Team Description for RoboCup 2017

Dipl.-Inf. Andreas Fürtig[1] (Teamleader)
Jonathan Cyriax Brast[2], Sina Ditzel[2], Hans-Joachim Hammer[2], Timm Hess[2], Kyle Rinfreschi[2], Jens-Michael Siegl[2], Stefanie Steiner[2], Felix Weiglhofer[2], and Philipp Wörner[2]

[1] Electronic Design Methodology Group
Institute for Computer Science, Goethe Universität Frankfurt am Main, Germany
Email: fuertig@em.cs.uni-frankfurt.de
[2] Institute for Computer Science, Goethe Universität Frankfurt am Main, Germany
Email: team@bembelbots.de
Homepage: www.bembelbots.de
Github: github.com/Bembelbots

1 Introduction

The RoboCup team *Bembelbots* was founded in 2009 at the Goethe University Frankfurt (Main), Germany as a group fully organized by students. As there is no robotics group at the university, the team should help students to increase their experience in robotics, as well as programming skills in addition to the theoretical orientation of the computer science degree program of the university. The team does not have any constant financial resources, so a lot of resources are used to gain funding to afford robot upgrades and additional hardware.

Currently 9 undergraduate students as well as one PHD student working on the implementation of the framework for playing soccer. Team Leader is Andreas Fürtig, a PhD student from the Electronic Design Methodology (EM) group at the Institute of Computer Science at Goethe University. The team owns six Nao

H25 v5 robots, as well as six old H21 v4 versions which are mainly used for demo purposes and public relations.

Since 2012 the team *Bembelbots* organize the FIAS BembelCup in line with the "Night of Science" event, a public event showing different aspects of science, technology, engineering and mathematic study paths offered at the Goethe University in Frankfurt. This small tournament of four competing teams is one of the final tests for the RoboCup, as it is temporally located nearby.

Fig. 1. Team *Bembelbots* at the RoboCup German Open 2017 in Magdeburg, Germany.

2 JRLSoccer System Architecture

As the main goal of the team is to teach students in the field of robotics as well as working on a complex software architecture, the team focused right from the beginning on building a full framework from scratch rather than using existing architectures. Our framework is split up into four main parts: a *backend* implementation, providing data for the *frontend*, which represents the functionality of the robots software. Backend and frontend communicate through a BOOST shared memory model, giving us the possibility to be platform independent. A standalone *monitor* binary provides system informations of robot health, as well as game configurations during competitions through TCP/IP. For debugging purposes we use several tools written in *Python* combined in our debug tool *BembelDbug*. Figure 2 illustrates the overall structure of our framework.

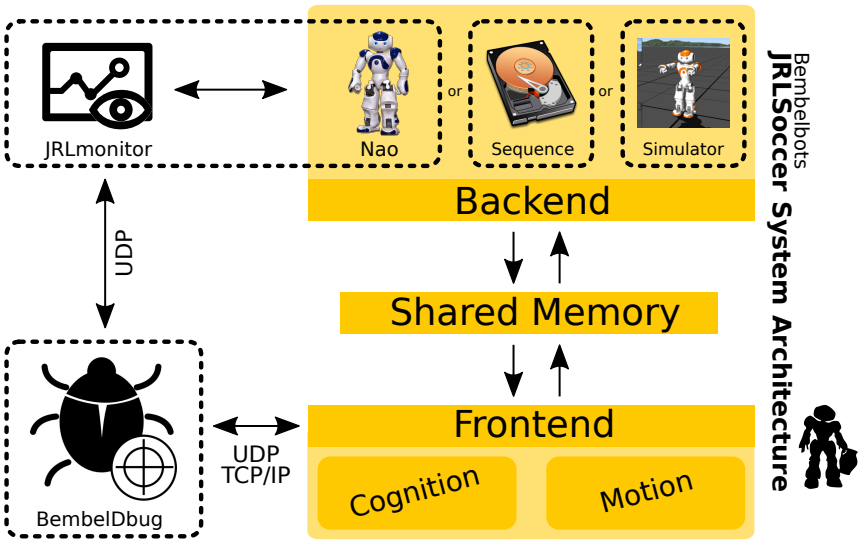

Fig. 2. Overview of the JRLSoccer Framework.

2.1 Backend

The backend can be replaced to provide data on different systems like the robot itself or to provide simulated data and sequences for offline testing of our framework:

backend-naoqi This backend is used on the robot and communicates with the hardware directly using the DCM provided by the NaoQi[1] (version 2.1.4) for sensor and actuator data. Images are captured directly from the camera using a Linux V4L2 implementation.

backend-sequence-loader Most of our tests are based on a given testset of sensor data and images combined for different game situations. These sequences can be loaded by a sequence loader application which communicates with our frontend. In this way we are able to test our frontend especially for memory leaks and communication purposes.

backend-simspark To simulate our robot and to test our behavior as well as motion based functionality we use a direct connection to the simspark simulator, providing sensor data and perceptions for the robot.

2.2 Frontend

The frontend *jsfrontend* is where all of our game logic is implemented. As a standalone program this allows us to test the frontend separately from a robot,

[1] Aldebaran Robotics Documentation, `http://doc.aldebaran.com/2-1/naoqi/`

which is very helpful to reduce the possibility of memory leaks. Software Management is organized using GIT so that the whole framework can be tested using a Jenkins[2] continuous integration process after every code change.

Our framework is organized as a blackboard architecture, giving every submodule the possibility to share data, access other informations and increase debugability. It is also possible to access and manipulate every blackboard directly using our debug tools.

Mainly the frontend is split up in two major parts. The *cognition* thread is synced with the image gathering process, calculating the localization, world beliefs and the desired behavior decisions. The *motion* thread itself is limited by the sensor acquisition frequency and computes the movements of the robot (see Section 5 for more details).

Network communication with other robots as well as the whistle detection (see Section 6) are also placed in separate threads. These threads will be started and stopped on demand. An overview of our frontend structure can be found in Figure 3.

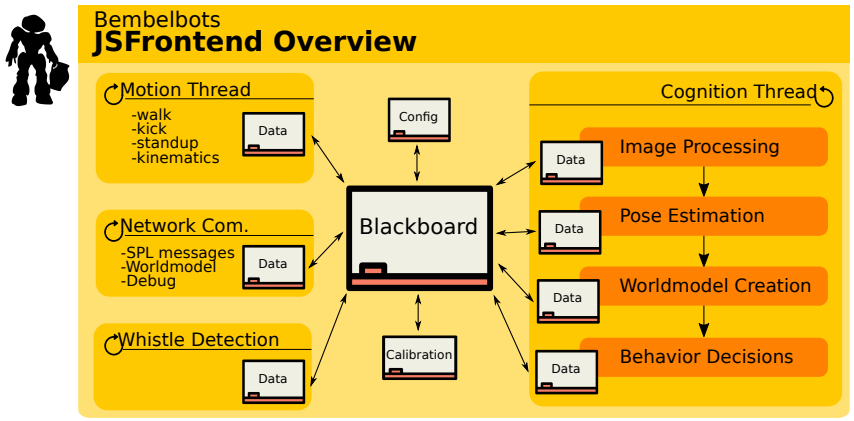

Fig. 3. Overview of the main parts of the JSfrontend architecture with its main threads.

3 Object Detection Pipeline

Finally, we decided to partially replace our statistics and color based vision [1] by the colorless vision module from HTWK Leipzig[3] and created a patch to compile it with the standard Aldebaran Robotics crosscompile toolchain. As the module only detects objects in the image plane, we made it compatible with our localization algorithms that were already in place.

[2] Jenkins open source automation server: `www.jenkins.io/`
[3] Nao-Team HTWK, HTWKVision: `github.com/NaoHTWK/HTWKVision`

3.1 Crossing Detection

We use a line based approach to detect T or L shaped crossings. The line segments recieved from the vision pipeline are compared to each other to see if they intersect at 90 degrees if projected from the image into the robots coordinate system. These are then considered as possible candidates for T or L crossings. The candidate line segments are extended as far as possible in order to compare the position of intersection to the width of the line. If distance of the intersection point is longer than half of the line width it is considered to be a T crossing, otherwise it is considered an L crossing. The intersection point closest to the end of the line is used for this purpose.

For the localisation of the robot it is important to estimate the orientation of the crossings. So each crossing is marked with a fixed angle [2] to calculate one unique position for each type of crossing on the field.

3.2 Ball Detection

The ball detection pipeline comprises of two main components, namely a region of interest (ROI) prediction process and a convolutional neural network (CNN) classifier. The ROI prediction process applies three filter stages to each image

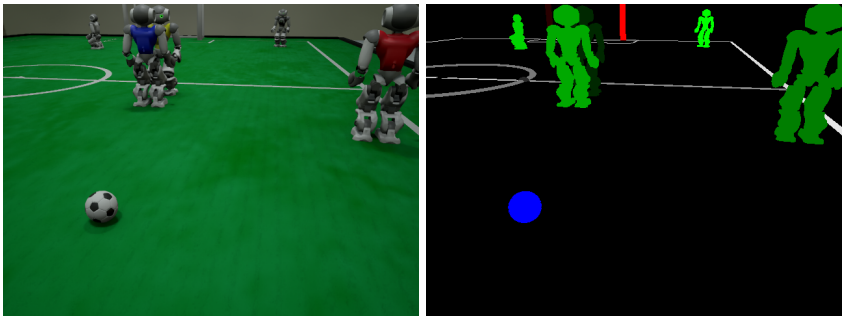

Fig. 4. Example of synthetic data with semantic ground truth annotation. The figure shows a single example of a simulated playing situation (**left**) together with its automatically generated pixel wise ground truth annotation (**right**).

while also making use of features already provided by the HTWK vision framework [8], such as line segments (green-to-white gradients) unassigned to line objects, field borders, and green- and white- color class estimations, to prevent redundant feature computation. In the first filter stage scanline points that could not be assigned to a line segment are clustered to extract the most interesting regions from the image, reducing the effective image region to be considered in the further image processing. In the second filter stage black colored blobs are segmented from each initial guess region as they are a main feature of the ball that is currently used in the SPL. The third filter stage performs a scanline

procedure using the image's Cr-color information to detect the ball border and close it in a tight bounding box. This stage also performs several sanity checks on aspect ratio and size of the bounding boxes before subsequently passing them to the CNN.

The CNN classifier discriminates into two classes, namely "ball" and "no ball" and has been implemented using the Tiny-DNN [6] framework. It is composed of two convolution-blocks (convolution, ReLU-activation, max-pooling), followed by two fully connected layers, which are also implemented as convolutions [7]. Training of the CNN has been performed using a combination of real world data and synthetic data, shown in Figure 3.2, generated using the Bembelbots UERoboCup Simulator which is our UnrealEngine4 based synthetic training set generator, that has been open-sourced recently[4] and is available for the RoboCup community trying to support all teams with "unlimited" amounts of annotated training data boosting the usage of data driven approaches throughout our league. A first look at the benefits and challenges of using synthetic data in training neural networks for classification tasks in the SPL RoboCup environment has been researched by our team and will be published soon.

4 Robot Localization

4.1 Particlefilter

To estimate the pose of the robot using visual perceived landmarks and odometry we are using a Monte Carlo Localization (MCL) algorithm based on a particle filter [3]. One of the main reasons for the usage of the MCL algorithm is the possibility to solve global localization problems, as well as the integration of all observed landmarks without further processing. The filter represents the belief of robot's pose with a set of samples. Each sample (particle) is a possible pose of the robot. In the initial state, after a penalty, and after manual placement the particle filter is initialized by fixed pre-determined distributions according to the situation.

In our localization approach we use a combination of motion controls and sensor data from gyroscopes for the motion model update and weight the particles by matching the visual perceived lines, crossings interceptions and poles with the field landmarks. In each measurement update the filter compares landmarks by their distance and angle and matches them using the maximum likelihood method.

We chose the systematic resampling algorithm [9], which yields a reliable localization result for low numbers of particles (less then 100). To determine the robot's pose from the particle distribution we decided to calculate the particle closest to the mean of all particles. This offers the best compromise between accuracy and runtime performance compared to choosing the particle with the highest weight or applying a cluster algorithm.

[4] Bembelbots UERoboCup synthetic training set generator: `github.com/Bembelbots/UERoboCupUERoboCup`

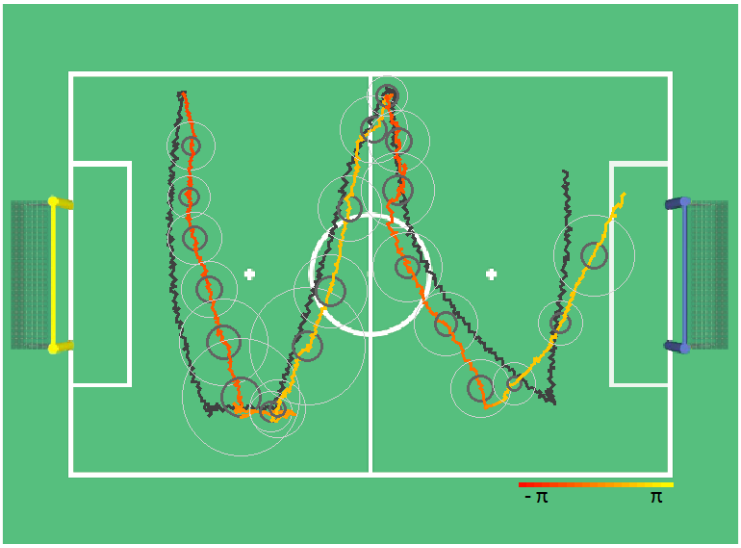

Fig. 5. Evaluation of the particle filter: The black line represents the ground truth position of the robot, the yellow to red line shows the estimated position and the color decodes the estimated orientation of the robot. The circles represent the distribution of the particles: Dark grey circles display the mean distance of the particles from the center, light grey circles indicate the particle position with the highest distance.

To evaluate the filter and the different methods of resampling and position determination we developed a system (a ceiling mounted camera tracking LEDs attached to Nao's head) which we use to generate ground truth sequences. The evaluation of a path over the playingfield is illustrated in Figure 5.

4.2 Worldmodel

Each robot broadcasts its estimated pose, last known ball location and additional data via UDP in the standard SPL message format (see Figure 6). All data received from teammates as well as the robot's own location and ball position are merged into a worldmodel, based on which the behavior decides its own actions as well as complex team strategy.

5 Bodycontrol

The new Bodycontrol and Motion Framework[5] has a modular structure, in which all motions are separate modules, some of which run exclusively such as whole body motions (i.e. stand up moves) and some run in parallel such as the walk and the head looking towards a position.

Apart from motions the bodycontrol includes additional functionalities like sensor data acquisition or including center of mass (CoM) calculations, cam-

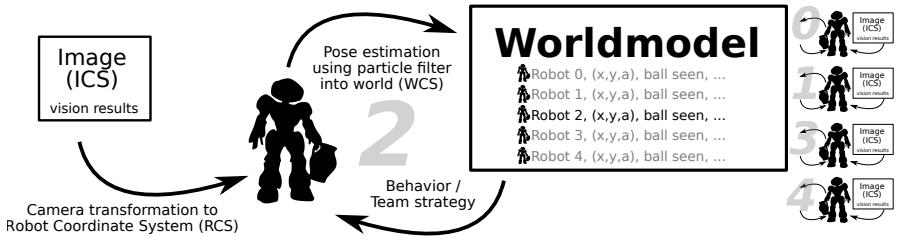

Fig. 6. Creation of the worldmodel from own pose position process and the by receiving broadcasts from teammates.

era transformations, inertial memory unit (IMU) filters and setting LEDs for debugging purposes, that follow the same modular structure.

Fig. 7 shows the structure of the framework. For each motion thread cycle all active modules modify the data on a blackboard upon being called in sequence by a control shell (Runner), which also handles activation and deactivation of modules as well as the communication with the backend and the cognition/behavior thread.

Testing individual modules is done by faking data on the blackboard and running a single module. Alternatively the bodycontrol may run independently from the rest of the framework. In this dummy frontend the whole cognition and behavior thread is replaced with a network thread that listens to our bembelDbug network protocol and can pass arbitrary commands to the same thread safe command queue the behavior would use from inside the JRLSoccer framework.

The modules are not interdependent on each other for compilation; they only depend on the blackboard that needs to have the correct fields and has no dependencies on it's own. Logical dependencies are declared on registration of the modules in a single file, where each module gets a human readable ID, and optional dependencies and priorities are declared, determining the execution sequence.

Compared to the standard dependency injection design pattern this design omits the need for defining interfaces. It also differs from the classical blackboard approach, as its control shell has no knowledge of the data the modules may modify. This puts more responsibility for module dependencies on the programmer, which is mitigated by the blackboard and the inclusion file giving an complete overview over all modules, data and dependencies.

For the composition of motions we provide some utility functions. Inverse kinematics of the lower body, stabilizing functions using the arms or legs, and interpolation functions can be used to program the motions in a conceptual space closer to how we would describe them as humans. Additionally most motions have different phases or sections for which we adopt the well known finite state machine concept. These concepts allow us to define states interpolating between keyframes that are calculated with inverse kinematics. On motions such as a stand up move we can check if a part of the motion succeeded and repeat parts

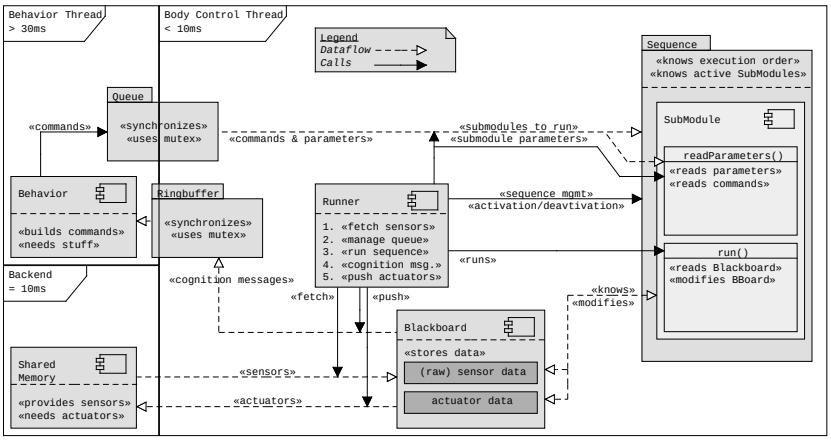

Fig. 7. Bodycontrol and Motion Framework.

as necessary. Through the return values of the states a motion may signal if it is currently stable or in a critical section and should not be interrupted.

6 Whistle Detection

Our whistle detection uses the `alsa` library to access microphones on the Nao robot. A Fast Fourier transformation is run on the acquired audio data using the library `fftw3`[5]. Once the data is detected in the frequency domain we check whether the frequency where most of the signal lies is above 2000 Hz. As the detection itself needs a lot of computational resources, the detection can be enabled only in special game situations like the set state and is disabled in the playing state.

Last year we published the whistle detection on our Bembelbots Github page[6]. The release also includes a small debugger to test the whistle detection on `wav` files.

7 Summary

Every year we, a team of mostly undergraduates, try to meet the challenges introduced by the SPL. Based on our experience over the last years we are able to maintain our unique JRLsoccer framework, described in Section 2. This is only possible due to our activity at various competitions which give us a good overview of our progress. Over the past year we made major changes to the

[5] Fast fourier transformation, `www.fftw.org/`
[6] Bembelbots whistle detection: `github.com/Bembelbots/NaoWhistleDetection`

main components of our framework including: localization, vision, motion and behavior.

Also, this year we applied for submission to the symposium in order to contribute to the community. We are glad to participate at RoboCup 2017 in Japan and hope to display the improvements that we have made.

References

1. Fürtig, Andreas, Holger Friedrich, and Rudolf Mester (2010): "Robust Pixel Classification for RoboCup." Farbworkshop Ilmenau
2. Becker, Christian (2013): Linienbasierte Featuredetektion und Positionstracking durch Voronoi-Diagramme in einer symmetrischen Umgebung, Diploma thesis at Goethe University, Frankfurt
3. Ditzel, Sina (2016): Selbstlokalisierung des Nao-Roboters im RoboCup mittels Partikel Filter, Bachelor thesis at Goethe University, Frankfurt
4. Hess, Timm (2016): Training convolutional neural networks on virtual examples for object classification in the RoboCup-environment, Bachelor thesis at Goehte University, Frankfurt
5. Brast, Jonathan Cyriax (2017): NAO Body Control and Motion Framework for RoboCup, Bachelor thesis at Goethe University, Frankfurt
6. Zheltonozhskiy, E.: Tinydnn. https://github.com/tiny-dnn/tiny-dnn (2017). Accessed: 09.05.2017
7. Min Lin and Qiang Chen and Shuicheng Yan(2013): Network In Network, CoRR
8. Reinhardt, Thomas (2011): Kalibrierungsfreie Bildverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfuball, Master thesis at Hochschule für Technik, Wirtschaft und Kultur Leipzig
9. R. Douc and O. Cappe (2005): Comparison of resampling schemes for particle filtering, ISPA 2005