

Similar to the ball, robots only communicate information about their position and orientation, if its change is considered to be relevant. Thus, this kind of information might be quite sparse, too[2]. Perceiving teammates and distinguishing them from opponents is possible (see Section 2.1) but distinguishing the teammates by their numbers is not.

Both kinds of information, communicated poses as well as robot perceptions, are used by the new `GlobalRobotsTracker` to compute the `GlobalTeammatesModel`. As already mentioned in Section 3.2.1.3, all teammates communicate their current walk target and speed, which allows a basic prediction of a teammate's position for a given point of time. Nevertheless, these predictions often deviate from the real position, for instance, if the robot is blocked by opponents. In case of a large deviation, the teammate is supposed to send a new message. However, this might arrive delayed to remain within the planned message budget. Furthermore, minor deviations remain unnoticed and could result in a lack of precision when playing a pass. To overcome these problems, the predictions become corrected by local observations of the teammates, as depicted in Fig. 3.3. In this process, the orientation of the teammates is not considered. Thus, the implementation corresponds to a basic 2D Kalman filter.

3.3.3 Opponents

In the past, we also computed a common model of the opponent robots. As this information was barely used, we completely removed it for the 2022 competitions. Thus, the robots currently solely rely on their local perceptions of opponent robots. However, to allow for more sophisticated behavior strategies in future years, it is currently planned to create a new `GlobalOpponentsModel`. Some stubs, which do not create any usable output yet, are already contained in the previously described `GlobalRobotsTracker`.

[2] Please note that different kinds of information are not sent separately but in one combined message. Thus, if a changed ball position resulted in the sending of a new message, an updated robot pose is sent, too.

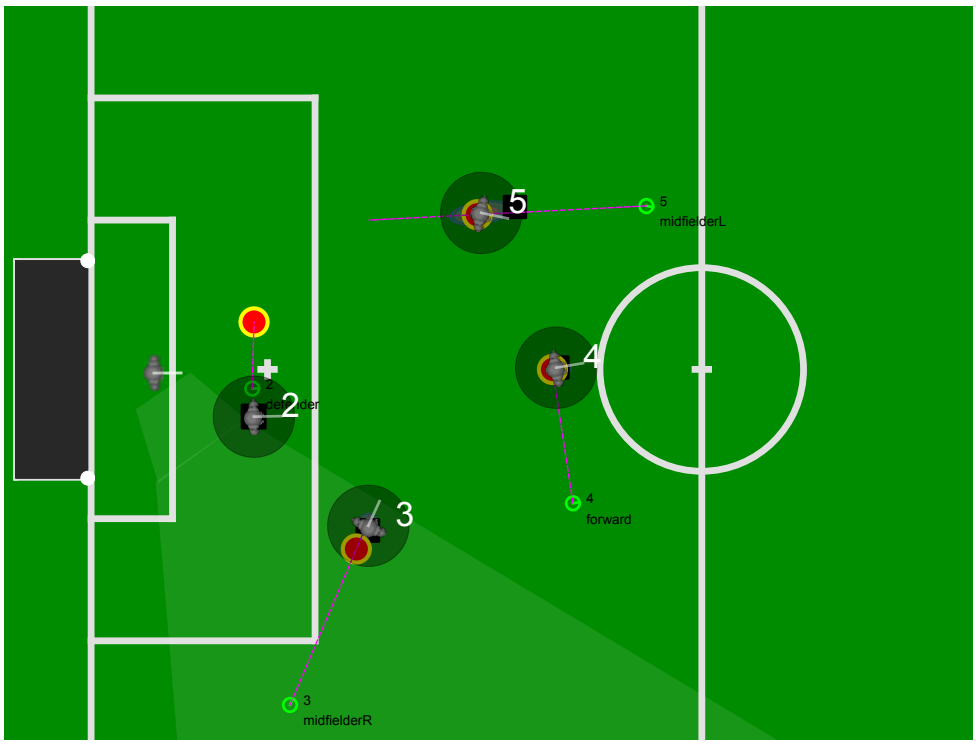
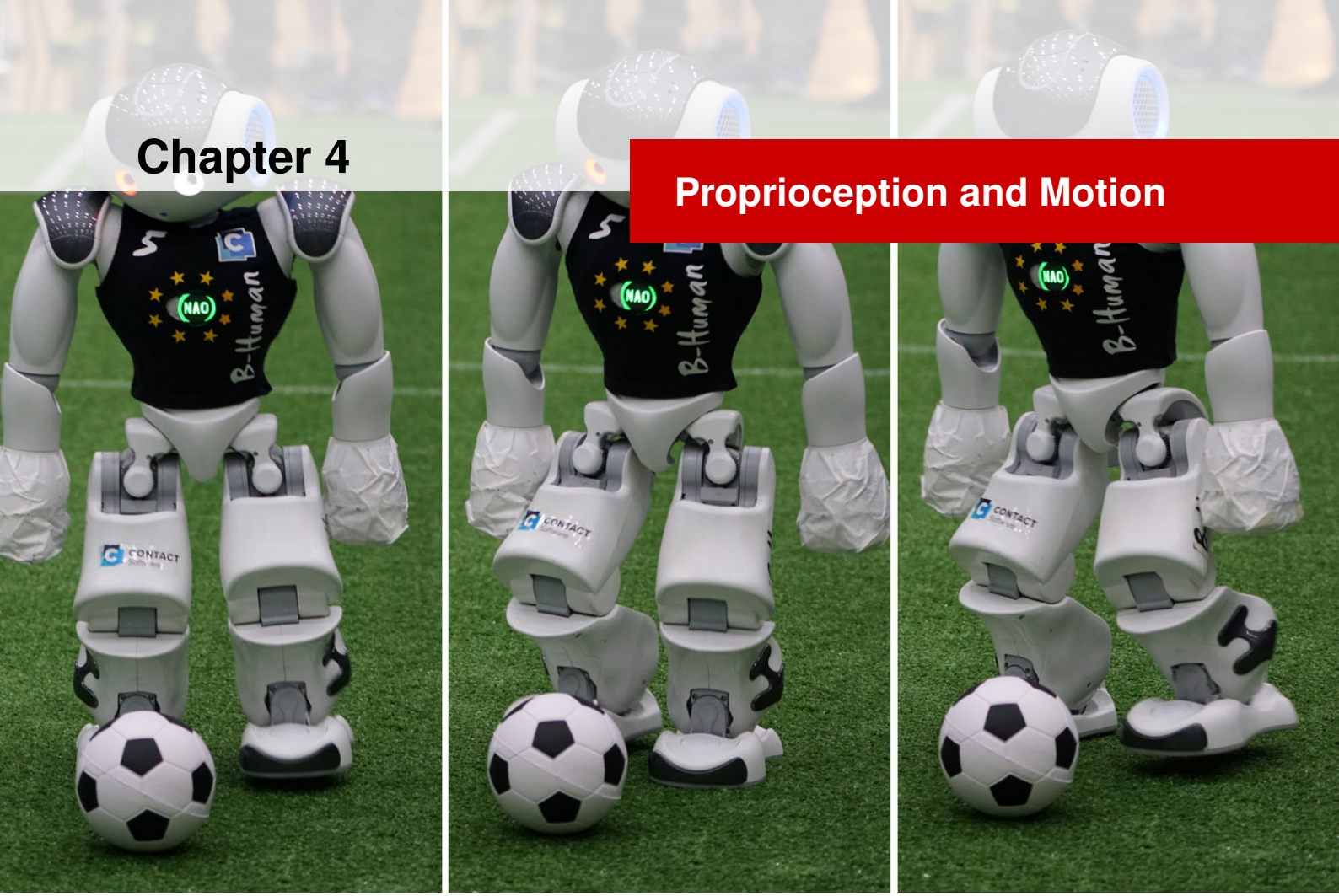


Figure 3.3: Sample situation of the `GlobalTeammatesModel` of the goalkeeper (leftmost robot, depicted including its current field of view). The red circles that are outlined in yellow depict the interpolated positions of the teammates based on their most recently communicated data. The dashed violet lines connect their last known positions with their walk targets. The semitransparent black circles with white numbers show their computed positions after incorporating local perceptions.

Chapter 4

Proprioception and Motion



To allow the robots to play soccer under dynamic circumstances, the sensor data needs to be processed to gain sufficient information about the internal state. This knowledge is crucial for the B-Human motion system, to allow dynamic and fast movements on the field, but also to ensure the well-being of the robots from a hardware point of view.

4.1 Inertial Sensor Data Filtering

As in 2021, we use the calibrated foot sole rotation of the support foot to improve our torso orientation estimation. We changed that it is only used once per walk step and added some restrictions, so it is not used when we cannot know for sure that the whole foot plane has ground contact. This helps to prevent the orientation to drift for too long and overall stabilizes the estimation.

The calibration process for the foot sole rotation determines an offset between the sole rotation, calculated from the joint positions, and the torso rotation, taken from the IMU of the robot. In theory, one can assume that on an even underground with full ground contact, the orientation in the x- and y-axis of the soles should match the torso orientation. This is not the case, because the robots are not perfectly calibrated. If they were, the sole rotations could be used to improve the orientation estimation of the torso while walking, with the assumption that on an even underground and the constraint that the soles need to have full ground contact, the orientation estimation could reset, to match the soles.

With this assumption in mind, we let the robots move to a field position without lines when we calibrate them. Afterwards, they balance on one foot each (see Fig. 4.1), to ensure that all

Figure 4.1: A robot balancing on one foot to calibrate the offset between its sole and torso orientation

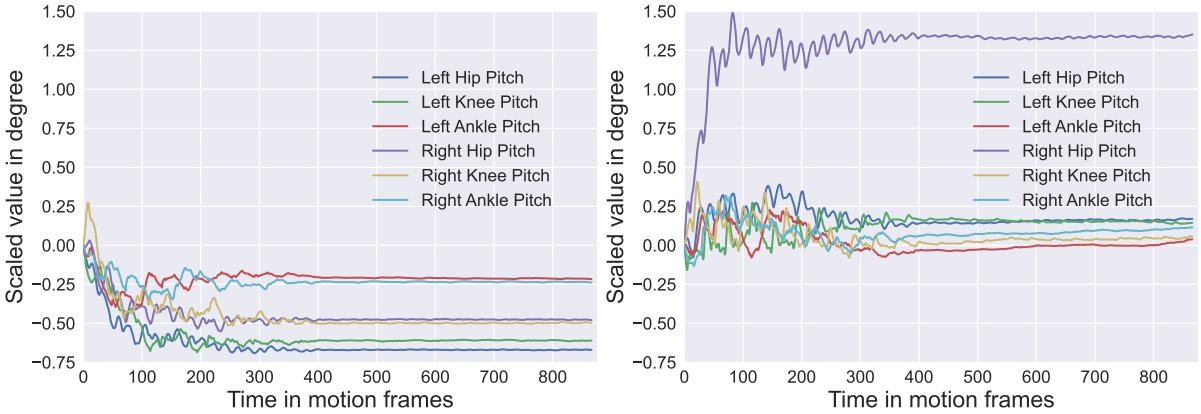
errors in both orientations only come from one leg. We repeat this process once after the robot turned 180° to take the average over two tries per sole.

4.2 Joint Play Estimation

Our walking is currently good enough to let new robots walk with over 30 $\frac{cm}{s}$ without any problems, but worn-out ones still struggle a lot at higher walking speeds. In previous years we would reduce their walking speeds or increase some balancing parameters by hand, which we wanted to eliminate without the tradeoff of slower walking. Therefore, we analyzed how well the joint play can be measured, which we assume is the main origin of why the robot's walk stability varies so much. The module `JointPlayProvider` provides a single value from 0 to 1 for the hardware state.

The idea originates from this year's 7 vs. 7 challenge, which asked for a hardware check for the robots. Here the teams played with the idea of automatically assigning a hardware value to the robots, which should have been used to determine the hardware state of the robots to avoid worn-out ones.

To estimate a value, we calculate the difference between the measured joint positions and the requested one, a few motion frames previously set resulting from the motion delay, when the robots are walking. This value is filtered with a low pass filter, whose parameter is scaled based on how long the filter is already active. This helps to get a fast estimation at the start that becomes further filtered over time.



(a) Joint play values for a robot in good hardware condition

(b) Joint play values for a robot in bad hardware condition

Figure 4.2: Comparison of a new robot and a worn-out robot.

We only calculate the differences for the leg pitch joints, because those are mainly responsible for the stability. Also, we compare the values to those of a new robot and subtract them. A good joint will then always have a value below 0, and worn-out ones always above 0. Based on experience, we also know that a joint play of up to 2°, meaning you can move the joint by hand freely by this much when the robot is just standing, only has little influence. It must be noted that a value of, for example, 0.5 does not mean a joint play of 0.5°, but in reality, it would mean more like 2° to 3°. We also scale the comparison values based on the walking speed, because even a new robot without wear in the gears has higher differences at higher walking speeds than at lower walking speeds.

The calculated values are then weighted, giving the ankle pitches the most weight, summed up and scaled, to get a value between 0 and 1. The usage of the sum is further described in Section 4.4. The result for the different joints can be seen in Fig. 4.2. Here a new robot (left) is compared to a worn-out one (right). The new robot has nearly zero joint play. The worn-out robot has 1° to 2° play, with the right hip pitch having between 5° to 7° play, depending on the position.

4.3 Odometry

The odometry calculation while walking is now implemented in the separate module `OdometryDataPreviewProvider`. This was done because we use odometry to tell how much the robot changed its position based on the sensor data of the current frame compared to the previous one. Therefore, when a new walking step starts, we use the motion request from the behavior and update it with the odometry change. This needs to be done because the behavior and motion run in different threads asynchronous to each other. Additionally, which is the main reason for the usage of a separate module, the change allows us to use the information of how much the robot turned (from the IMU) and translated (from the joint position sensors) within the last motion frame. This gives us some higher accuracy to position behind the ball. Otherwise, this last odometry change would only be available at the end of the motion frame, at which point a new walk step already started and the odometry update would have no influence.

4.4 Walking

Our walk is still based on the walk developed by Bernhard Hengst [9] for the team UNSW Australia/rUNSWift, but extended with the walk step adjustment described in [13].

The walking is therefore mostly the same compared to last year's version. We mainly improved the balancing parts, to allow the higher walking speed of up to 30 $\frac{cm}{s}$, even for very worn-out robots.

Walk Step Adjustment
The adjustment is a bit smoother. The clipping that avoids unintentionally walking into the ball was reimplemented. Now the robots are allowed to walk into the ball in case the delta value for the adjustment gets too large.

Gyro Balancing
The adjustment value of the walk step adjustment was already used to activate specific gyro balancing with the support foot's knee and hip. This procedure was improved and we added a sideways gyro balancing with the ankle roll, scaled based on the center of mass position.

Support Foot Sole Rotation Compensation
Slightly improved the compensation, to result in fewer side effects for worn-out robots.

Jointplay Scaling
We use the calculated value from the representation `JointPlay` (Section 4.2), to scale the allowed maximal walking speed and the thresholds for the special gyro balancing of the knee and hip.

Stepping on Another Robot's Foot
This feature is currently not used because of a too high false positive rate. However, we improved our detection for a switch of the support foot. If the swing foot is measured as the support foot for a few motion frames, a new step is started. This occasionally prevent falls, after stepping on another robot's foot.

Foot Support Switch Prediction
We implemented a new support foot switch prediction, which reduces the sensor delay by one frame. It is currently not used when the robots walk backwards. We noticed that worn-out robots fall significantly less often when walking backwards without the predictions. This is not because of false positives, but because the pendulum swing dynamic would be removed, when the robots just walk on their heels while close to falling.

Joint Position Reset
At the start of a new walk step, the current delta from the walk step adjustment is used to determine, if the leg joints should be set to the measured positions. This is due to the problem that the joints often lag behind by a lot in edge cases. To make sure this does not result in further problems, such as the swing leg still moving backwards when it should move forwards, the last requested joint positions are partially interpolated to match the measured ones to determine new start positions.

Walk Height
The walk height is no longer a pure parabolic interpolation, but the highest point is set to be always after half of the walk duration or after 125 ms, depending on which happens first.

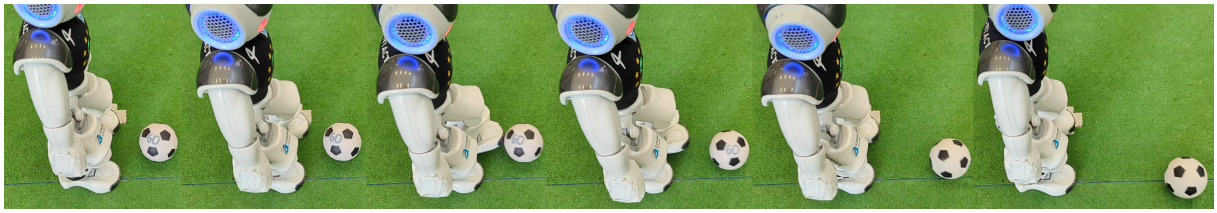


Figure 4.3: The *forward steal* kick. The robot holds a V-shape pose to block the ball and moves one leg rapidly to the ball and back again, to move the ball without walking into the opponent.

Torso Rotation – Speed Scaling

Based on the torso rotation, the interpolation speed for the feet positions is reduced. This helps at faster walking speeds, by resulting in a slower walk, therefore enabling the robots to react in time when they are tilting backwards. Otherwise, the swing foot will have moved too much forward until the next foot support switch. This idea was adapted from the SPL team HTWK Robots [18].

4.5 In-Walk Kicks

The overall structure of the in-walk kicks remains the same compared to the previous version. We replaced our duel kick *forward steal*, which is the kick with the V-form (as seen in Fig. 4.3), and made smaller adjustments to improve the strength of our stronger kicks.

Forward Steal

The forward steal was reimplemented. It still brings the robot into a position behind the ball with a V-shape of the feet. The position is further back to prevent unintentionally stepping on the other robot's foot. Also, the robot does no longer walk sideways to move the ball, but only moves the swing foot inside, to kick the ball, and back to the starting position.

Walk Height

For most kicks, the walk height duration interpolation is increased. This shall insure that the kick will not end prematurely from a support foot switch.

Precision Modes

We added the *justHitTheBall* precision mode. Most starting conditions are deactivated and precision thresholds are looser. This mode is only used for duels to ensure that the robot just executes the kick as fast as possible without wasting a few seconds, with the tradeoff that often the ball is not even touched or the kick direction has more deviation. The other two modes, *precise* and *notPrecise*, only change how much the ball can differ relative to the swing foot, to influence how precisely the robot shall position itself behind the ball.

Also, the kicks are still configured based relative ball positions, which are converted into walking steps, as seen in Fig. 4.4

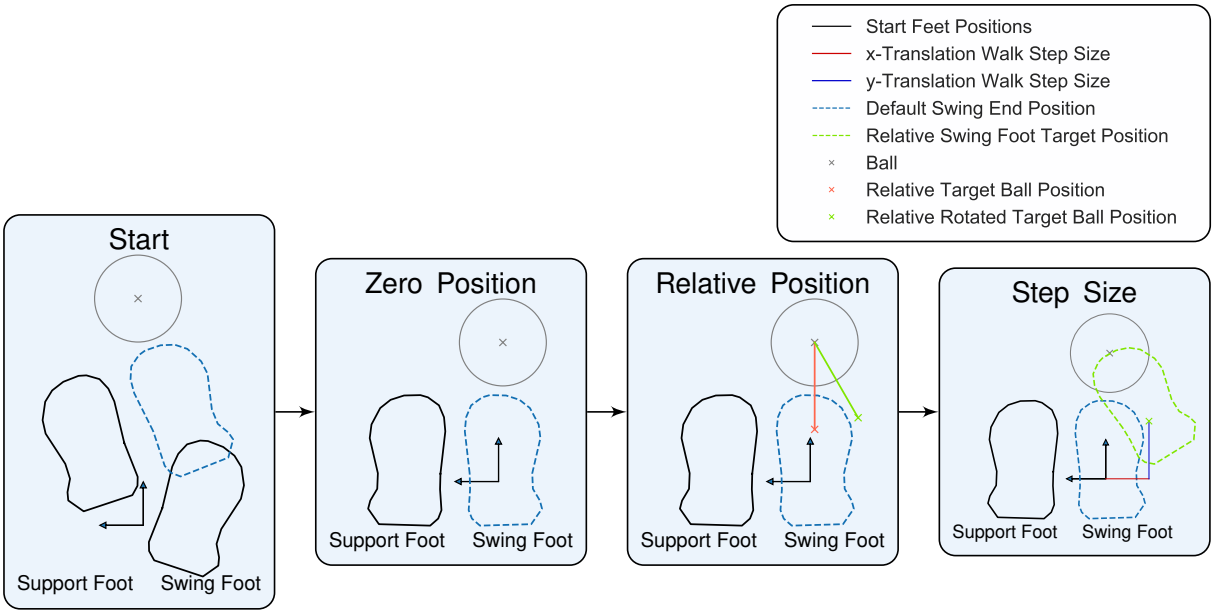


Figure 4.4: The calculation flow to generate a walking step to kick the ball, starting from a previous walk step.

4.6 Falling

The fall motions are further improved. For the back fall[1], the knees are no longer fully closed, but are kept open for a few degrees. The knees are used as a spring to buffer the kinetic energy from the fall and to prevent the heels to receive all of it in one instant. Also, after reaching a specific tilt, the legs are stretched out again and moved upright, to make sure the robot tilts over its arms and releases the stored fall energy from this motion, instead of colliding with the ground.

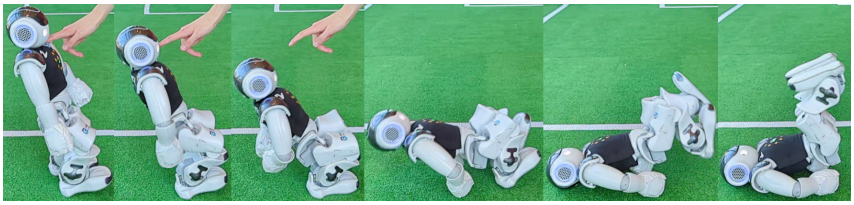
For the forward fall[2], the pose is more asymmetrical. This helps to let the robot fall a bit more diagonally. Also, just as for the backward motion, the legs are stretched out to spread out the kinetic energy more. Both motions are shown in Fig. 4.5.

4.7 KeyframeMotionEngine

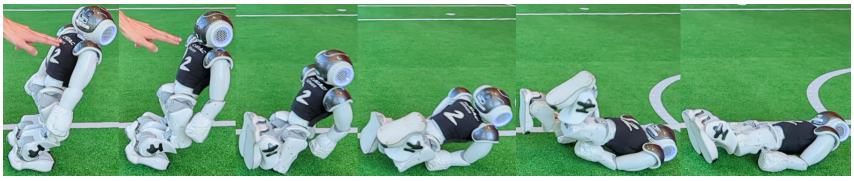
The implementation of the `KeyframeMotionEngine` is now separated into multiple files, grouped by their context, such as balancing or joint compensation. Also, the keyframe selection is now a branching system. A single motion such as getting up from the front is now explicitly set to have multiple keyframe blocks. Only those simple motion definitions can be executed from the outside. The keyframe blocks contain the specific keyframes, with the joint positions and the balancing parameters.

The conditional keyframes are now keyframe blocks too and every keyframe has a *keyframeBranches* parameter. In those, you can define which other keyframe blocks or motions shall be executed given some conditions. This allows for loops, auch as a robot that reached a sitting motion, falls backwards and catches itself to continue getting up from an upright position over and over again. Also, multiple branches can be defined with more clarity.

[1] `https://youtu.be/Pe6cBS_0BK8`
[2] `https://youtu.be/r7AEEtXkElg`



(a) Backward fall motion



(b) Forward fall motion

Figure 4.5: Illustrations of our fall motions

4.8 Energy Saving Mode

The energy saving mode was improved. The adjustments jump less often and when too many joints have too high offsets, only the joint with the highest current is adjusted. Also when standing, the maximally allowed offset is now based on the center of mass and a defined stable area. Additionally, when the robot starts the energy saving mode, a base offset is used based on the differences between the requested and measured joint positions. This allows the robots to remain in a standing position, which is not the high stand, without increasing the joint temperatures even after an hour.

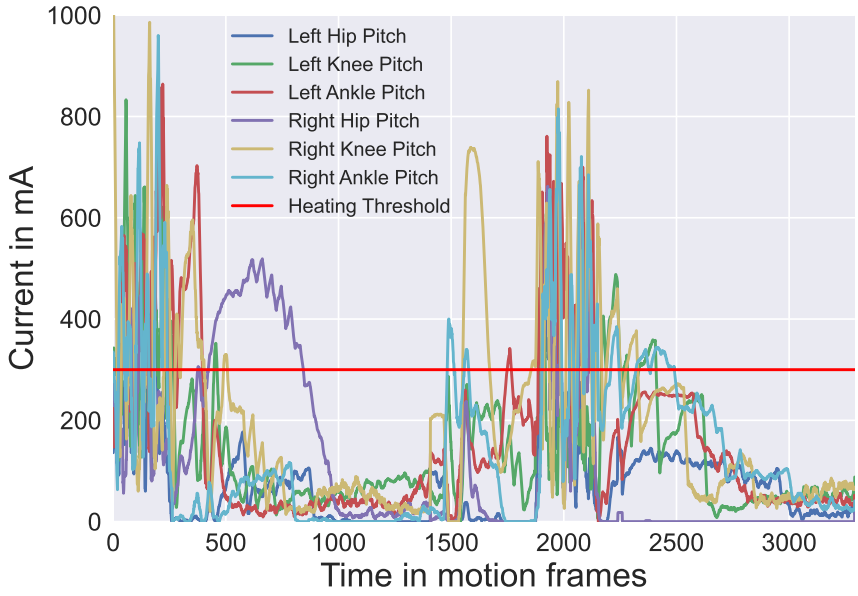
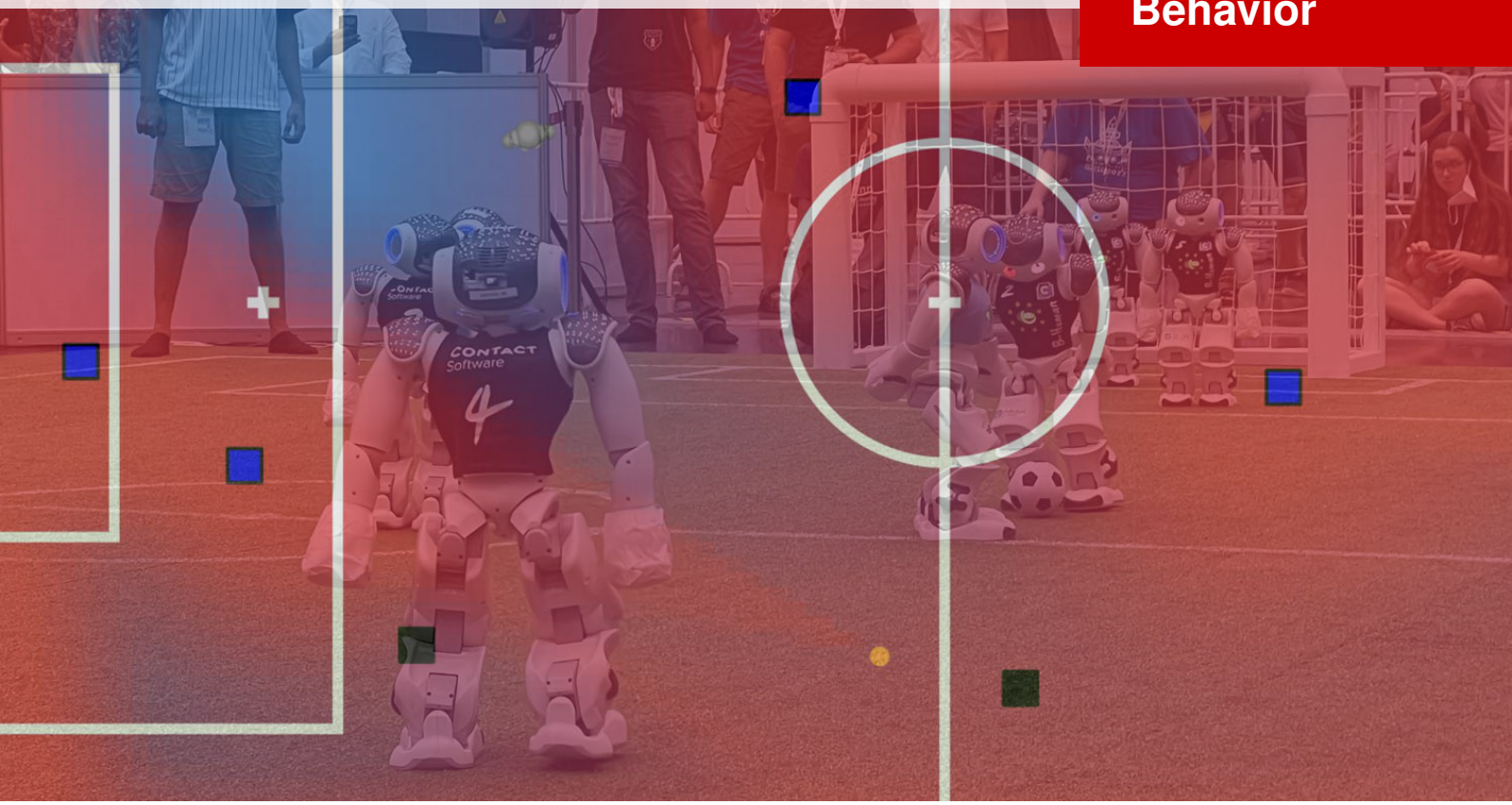


Figure 4.6: The currents of the leg joints after walking and a transition into a high stand in a kick-off (about frame 400), followed by a transition into the normal stand (frame 1400), a short walk phase (about frame 1900) and a normal stand phase (about frame 2500). The red horizontal line marks the 300 mA threshold, at which the joints start to heat up. The data jumps at 1400, because we removed the long time period where the robot was just in its high stand position.

Chapter 5

Behavior



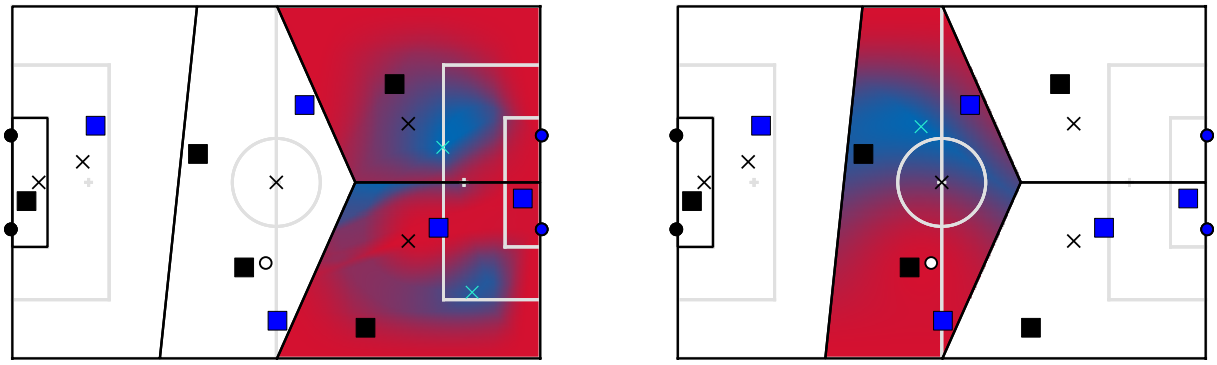
This year, B-Human developed a new architecture for the behavior, which will be described in depth in the B-Human Wiki [2] a couple of months after this code release. This chapter is focusing on the approaches we used in 2022 for improving the skills of our robots.

5.1 Roles

5.1.1 Forward

The duty of a forward player is mainly to shoot goals for its team. Whenever a teammate is currently playing the ball, the forward role has to receive a pass in order to be able to shoot at the goal. For determining the optimal position to achieve this task, the search space is limited to the area that the role is currently assigned to i. e. the corresponding Voronoi cell resulting from the team's tactic. A tactic defines the general formation of the team by specifying exactly one base position for each role that can be used as seeds for generating a Voronoi diagram [15].

As we have rating functions for passes (Section 5.3.2) and goal shots (Section 5.3.1) we construct a new rating function that combines these with a normal distribution around the base pose to encourage positions near to it. They are combined by multiplication and clipped with the Voronoi region. The robot's ideal position should be at the maximum of the rating function [15], as shown in Fig. 5.1a. The search uses gradient ascent which may get stuck in a local maximum. However, this procedure is computationally cheap especially since we can start each frame at the optimal position found in the last frame [15].



(a) Rating of positions for both forward players.

(b) Rating of positions for the midfielder.

Figure 5.1: Normalized rating functions of the position roles inside their corresponding Voronoi regions (black lines). The crosses mark the base poses (black) and found maxima (cyan). The ball is located at the white circle. Squares indicate teammates (black) and opponents (blue). The goalposts are visualized by circles in the color of the respective team [15].

5.1.2 Midfielder

As the midfielder has to play both in the offense and defense, the description of its behavior is slightly more complex than for the forward player. The midfielder attempts to stay away from its teammates to ensure high field coverage and to be a potential alternative pass target while also allowing for a quick regain of ball possession [15]. Additional factors such as the distance to the field border and the distance to the base pose are taken into account [15]. In the case that the ball is in the opponent's half, we also shift the base position slightly towards it to allow for quick pressing when the opponents regain the ball. The resulting rating is shown in Fig. 5.1b. With this, we construct a rating function and search for its maximum like for the forward (Section 5.1.1).

One special case is when we have a free kick as that is the only state in which we know for sure that we are on the offensive. In that case, the rating function is similar to the forward but with less weighting on the probability to shoot a direct goal.

5.1.3 PlayBall

This role implements the default ball playing behavior. It is assigned by the behavior to the one agent that can reach the ball most quickly in the direction of the opponent's goal.

The role's intention is calculated by the so-called *SmashOrPass* algorithm, which decides whether to shoot at the goal or pass the ball to a teammate [15]. The probability of success of these actions are estimated by rating functions with hand-crafted features [1], which are described in Section 5.3.1 and Section 5.3.2.

The legacy algorithm is still included, which does not consider the obstacles from the world state when making its decision, and can be activated via a Boolean parameter.

5.2 Skills

5.2.1 PassToTeammate

The pass skill is executed on the passing robot when the strategy behavior control sets a pass skill request and no option for special case handling is currently being executed.

First, the position of the passed-to teammate, i. e. with the matching jersey number from the parameter of the skill request, is determined based on the received team messages.

Then, the obstacle model is used to compute an angular sector wheel from the perspective of the ball, containing free sectors and those blocked by obstacles. From this, a pass angle and kick target is calculated as close to the teammate's position as possible, while reducing the likelihood of obstacles blocking the ball on the way [15].

Among the types of kicks matching the distance for the pass, we select the one that will likely move the ball most quickly and accurately to the pass target based on the kick's range and the robot's time to reach the required kick pose behind the ball. The selection of available kicks can be limited based on the current situation and game state. For example, an own goal kick can utilize the strongest available kick, which can roll up to 8 m and deep into the opponent's half, to immediately switch from defense into offense. This kick is not available for passing during the regular game though, mainly due to the long range, but also because it requires the robot to stand still and therefore makes its execution too slow when opponents are nearby.

During set plays however, the rules guarantee that there is no opponent allowed in a radius of 75 cm around the ball and there is sufficient time for the executing robot. Therefore, the player might wait until the time of this state is almost up before executing the pass. This will increase the probability that all teammates have reached their target positions as specified in the team plan for the current set play. The waiting pose behind the ball is calculated based on the planned trajectory and selected kick type so that the adjustment steps needed when executing the kick are minimized. However, to not unnecessarily delay the game, the robot can only transition into the waiting state when obstacle sectors are blocking the teammate's position in such a way that no wide enough free corridor is available and a pass is estimated to be unlikely to succeed. Otherwise, the pass is executed without waiting behind the ball, which will always be done during the regular game.

5.2.2 ReceivePass

This skill is executed on a robot that knows that a teammate is planning to play a pass to it, i. e. a received team message contains a pass target which matches the number on this player's jersey. The communicated shooting target, to which the other teammate is planning to kick the ball to, is used as the target position for the receiver to walk to. The orientation of the receiver will be set so that the shooting teammate is in sight and the receiver will see the rolling ball before it arrives, to perform the necessary adjustment steps for catching the ball.

5.2.3 WalkPotentialField

According to the rules there are specific areas of the field that are illegal for robots to be in in certain situations [4, section 4.4]. Being inside these areas results in a time penalty, which is to be prevented. The specific illegal areas are:

- The own penalty area, if there are three robots of the own team inside. During a penalty

kick, the own penalty area becomes an illegal area for all robots of the team except of the goalkeeper.

- The opponent penalty area, if three robots of the own team are inside. During a penalty kick, the opponent penalty area becomes illegal, if the entering robot is neither the defending goalkeeper nor the attacking robot.
- The opponent half, during a kick-off of the opponent team. During a kick-off of the own team the opponent half becomes an illegal area, except of the whole center circle.
- The center circle, during a kick-off of the opponent team.
- The ball area, with a radius of 75 cm around the ball, during free kicks.
- The border strip, excluding the border lines, during penalty kicks.
- Everything except for the own goal line, for the goalkeeper during penalty kicks.

The representation `IllegalAreas` provides functions to check whether a given position is (or will be) inside of an illegal area that can be expanded outwards by another parameter. It should be noted that the 5 cm wide field lines are part of the respective area that we have extended by an additional buffer area of 25 cm.

When a robot is about to enter an illegal area, the `HandleIllegalAreas` option will prevent the robot from walking inside (i. e. becoming illegal) by making it stand and observe the original target inside of the illegal area. Once the area around the original target becomes legal again, the robot will continue to walk directly to it. For example, a defending robot will wait at the border of the ball area during an opponent free kick until “ball free” was called by the referee.

When the robot already is inside an illegal area, the `WalkPotentialField` skill will be called. This skill is using the `FieldDimensions` representation to calculate the currently illegal areas in field coordinates. Then, a potential field is calculated by applying a maximal rejecting force inside the illegal area. Starting from the border of the illegal area, the rejecting force is proportional up to a maximum distance outwards. If the robot detects that its current position is inside of an illegal area, the robot leaves it by calling the `WalkToPoint` skill with a relative angle given by the potential field. Additionally, the robot’s orientation while walking outwards the illegal area can be set in a way that the target of interest (e. g. the ball during a free kick) is kept in sight. The debug drawing provided by the skill is shown in Fig. 5.2.

5.2.4 DribbleToGoal

The *DribbleToGoal* skill dribbles the ball to the goal, but tries to avoid obstacles and handles the field border to not dribble outside. For this the potential field from the representation `FieldRating` from Section 5.3.3 is used. Based on the last direction, the potential field direction is 1 cm ahead determined in a loop, until a distance of 1 m is reached. The end position is then used to calculate the base dribble direction. Afterwards, a sector wheel and the distances to the field border are used, to determine the nearest best dribble direction as seen in Fig. 5.3

5.2.5 Zweikampf

In situations, in which one of our robots wants to play the ball and an opponent robot is close, it uses the *Zweikampf*[1] skill.

[1] *Zweikampf* is a German word. In the soccer context, it refers to situations in which two players are both close to the ball and want to gain or keep ball possession. This includes situations with significant physical interaction, such

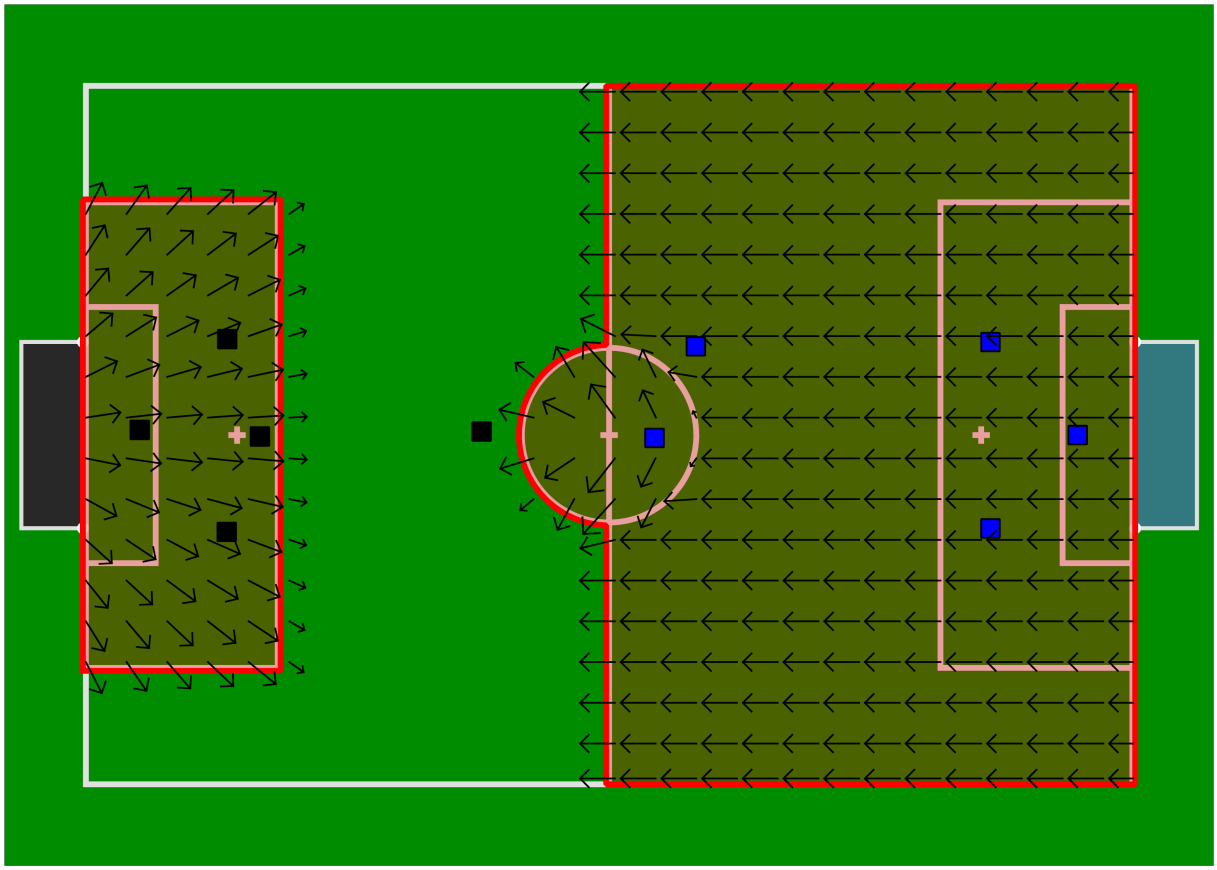


Figure 5.2: Visualization of the illegal areas (red rectangles) and potential field (arrows) in the robot's world view during an opponent kick-off from the perspective of the black team (left). The own penalty area is illegal, because more than three teammates are inside.

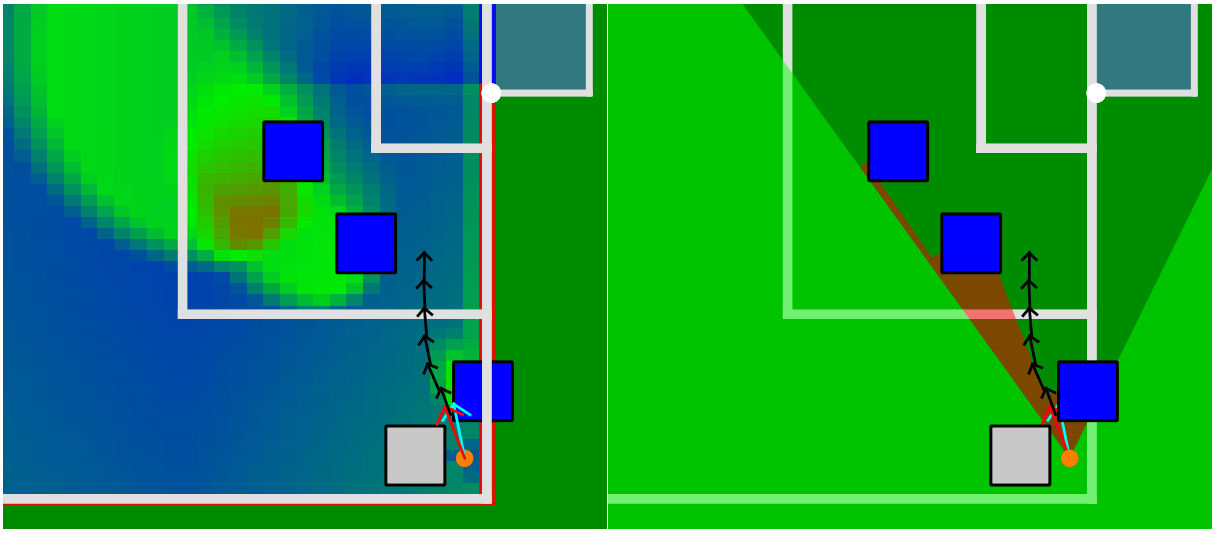


Figure 5.3: The potential field (left) and the sector wheel (right) used in the *DribbleToGoal* skill. The cyan arrow shows the dribble direction from the potential field, the red arrow the actual executed one. The black arrows show the intermediate steps while iterating over the field points.

A subset of possible kicking directions is generated based on the last planned kick direction. Afterwards, we check for every kick direction some constraints and a subset of ratings from a potential field (Section 5.3.3), provided by the representation `FieldRating`. Then we determine the final rating for every kick type for the given direction. Here, a handful of kicks are filtered through some additional constraints. Also, kicks that place the robot into another robot are given worse ratings, while goal kicks and kicks similar to the previously determined kick are given better ratings.

Now we have a list of kick types with directions, ranges, and ratings. They are sorted into four categories: goal, steal, pass, other. A kick of type goal is always better than every other kick, even if the rating is a lot worse. These categories are used to make sure, given the current context, a good kick is always selected.

Last but not least a precision range is calculated. This information is used by the thread *Motion* to start a kick early because it then knows how much the kick direction is allowed to vary, while still executing walking steps for the target direction.

In case no kick was calculated because all kicks were filtered out, the robot will simply walk between the own goal and the ball. After a short waiting time, the *DribbleToGoal* (Section 5.2.4) skill is started to resolve the situation.

5.3 Modules

5.3.1 ExpectedGoals

This new module estimates the probability of scoring a goal when shooting from a hypothetical ball position [1]. Two different approaches are presented. The main difference between the two functions is that one takes into account the known obstacles from the world state [11] and the other one ignores them in favor of robustness and performance. Both approaches consider the distance to the opponent's goal for rating goal shot opportunities. The first approach, which is currently used by most parts of the behavior [15], calculates the opening angle on the opponent's goal from a given position, by finding the largest angle range in the sector wheel that is between the goalposts and not blocked by obstacles [12]. This criterion is used to estimate the probability that the goal shot will be successful [3].

5.3.2 PassEvaluation

This new module evaluates a hypothetical target position of a pass by estimating how likely it is that the ball would arrive there when kicked from its current position [1]. This takes into account the positions of the opponents from the world state of the robot to assess the risk of an interception of the rolling ball [7]. The rating function is described in detail in [11] and is used by several parts of the behavior, most importantly the positioning as described in Section 5.1.

5.3.3 FieldRatingProvider

The module `FieldRatingProvider` provides functions to calculate a potential field for the *Zweikampf* (Section 5.2.5) and *DribbleToGoal* (Section 5.2.4) skill. Different potential fields are supported, which all use a base computation of linear scaling from a repelling or to an

as tacklings, as well as skillfully tricking the opponent without any physical interaction. We did not find any suitable translation to English that describes what our robots are doing and that we like.

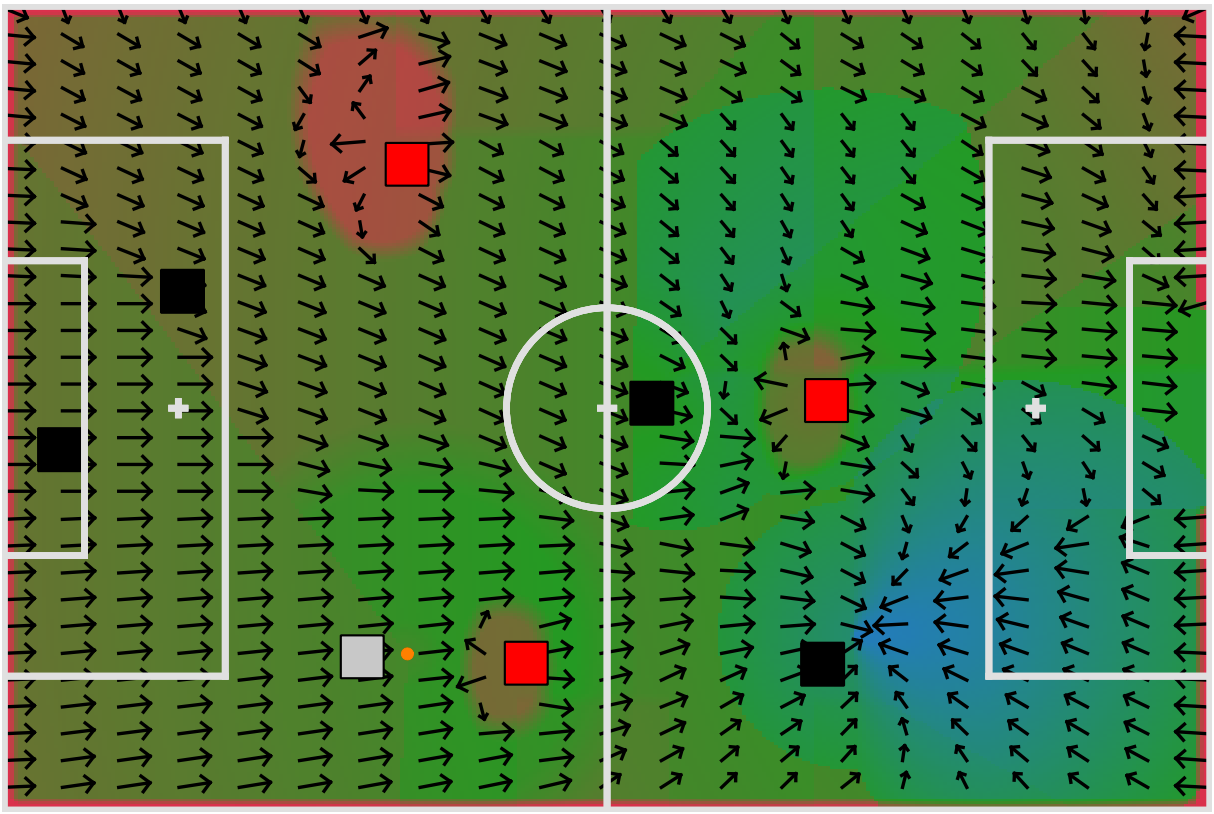


Figure 5.4: The sum of all potential fields from the `FieldRatingProvider`. Black rectangles are teammates, red ones are opponents, the white one is our ball playing robot. The ball is the orange circle. Blue areas are rated good, brown areas are rated bad. The arrows show the potential field direction.

attracting point on the field. Weighted based on their ratings they are summed up to give a value for each point on the field. An example for the sum of the different fields is shown in Fig. 5.4.

5.4 Ball Search

The ball search behavior was implemented for the two set plays *corner kick* and *goal kick*. These two situations contain eight possible positions on the field, where the ball could be: The four corners of the field and the four goal area corners. However, only two of these positions are possible at the same time. The goal is to search these two positions effectively without all players walking around to search for them.

The first step is to see which set play is currently active and for which team it is. With this information, the two possible positions are decided. Depending on the situation being in the own or opponent half, only the players inside these halves will check, if one of those two positions is inside their Voronoi region (see Section 5.1.1). For instance, the defender does not need to check whether one of those two positions is in its region in case of an own corner kick, since both positions for this situation are in the opponent half. In case that both positions are in the Voronoi region of the same player, a midfielder moves to check the corner, which is further away from that player. The others move to predefined base positions until the ball is found or the set play ends. The goalkeeper specifically will not actively move to search for the ball, even

if the positions are in its own half. Thereby, the goalkeeper will not leave the goal undefended. For the same reason, the goalkeeper leaves the execution of goal kicks to the defenders and gives them the space to do so.

Since it can happen that not all players are on the field due to penalties or hardware defects, a minimum number of players that maintain the defense was defined.

In general, the ball search worked as intended for the two set plays it handles. In test games with planned situations as well as at the RoboCup in normal games, only one player for each position was sent while the others moved to their base positions in both the goal kick and corner kick situations. Unfortunately, this walking to the base positions also happens during the kick-in set play, for which a dedicated ball search is not implemented yet. This happened twice at the RoboCup so that the team lost the ball and stopped searching. All players moved to their base positions and only found the ball coincidentally by looking around.

Chapter 6

Technical Challenges

At RoboCup 2022, in addition to the main soccer competition, four technical challenges were held. B-Human won the *7 vs. 7*, *Visual Referee* and *Open Research* (joint first with SPQR Team) challenges and became second in the *Dynamic Ball Handling* challenge and consequently became first in the overall technical challenge ranking. This chapter describes our approaches for the different challenges along with the detailed results.

6.1 7 vs. 7 Challenge

6.1.1 Task

In the 7 vs. 7 challenge, the task was to play matches between teams of seven players per team instead of the usual five. The challenge can be seen as a continuation of the mixed team tournaments from 2017–2019, which were played with six players per team, but with the additional challenge that the players had to cooperate with teammates with different software. B-Human had participated successfully in all three tournaments, laying the software foundations for playing with more than five players. In contrast, the 7 vs. 7 challenge was played by regular teams, i. e. all players on a team ran the same software. Teams could optionally use up to three robots from another team for those games if they did not have enough robots on their own. This was initially planned to be mandatory for participants, but was dropped to reduce efforts in the game preparation phase. However, some rule adjustments intended to protect shared robots were still applied, such as ejection of a player after multiple hardware-related penalties of that robot and a penalty for pushing between teammates.

Figure 6.1: A kick-off scene from the game against the Bembelbots.

	B-Human	Nao Devils	Bembelbots	SPQR Team	GD
B-Human	–	8:0	7:0	9:0	24:0
Nao Devils	0:8	–	2:1	8:0	10:9
Bembelbots	0:7	1:2	–	2:1	3:10
SPQR Team	0:9	0:8	1:2	–	1:19

Table 6.1: Full results of the 7 vs. 7 challenge.

6.1.2 Implementation

Our new behavior architecture handles an increased number of players with little configuration effort. Since the tactics and set plays in the normal strategy contain positions for only up to five players, a special 7 vs. 7 strategy had to be defined. This strategy includes a single tactic as well as one kick-off and penalty kick each for attacking and defending situations.

6.1.3 Results

Only three other teams participated in this challenge. Therefore, a round robin tournament was played. We won all our games with an overall 24:0 goal difference (cf. Table 6.1) and eventually won this challenge. A scene from the game against the Bembelbots is depicted in Fig. 6.1.

6.2 Visual Referee Challenge

6.2.1 Task

The main goal of the Visual Referee Challenge was to explore how referee decisions could be recognized by the robots in a visual way rather than being sent as data packages by the game controller. For that, a robot is placed on the middle point of a soccer field, facing the human referee, who is wearing red gloves. Upon the referee blowing his whistle and showing a gesture

from a predefined set for ten seconds, the robot then gets another ten seconds to mirror and hold this gesture and say the corresponding name of the gesture out loud repeatedly, until its head button is pressed. The team scores points for each correctly identified pose; in case of a tie, the team that detected all poses in sum faster will win. [4, appendix B.2]

6.2.2 Approach

6.2.2.1 Constraints

With the platform to run this challenge being a NAO V6, the hardware is limited and thus are some of the possible approaches. For this challenge, especially camera and CPU have to be taken into account.

Since lighting at an event location can have a great impact on the recorded color values, using the red gloves as marker points for the positions of the wrists would be a risk that should not be underestimated. In addition, since the robot is facing the crowd, it might just as well detect T-shirts or shoes worn by a person in the crowd.

Furthermore, due to the limited computing power of the NAO's CPU, the use of very advanced machine learning approaches or power hungry neural networks might only work with delays or not at all.

The first constraint mentioned prior led to us not training our own machine learning model / neural network. Since we had no knowledge of the lighting conditions and possible backgrounds at the location, any self-trained model would have been likely to be very overfitted. Additionally, we also wanted to circumvent labeling another massive amount of data, since this was already part of this year's Open Research Challenge.

These two factors led to us seeking out a pre-trained model. We evaluated different options, however, it was necessary to take the limited computing power of the NAO robot into account. Therefore, options were limited and in the end we used *MoveNet* [19], a neural network that will detect both joints and face features in images and videos.

6.2.2.2 Keypoint Detection

The module `KeypointsProvider` detects 17 so-called keypoints in the images taken by the upper camera in NAO's head. Most of these keypoints are located on a person's joints, but some mark attributes of a person's face, such as ears, eyes, and the nose. The single pose version of MoveNet is used to detect these keypoints. The network delivers 17 triples of x and y coordinates and a confidence, one for each type of keypoint. There are two versions available that both use RGB images as input: The *Lightning* variant processes an input of 192×192 pixels and runs at around 13.5 Hz on the NAO. The *Thunder* version processes an input of 256×256 pixels and subsequently runs slower (3.5 Hz on the NAO). Both networks cannot be computed by our implementation of efficient network inference *CompiledNN* [17], because they use some network layer types that are not supported. Therefore, we integrated the *ONNX Runtime* [6] into our code base, which was already used as a replacement for CompiledNN on ARM64 Macintosh computers in the past.

Given that the setup of the challenge is rather static, a quadratic region around the center of the upper camera image is used as the input for the MoveNet network (see Fig. 6.2). The region has a fixed size of 384×384 pixels. It is scaled down to the required input size using bilinear interpolation and it is also converted from the YUV422 colorspace to the RGB colorspace. To make sure that the referee is actually visible in this region, the NAO performs a self-localization

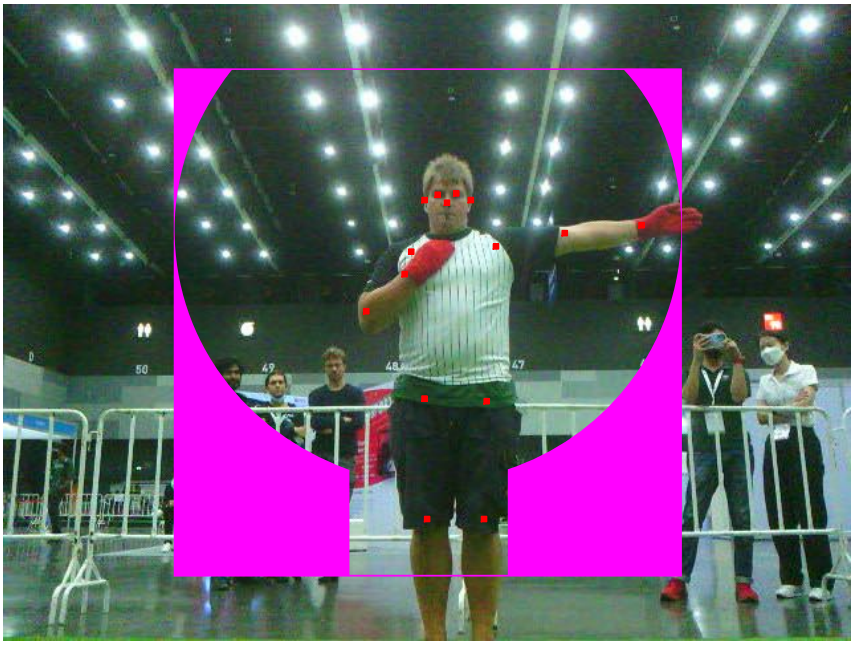


Figure 6.2: Image taken by the upper camera with the quadratic region used as input for the network. Parts of that region are recolored in magenta to mask them out. The red dots mark the keypoints found.

based on the images taken by the lower camera. The position is fixed to the center of the field, but the rotation is estimated to compensate for slight orientation errors that might have been made during the placement of the robot. The robot's head is then controlled to look at a fixed point in 3D space, i. e. at the crossing between the sideline and the halfway line at a height of 1.2 m.

The single pose version of MoveNet is supposed to detect keypoints on a single person. It is not defined what will happen if several people are visible in the image. As there can be spectators present in the background during the execution of the challenge, we tried to increase the likelihood that the actual referee is marked with the keypoints. In the part of the image that is used as input for the network, we defined a mask to exclude regions where the referee could not be visible. The idea is that the network will prefer a person more of which is visible over one, e. g., the legs of which are hidden. The mask consists of a rectangular region representing the area where the legs of the referee could be and a circular region that represents the area the arms can reach. Everything outside these two regions is recolored to magenta. We also tried to color it black, which was less effective, probably because the network simply considered these parts to be in the shadow and still placed keypoints into them.

To allow the keypoint detection to work under different lighting conditions, the weighting table for the automatic exposure of the upper camera was configured to focus on the region of the image where the referee should appear.

6.2.2.3 Gesture Detection

The algorithm will work with a subset of these keypoints to detect the pose. First, the confidence per keypoint is used to check whether a keypoint is valid. In case of an invalid keypoint, the whole arm will be ignored later on. Since MoveNet tends to swap left and right side in the keypoints, each pair of joints is additionally compared by their x-coordinate. If the left keypoint

is further on the right side than the left one, their place in the array of joints is swapped.

To detect poses based on the recognized keypoints, we used a highly parameterized system, which allowed us to adapt thresholds while viewing the live camera of a connected robot or a log, with the option to show debug drawings in the picture in SimRobot. In our approach, a gesture as defined in the rules, is a composition of the position of the two arms. Therefore, we used a predefined set with defined possible gestures of a single arm, which in combination can map all possible gestures from the rules while also maintaining a certain level of scalability and flexibility. By that, we are able to adapt the code quickly if a similar challenge with more or different poses is held again or a 5 vs. 5 rule change will pick up on the idea of visual decision communication by the referee.

The poses recognized for a single arm are defined with the following :

- `down`: The arm is hanging straight down
- `degree45`: The arm is spread away by roughly 45° from the body
- `degree90`: The arm is spread away by roughly 90° from the body
- `up`: The arm is raised over the head of the referee
- `forward`: The arm is straight and in front of the body, respectively pointing towards the middle circle if we assume the referee on the position defined in the rules (middle T-cross)
- `folded`: The arm is folded across the chest
- `none`: The positions of the keypoints do not fulfill the requirements of any of the poses prior
- `wing` and `spreadwing`: these to belong together and describe the arm hold like a bird wings, either folded or on the way to 90° spread.

At the parameter layer, the following values are used:

- Angle of the upper arm to the middle of the body
- Angle of the upper arm to the lower arm (angle point being the elbow)
- Distance of shoulder to wrist

Therefore, the relevant points from the MoveNet output are shoulders, elbows, and wrist. In addition, the hips are also considered for the calculation of the first parameter. The vertical center line of the body that is used for the angle is calculated as a vector from the middle of the shoulders to the middle of the hips.

To compose the overall gestures, a bitmask is used. A whole gesture is represented by a mask in which one bit is set for each of the two arms' individual gestures. The combinations used are shown in Table 6.2.

However, since the keypoint detection is quite noisy, individual gestures might be detected wrongly. Therefore, the detections are buffered for a while. Only if the predominant gesture was detected sufficiently often, that gesture will be accepted and written into the representation `RefereePercept`.

		Right arm								
		down	degree45	degree90	up	forward	folded	wing	spreadwing	none
Left arm	down	-	cornerKickRed	kickInRed	goalKickRed	-	-	-	-	-
	degree45	cornerKickBlue	-	-	-	*goalRed*	*pushingFreeKickRed*	-	-	*goalRed*
	degree90	kickInBlue	-	fulltime	-	goalRed	pushingFreeKickRed	-	-	*goalRed*
	up	goalKickBlue	-	-	-	*goalRed*	*pushingFreeKickRed*	-	-	*goalRed*
	forward	-	*goalBlue*	goalBlue	*goalBlue*	-	-	-	-	-
	folded	-	*pushingFreeKickBlue*	pushingFreeKickBlue	*pushingFreeKickBlue*	-	fulltime	-	-	-
	wing	-	-	-	-	-	-	fulltime	-	-
	spreadwing	-	-	-	-	-	-	-	fulltime	-
	none	-	*goalBlue*	*goalBlue*	*goalBlue*	-	-	-	-	-

Table 6.2: All combinations of single arm poses and the corresponding referee gesture. Combinations that are not defined in the rules but nevertheless recognized to compensate for mistakes made in the keypoint calculation are marked in *italic*.

6.2.2.4 Behavior

The behavior is written using the default B-Human Option and Skill architecture. The robot starts in the initial state. When the game state switches into *Playing* (for the challenge this was done by pressing the chest button to penalize and unpenalize the robot), the state will change into `waitForWhistle`. To make it easily noticeable whether the whistle was heard, the robot will say “Let’s Go” upon recognizing a valid whistle sound. After that, the state will switch to `detectGesture`. However, to give the referee time to move his or her arms into their final position, the robot will wait for one second. In the meantime, the buffer in the module `RefereeGestureDetection` has time to fill. The time limit is therefore implemented in two ways: The hard-coded wait time before gesture stored in the `RefereePercept` is read at all and the time the buffer needs to fill with the same gesture so it can be accepted. After the initial second has passed and a gesture was detected, i. e. the one stored in the `RefereePercept` is not `none` anymore, the behavior will switch to a specific state for that gesture. In that state, the movement to mirror the referee is executed and the name of the gesture is announced out loud.

The gestures of the referee defined in the rules, which are to be recognized and mirrored by the robot, consist essentially of five core movements, whereby the arm hanging next to the body is regarded as a default movement and accordingly is not considered here. These five core movements are

- Arm extended 45° downwards to the side
- Arm extended 45° upwards to the side
- Arm extended 90° to the side (horizontally)
- Arm extended 45° to the front
- Arm horizontal, with the wrist in front of the chest

In Fig. 6.3, it is shown how the poses look when executed on a NAO robot.

The referee’s gestures are just arm configurations, one of them a moving one, the others static positions. The moving gesture is waving both arms from being fully extended horizontally to holding the hands in front of the chest while the arms are still kept horizontally. Especially with this moving gesture, care must be taken when developing the movement of the robot that it is not performed too quickly or jerkily to prevent the robot from moving too much or even tipping over. If too much movement were to occur, the recognition of subsequent gestures would be disrupted. In addition, there is the possibility that the arms of the robot get stuck during the execution of the movement, which also limits a mirroring of following gestures. To avert these risks, the gestures of the robot were first tested in the simulator and then their speed was

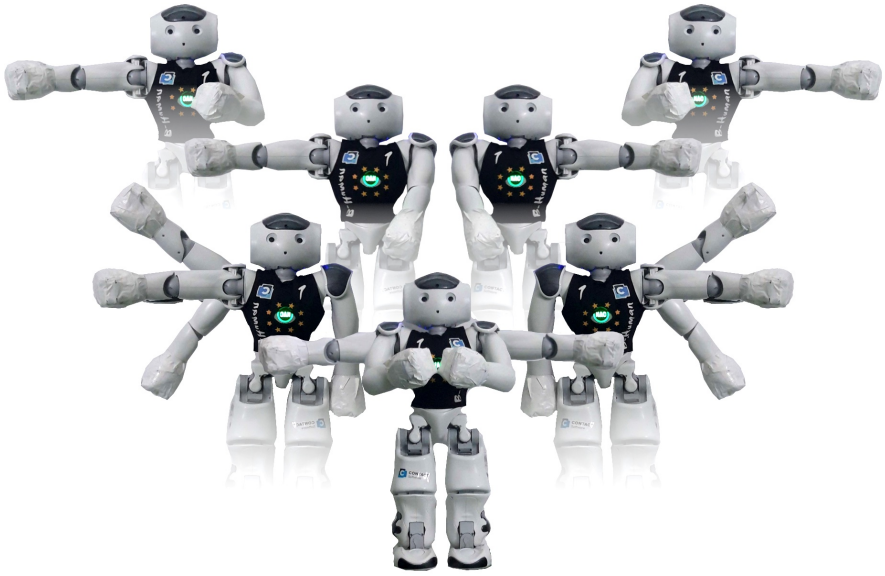


Figure 6.3: All possible referee poses shown by a NAO robot

adjusted. To prevent the arms from getting caught on the robot's body, some gestures are initially performed as intermediate states from which the target gesture can be reached without getting stuck. After completion of a round, the movement is performed backwards to also avoid getting stuck.

6.2.3 Evaluation

After extensive tuning of the parameters, the result of the neural network and detection were satisfying and correct in most cases. In the competition of RoboCup 2022, B-Human was able to detect all five gestures correctly. While risky, since we had no prior knowledge how other teams were performing in the challenge, we made a conscious decision to slow down the detection process in order to make a decision that is based on more frames in the actual position rather than the referee moving his or her arms into it. This might have led to recognizing an arm spread by 90° as spread by 45°. In the end, it was actually this decision that led to the win, since another team was faster, but only got 4 out of the 5 gestures correct. The results of the challenge at the RoboCup 2022 event are shown in Table 6.3.

Notable milestones concerning the accuracy of the algorithm came mainly from the following changes:

- Using the Thunder version of MoveNet instead of the Lightning version, which is a lot slower but produces a more reliable output. After the competition, the log file from the actual challenge execution was replayed in SimRobot with the Lightning net. While not entirely realistic, because of the lower frame rate of the log, the results with that version of the network would most likely not have been sufficient to recognize the gestures. In many frames, the small stripes printed on the referee's T-shirt were mistaken for an arm, leading to very undesirable keypoints.
- Using a mask on parts of the image to black out every part that has no or a very low chance of containing body parts of the referee. The reason for this is the circumvention of the net detecting other persons than the referee, which sometimes happened in tests

Teams	Points	Rank
B-Human	25	1
NomadZ	22.5	2
RedBackBots	17.5	3
Bembelbots	10	4
rUNSWift	5	5

Table 6.3: Points and ranking of the Visual Referee Challenge at RoboCup 2022.

prior to this. The latter became less with the prior change, but to our knowledge entirely stopped with the introduction of the mask.

- Accepting the gesture for “goal” if one arm is recognized as `degree90` but the other one is `none`. The `forward` position of the arm was by far the hardest one to recognize, since there was the most variation in how it was done by different “referees”.

While the results in the challenge were satisfying, this version of the gesture detection is not suitable for real soccer in its current state. Due to the high computational requirements of the thunder version of MoveNet, the software cannot be used while also playing soccer; 3.5 Hz is not nearly enough, and it might run even slower with all the other modules used in soccer play. We suppose that it might be an option to have the execution of the module disabled for the most part and only activated if the whistle is detected. However, the robots would then first have to find the referee on the field, before they can detect any gestures.

6.3 Dynamic Ball Handling Challenge

6.3.1 Task

The Dynamic Ball Handling Challenge [4, appendix B.3] extended the 2021 Passing Challenge. The major new challenge was the introduction of moving opponents in a more realistic game situation.

Six robots are on the field, three each in the attacking and defending team. Player 1 of the defending team is the goalkeeper, who stands in the middle between the two goalposts. Player 2 stands on the front line of the penalty area and Player 3 in the center circle. Player 1 of the attacking team is the goalkeeper and stands on the front line of the goal area. Player 2 and Player 3 are standing on the center line, one on the left side and the other one on the right side of the center circle. The ball is placed on the penalty mark of the attacking team. The setup of the challenge is shown in Fig. 6.4.

The goal of the attacking robots is to score a goal after two successful passes without letting the defenders touch the ball. The defenders are running one of multiple different behaviors that could be contributed by teams in advance, which Nao Devils and we did.

6.3.2 Approach

6.3.2.1 Attackers

The basic idea of the implementation is that the first player decides which of the two remaining players to pass to based on the pass rating. After accepting the first pass, the receiver should

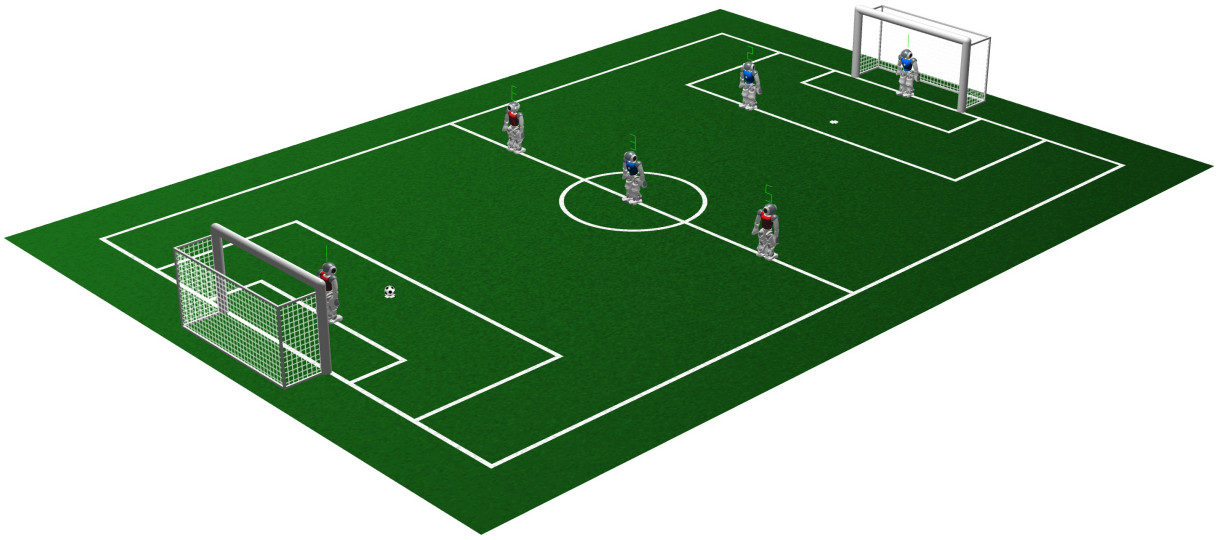


Figure 6.4: Initial positions of the attacking robots (red) and defending robots (blue) for the Dynamic Ball Handling Challenge

then pass to the player that has not yet made contact with the ball. The last player ideally ends the challenge by successfully scoring a goal, but further passes between the second and third player can happen if the goal seems blocked. Therefore, the behavior differs fundamentally between player 1 and player 2 / 3.

Player 1

The first player starts at the front goal area line, the ball on the penalty spot. The first player passes either to the second or the third player depending on which pass rating is higher. The pass rating is given by the `PassEvaluation` representation as described in Section 5.3.2. After passing to the second or the third player, the ball will be out of range and the player will walk to the center point, to support the attack.

Player 2 and 3

The second and the third player have the same behavior and are referred to as *forwards* here (in contrast to player 1). As long as they are not currently playing the ball, they try to get into a good position to receive a pass. This is comparable to the behavior of the forward described in Section 5.1.1.

Both players estimate whether the other forward already had the ball based the distance between its communicated position and the ball. If the other forward did not have the ball yet, it will be passed the ball. Once all three players had the ball after this second pass, the forward at the ball decides between a shot at the goal and a pass to the other forward while that one continues to position itself for a pass. The decision is based on expected goals (cf. Section 5.3.1) and the pass rating (cf. Section 5.3.2), similar to the normal ball playing behavior (cf. Section 5.1.3).

Teams	Points	Rank
HTWK Robots	25	1
B-Human	20.9	2
Nao Devils	12	3
RoboEireann	5	4

Table 6.4: Points and ranking of the Dynamic Ball Handling Challenge at RoboCup 2022.

6.3.2.2 Defenders

As different restrictions apply to the individual defenders, we differentiate the behavior of them. Only the second and third player have in common that they go directly towards the ball and try to touch it if they are less than 1.5 m away.

Player 1

The first player is the goalkeeper and is not allowed to leave the ground line. Its task is to intercept goal shots. To prepare for that, it positions itself on the side of the goal where the ball is but with a tendency toward the center if the ball is far away.

Player 2

The second player has to defend the own half. If the ball is in the other half, it searches for the attacker next to it and marks him so that it can engage in a duel if a pass to the marked attacker is played. To consistently mark the right attacker, we check whether the new position is within a certain radius around the old one. This radius increases with the time in which no suitable attacker was found to adapt for moving opponents and allow to recover from false positive detected attackers. If the ball is within the own half, the second defender has to go towards it. We do so not in direct manner but rather with a slight arc to intercept the line between the ball and the last marked attacker. The further away the defender is from this line the more it will move to intercept it, when it is near the line it will move more towards the ball.

Player 3

The third defender has to stay in the opponent half. If the ball is in that half, it moves towards it in the same arc-like manner as described above. This is done to close the line between the ball and a fixed spot on the center line at the opposite half where the second defender is located. To encourage a pass to the opponent that is marked by the second defender. If the ball is in the own half, the third player is not allowed to go there but it helps the other ones by standing near the center line and looking at the ball. If the ball is near the center line, it will stand near it to be able to quickly reach it in case it rolls back in the opponent's half. If the ball is far into the own half, it will position itself more in the center of the field to be able to remain a good look at the ball in case of a long pass to the other side. This also allows us to use no hysteresis as we have a quite smooth transition at the point where the ball crosses the center line.

B-Human	3 Defenders	2 Defenders	1 Defender
1st try	2:59	3:39	3:31
2nd try	3:21		
3rd try	3:46		

Table 6.5: The time when the Dynamic Ball Handling Challenge stopped.

6.3.3 Results

As shown in Table 6.4, we were able to place second in this challenge, with a total of 20.9 points. In the first three attempts, no team managed to score a goal against the 3 defenders. We managed to score a goal, which did not count, because an attacker left the field. Next, one less defender was lined up and each team had one more try. HTWK was the only team that managed to score a goal. Last, there was only one goalkeeper left as a defender. In this attempt our robot managed to shoot at the goal, which was stopped by the defending goalkeeper. The times are shown in Table 6.5.

6.4 Video Analysis Challenge

6.4.1 Task

This Open Research Challenge is about developing software capable of automatically analyzing video recordings of SPL games in order to facilitate evaluating the progress of the league over time. More specifically, the two main goals of this challenge include using a GoPro recording of a SPL game to determine the extrinsic camera parameters as well as detecting all relevant objects in the video with the aim of calculating statistics similar to human soccer such as each team's ball possession.

6.4.2 Approach

6.4.2.1 Object Detection

To enable solving the task of locating objects in the video, members from seven teams labeled images from recordings of RoboCup 2019 soccer games. Each team labeled the ball and all players including their jersey colors and their jersey numbers in 5000 images. In our contribution to this challenge, a YOLOv5 [10] network was trained with an input resolution of 1920x1080 with bounding boxes of the ball and the players including their jersey colors (see Fig. 6.5). Only 682 images were actually used at a 70/20/10 split. The inference takes approximately 55 ms per image on an M1 Apple Neural Engine.

6.4.2.2 Camera Calibration

To transform the detections from image coordinates into field coordinates, an intrinsic and an extrinsic camera calibration are needed. To obtain the extrinsic camera calibration, we adapted the code provided by the team Berlin United [16] that offered the ability to optimize the extrinsic camera calibration using a hard-coded intrinsic calibration. We added the optimization of the intrinsic calibration together with the extrinsic one. The calibration works by firstly detecting field lines in a background image that is computed from many images of the video to eliminate

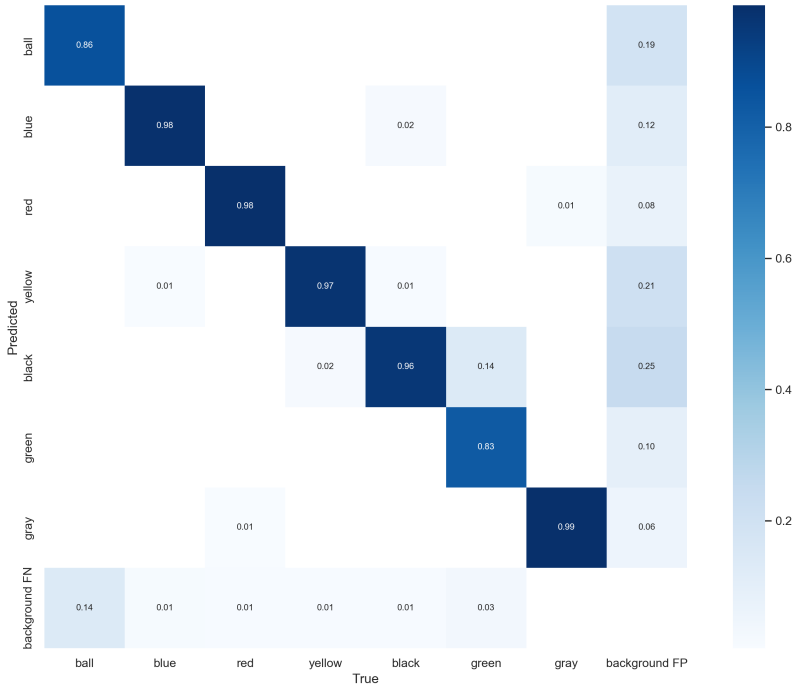


Figure 6.5: Confusion matrix of ball and player colors in the test set.

moving objects from the field in order to get a clear view at the field lines. Then, the camera pose is calculated by determining the transformation matrix which maps lines in the image to lines on the field, taking into account the distortion of the GoPros used for recording at the RoboCup. To achieve a satisfying result, an adequate initial guess of the camera pose and the intrinsic calibration has to be provided. The calibration is performed before the first playback of a game video and then stored for later uses.

We also added an inverse transformation from field coordinates back into the image which allows drawing on the video for visualizing statistics.

6.4.2.3 World Model

The bounding boxes of the players are tracked using a centroid multi object tracker [5] ignoring the colors in YOLOv5's non-maximum suppression. For each track, the recent history of colors associated with it is maintained and the most frequent color is assigned as the player's color. False ball detections are filtered out based on an assumed maximum speed of the ball. The centers of the bounding boxes are projected into the world based on an assumed height above ground, i. e. 5 cm for the ball and 26 cm for the players.

6.4.2.4 Statistics

Game Controller Logs

The application reads the GameController's log file of the game and plays back its events while the video is running. For each team it displays statistics about game events such as the number of goals scored (per minute) and various accumulated penalties/set-pieces over the course of the game. Additionally, the GameController logs are used to determine the current game state (Set, Playing, etc.), which improves the results of other statistics e. g. by limiting their calculation

Figure 6.6: The application showing the video of a half not in the training set with the positions of the ball and the players. The orange line indicates the ball possession of the player that just kicked the ball while it is still rolling. The statistics determined from the GameController log file and from the video are shown next to the video.

to the Playing state.

The camera setup at the RoboCup starts recording automatically as soon as the game state changes from Initial to Set with a slight delay. Based on labeled images we assumed this offset to amount to 1.5 s, which improves the alignment of the video with the events from the GameController log.

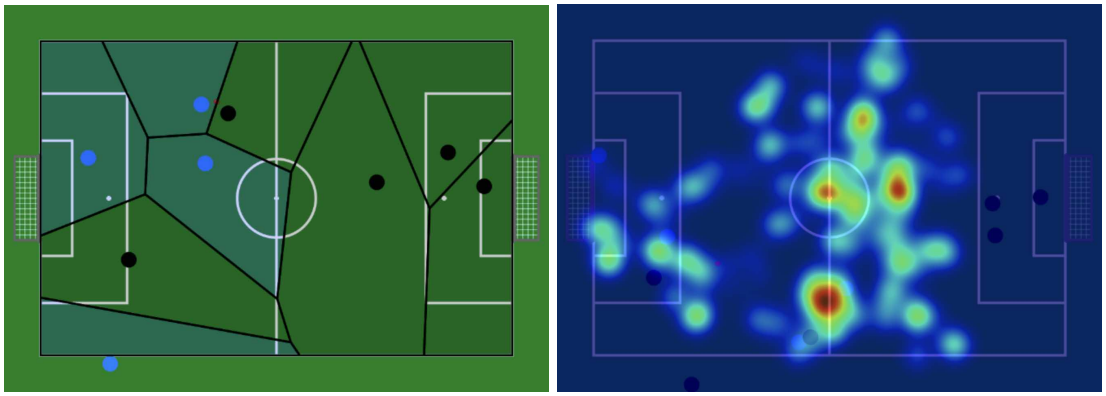
Ball Possession

The calculation of the ball possession is implemented using a simple state machine consisting of a “None” state (no team is in possession of the ball) and one state for each team respectively, meaning this team currently controls the ball. Since just being close to the ball for a brief moment doesn’t necessarily imply control of the ball, a change of possession can only occur when the ball is not moving. If the state currently is “None” (e. g. in the beginning), the possession changes to one of the teams as soon as one of their players gets closer than 40 cm to the ball. The possession can change from one team to the other once no player of the team in possession is closer than 60 cm to the ball, there is a player from the other team less than 40 cm to the ball and the ball has stopped moving for at least 0.6 s. The possession changes from one team to “None” when no player of the team in possession is closer than 60 cm to the ball, the ball has stopped for at least 1 s and the aforementioned criteria for a change of possession are not met.

In addition, it is computed how far the ball traveled while being in possession of a team.

Controlled Area

This statistic measures how much space is controlled by each team during the game on average. A position on the field is assumed to be controlled by a team if it is closer to a player of that team than to any player of the other team. The sum of all these positions is the



(a) 2D field view with controlled areas (b) Ball position heat map view

Figure 6.7: 2D views for displaying statistics.

area controlled by the team. These areas are computed by constructing the Voronoi diagram from the positions of all players on the field (see Fig. 6.7a).

Falls

Detecting how often robots fall to the ground during the game can be an important indicator regarding the quality of the game. The detection of falls is implemented by making use of the bounding box: As mentioned earlier, the centers of the player's bounding boxes are projected into the world based on their assumed height above ground of 26 cm. If the projection of a player's ground position back into the image is outside its bounding box, the player is assumed to be fallen. However, the fall detection is quite simple. It fails if YOLOv5's bounding box does not contain the whole robot. This often happens if a robot is partially hidden by another robot or a referee.

Ball Position

There are several statistics being calculated using mainly the position of the ball.

- The "distance to own goal" statistic calculates the average distance between ball and the own goal for each team.
- The heat map is a visual representation of the positions of the ball over time, created by applying a Gaussian filter on the accumulated ball positions on the field (see Fig. 6.7b).
- The "ball isolation time" indicates how many seconds no player has been closer than 1 m to the ball in total. This statistics is special, because it is not directly linked to one of the teams. Therefore, it will not be shown in the user interface, but still added to output file for both teams.

Distance Walked

This statistic determines the total distance that all players of each team have moved. For the movement of a player to be considered in this statistic, a minimum distance of 10 cm from the previous position is required in order to filter out noise in the detections.

Teams	Points	Rank
B-Human	25	1
SPQR Team	25	1
NomadZ	21.9	3
Nao Devils	18.1	4
RoboEireann	13.5	5
R-ZWEI KICKERS	5	6

Table 6.6: The points and the ranking of the Video Analysis Challenge at the RoboCup 2022.

6.4.3 Results

In this challenge, the score was determined by a voting of all participating SPL teams after the presentation of the results at the RoboCup 2022. The teams were asked to evaluate the outcome based on the following criteria [4, section B.4]:

- Achievement of the long/short-term goal
- execution time (and hardware requirements)
- metrics (accuracy/precision/recall)
- technical strength and
- novelty.

We received a total of 34 votes from other teams and thus achieved first place, joined with the SPQR team. As a consequence, we were awarded 25 points for this challenge (see Table 6.6).

Bibliography

[1] Gabriel Anzer and Pascal Bauer. Expected passes. *Data Mining and Knowledge Discovery*, 36(1):295–317, 2022.

[2] B-Human. B-Human wiki. `https://wiki.b-human.de/coderelease2022/`, 2022.

[3] Joydeep Biswas, Juan Pablo Mendoza, Danny Zhu, Benjamin Choi, Steven Klee, and Manuela Veloso. Opponent-driven planning and execution for pass, attack, and defense in a multi-robot soccer team. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 493–500, 2014.

[4] RoboCup Technical Committee. RoboCup Standard Platform League (NAO) rule book. `https://spl.robocup.org/wp-content/uploads/SPL-Rules-2022.pdf`, 2022.

[5] Aditya M. Deshpande. Multi-object trackers in Python (v1.0.0). Zenodo, 2020. `https://doi.org/10.5281/zenodo.3951169`.

[6] ONNX Runtime developers. ONNX Runtime. `https://onnxruntime.ai/`, 2021. Version: 1.10.0.

[7] Uwe Dick, Daniel Link, and Ulf Brefeld. Who can receive the pass? A computational model for quantifying availability in soccer. *Data Mining and Knowledge Discovery*, 36(3):987–1014, 2022.

[8] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[9] Bernhard Hengst. rUNSWift Walk2014 report. Technical report, 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.

[10] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Jiacong Fang, imyhxy, Kalen Michael, Lorna, Abhiram V, Diego Montes, Jebastin Nadar, Laughing, tkianai, yxNONG, Piotr Skalski, Zhiqiang Wang, Adam Hogan, Cristi Fati, Lorenzo Mammana, AlexWang1900, Deep Patel, Ding Yiwei, Felix You, Jan Hajek, Laurentiu Diaconu, and Mai Thanh Minh. ultralytics/yolov5: v6.1 – TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference. https://doi.org/10.5281/zenodo.6222936, February 2022.

[11] Jo Lienhoop. Dynamisches Passspiel im Roboterfußball. Bachelor thesis, Universität Bremen, to appear.

[12] Juan Pablo Mendoza, Joydeep Biswas, Philip Cooksey, Richard Wang, Steven Klee, Danny Zhu, and Manuela Veloso. Selectively reactive coordination for a team of robot soccer champions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[13] Philip Reichenberg and Thomas Röfer. Step adjustment for a robust humanoid walk. In Rachid Alami, Joydeep Biswas, Maya Cakmak, and Oliver Obst, editors, *RoboCup 2021: Robot World Cup XXIV*, volume 13132 of *Lecture Notes in Artificial Intelligence*, pages 28–39. Springer, 2022.

[14] Thomas Röfer, Tim Laue, Andreas Baude, Jan Blumenkamp, Gerrit Felsch, Jan Fiedler, Arne Hasselbring, Tim Haß, Jan Oppermann, Philip Reichenberg, Nicole Schrader, and Dennis Weiß. B-Human team report and code release 2019. https://github.com/bhuman/BHumanCodeRelease/raw/master/CodeRelease2019.pdf, 2019.

[15] Thomas Röfer, Tim Laue, Arne Hasselbring, Jo Lienhoop, Yannik Meinken, and Philip Reichenberg. B-Human 2022 – more team play with less communication. In *RoboCup 2022: Robot World Cup XXV*, Lecture Notes in Artificial Intelligence. Springer, submitted.

[16] Benjamin Schlotter. Analyse von RoboCup Spielen – Erkennen und Lokalisieren von Nao Robotern. Technical report, Humboldt-Universität zu Berlin, 2020. https://www.naoteamhumboldt.de/wp-content/papercite-data/pdf/2020_studienarbeit_schlotter.pdf.

[17] Felix Thielke and Arne Hasselbring. A JIT compiler for neural network inference. In Stephan Chalup, Tim Niemueller, Jackrit Suthakorn, and Mary-Anne Williams, editors, *RoboCup 2019: Robot World Cup XXIII*, volume 11531 of *Lecture Notes in Artificial Intelligence*, pages 448–456. Springer, 2019.

[18] Rico Tilgner, Thomas Reinhardt, Stefan Seering, Tobias Kalbitz, Samuel Eckermann, Michael Wünsch, Florian Mewes, Tobias Jagla, Stephan Bischoff, Carolin Gümpel, Marvin Jenkel, Andreas Kluge, Tobias Wieprich, and Felix Loos. Nao-Team HTWK team research report. Technical report, Hochschule für Technik, Wirtschaft und Kultur Leipzig, 2020.

[19] Ronny Votel and Na Li. Next-generation pose detection with MoveNet and TensorFlow.js. https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html, 2021.