DAInamite

Team Description for RoboCup 2013

Axel Heßler, Yuan Xu, Erdene-Ochir Tuguldur and Martin Berger
{axel.hessler, yuan.xu, tuguldur.erdene-ochir,
martin.berger}@dai-labor.de
http://www.dainamite.de

DAI-Labor, Technische Universität Berlin, Germany

Abstract. This paper describes the status of the team DAInamite from DAI-Lab, TU-Berlin regarding its development progress for RoboCup 2013 in the Standard Platform League. We give a brief overview of the team's history and constitution, our general architecture and its relevant components like motion, vision, and behavior.

1 Introduction

The team DAInamite from the Distributed Artificial Intelligence Laboratory of the Technical University of Berlin wants to take part in the *Standard Platform League* (SPL) during the RoboCup 2013 in Eindhoven, Netherlands, with its five NAOs (RoboCup edition V4 H21).

The team's origin lies within the 2D soccer simulation league, in which it participated multiple times since 2006 [1–4]. DAInamite's first participation in the SPL was during the RoboCup German Open 2012 in Magdeburg. For that we had implemented a simple agent with a *sense-think-act* execution cycle in the Java programming language, and used as many of Aldebaran's modules as possible (for example the ALMotion-module, including the walk and keyframes for standing up). We collected some additional practical experience during a 14-day long event called Ideenpark[1] in August 2012. There we played demonstration games together with members of the NAO Devils SPL team from TU Dortmund, and learned a lot from them about humanoid robot soccer.

Shortly after these two events we switched the programming languages to Python and C++. The main advantage for using Python is having a flexible programming language for rapid development of new ideas and prototypes. Time critical parts such as motion, and vision are implemented in C++, other modules such as localization, and behavior are implemented in Python. We still rely on NAOqi architecture as a communication infrastructure for modules, and the communication overhead between proxy and broker can be removed by embedding Python code in C++ process.

[1] http://www.ideenpark.de/ideenpark/funbox/roboterfussball

2 Team Members

The team is constituted of undergradute, graduate students and postdocs from faculties IV (Computer Science and Electrical Engineering) and II (Mathematics and Natural Sciences) of the Technical University of Berlin (TU-Berlin).

The currently active team members include:

- Axel Heßler (team leader),
- Erdene-Ochir Tuguldur,
- Martin Berger,
- Yuan Xu, postdoc,
- Sean Violante,
- Lars Borchert,
- and Nadine Rohde.

The team is hosted at the chair Agent Technologies in Business Applications and Telecommunication (AOT) and the DAI-Laboratories (DAI-Lab) of TU-Berlin. Prof. Dr. Sahin Albayrak is the head of the chair AOT and founder and head of the DAI-Lab. The DAI-Lab performs applied research and development of new systems and services and apply and test these solutions in real environments to make them tangible for users. Current application fields include, electromobility, smart grid, health, ambient assisted living, security, and service robotics.

Concrete research fields of our team members are agent-oriented software engineering, agent testbeds, cooperation and coordination, machine learning, planning and scheduling, computer vision, optimization, and human-robot interaction. The main application field of our NAO robots is teaching agent-oriented and robotic principles.

3 Architecture

We are using Aldebaran's NAOqi architecture as the basis. As mentioned earlier, the main policy was to use the existing modules if possible and implement missing functionality by adding our new modules. These new modules are written in either Python or C++, depending on the requirements.

3.1 Motion

We have developed our own motion module in C++ for better performance in soccer games. Especially, we implemented a fast (20 cm/s) omni-directional walk based on *Linear Inverted Pendulum* [5].

We also keep our motion module compatible with other modules from Aldebaran. Our replacement module implements the API of ALMotion and functionality such as *Self-collision avoidance*, a simple *Fall Manager*, and *Smart Stiffness* were developed as well. Furthermore, the module provides odometry data and

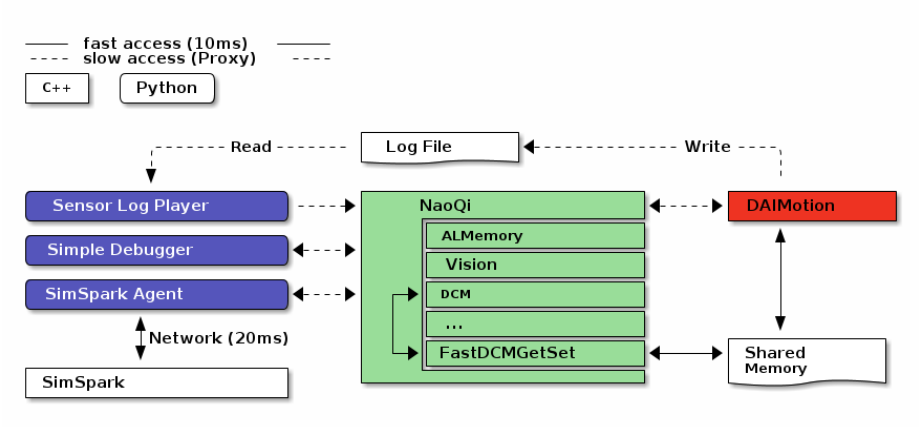

Fig. 1. Architecture of motion module.

homogenous transformation matrices for both cameras, derived from the robot's configuration, for other modules.

In order to test and debug our motion module easily, we divided it into several different sub-modules, see Figure 1. The *DAIMotion* can run as a local or a remote module. It accesses the DCM through shared memory when it is run locally on the robot. The module can also run with recorded logfiles and the SimSpark simulator.

3.2 Vision

Our team has implemented a calibration free vision algorithm strongly inspired by [6]. Like our motion module, the vision module is also implemented in C/C++ and replaces the Aldebaran video device module. To accelerate image aquisition, we are using *Video4Linux* directly to capture images from both cameras in parallel.

Currently, the vision module is able to detect the goal posts, the field border, lines and the ball. An exemplary result of the vision processing is visualized in figure 2. Detected entities are highlighted in different colors. These results are then written into the agent's central memory managed by Aldebaran's module (*ALMemory*), from which other modules, such as localization, can retrieve them.

Furthermore, the vision module provides methods for debugging and configuration, such as setting camera parameters and enabling or disabling processing of either camera.

3.3 Localization

For localization, a particle filter is implemented in Python using NumPy. Perceived goalposts, field lines, and the field's border are used as features for evaluating the hypotheses for the robot's position. Odometry for the predict phase

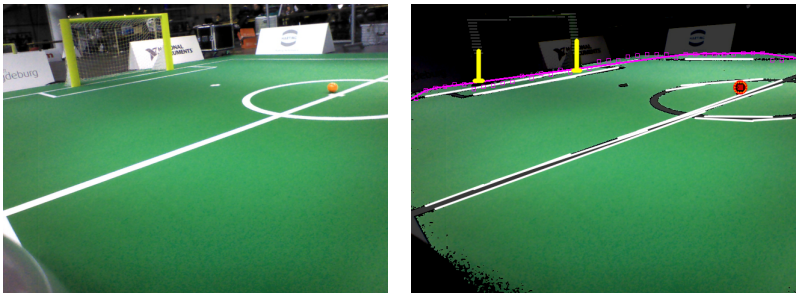

Fig. 2. Original image (left) and a visualization of the processing results (right) for a sample image. Detected elements are highlighted: Field border (magenta) and field pixels, goal posts (yellow), line-segments (white), and ball (red).

is provided by the motion module. The robot's position and orientation have to be tracked continuously, to distinguish the opponent's goal from its own.

To reduce the risk of scoring on our own goal, the goalie informs potentially delocalized players when the ball is approaching its (and their own) goal so they can correct their orientation if necessary.

It is essential to use the information that can be inferred from the rules. At the beginning of each kick-off and when re-entering the game after having been penalized, for example, players assume they are on their own half of the field.

3.4 Ball tracking

Like our other high level components, the ball tracking is also implemented in Python. Our vision module reliably detects the ball's position if a ball is present in the image and not heavily occluded. But if no ball is present in the current image, the vision occasionally returns false positives. To cope with this situation, we have implemented a multi-model Kalman filter as described in [7]. The detected balls' positions are transformed from pixel coordinates to relative coordinates in the robot-frame using the camera matrix. These measurements are then integrated into the best fitting hypothesis or spawn new hyptoheses, if they are deemed outliers for the existing ones.

3.5 Behavior

The behavior is implemented in Python using Hierarchical State Machines in an attempt to simplify high-level debugging. A visual representation of the agents behavior can be generated from these state machines using *DOT* [8]. Figure 3 shows an examplary image of a part of our striker's behavior.

Currently, there are three roles defined in our NAO team. The robots communicate with each other and share their perceptions: Their own position, and (relative) ball position.

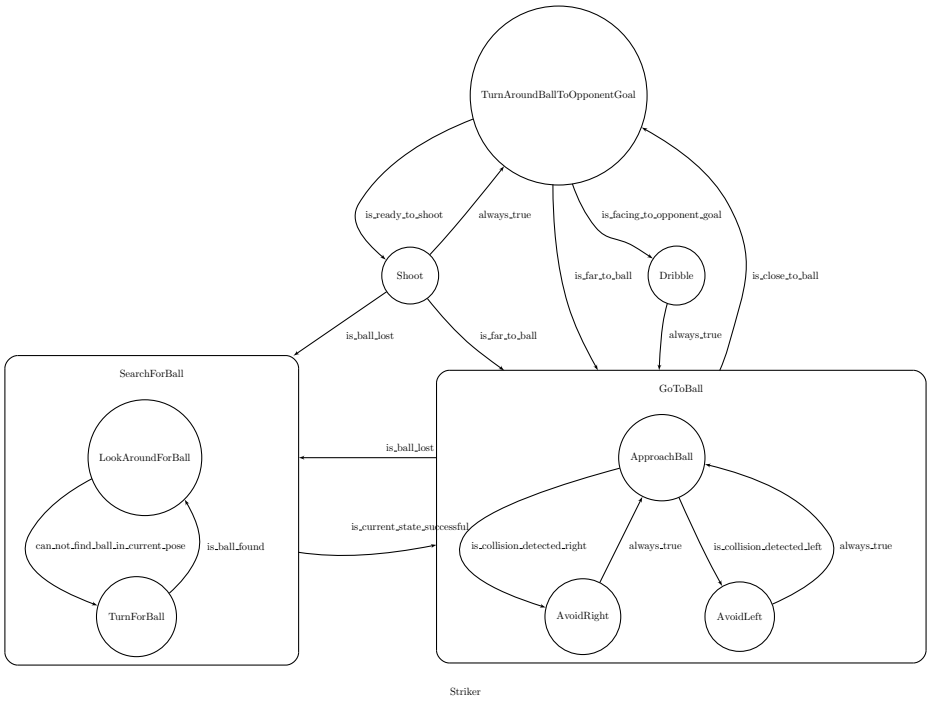

Fig. 3. Generated visualization showing a part of the active striker's behavior during gamestate Playing

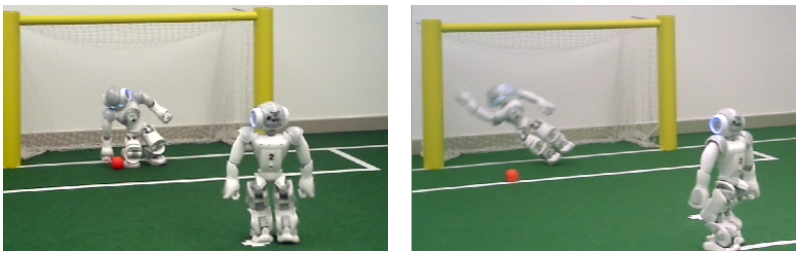

Fig. 4. Snapshot of the goalie performing a save while staying up straight (left) and by falling to the side (right).

– *Striker:* The field player who is closest to the ball becomes the striker. It approaches the ball and positions itself behind it into the direction of what it considers to be the opponent's goal. The robot then executes a kick-motion and repeats.
– *Supporter:* The field players who are not currently assigned the striker role become supporters. Supporters do not actively pursue the ball, but assume designated positions on the field.
– *Goalie:* The goalie tries to stay roughly in the middle between its own two goal posts. During the game, the goalie computes the direction of the ball movement by using the information provided by the ball tracking module. The robot tries to parry if the direction of the ball points towards its own goal and the ball's velocity exceeds a threshold. It will then try to block the ball by executing a parry motion to the appropiate side that is either lowering one arm to the ground while remaining standing, or executing a save by falling to the side, as depicted in Figure 4.

4 Conclusion

We gave an overview of our first approaches to the major algorithmic challenges that have to be faced in humanoid robotic soccer. There is still a lot of room for improvements when working on specific tasks in depth. Localization and visual perception are to be improved and speed up further. Robot detection and collision avoidance with other players has not yet been implemented. Also the behavior is still very simple, information about the ball's speed and other robots' positions still needs to be fully incorporated into the decision making.

Acknowledgments

Large influences came from four German teams, namely: The Nao Team Humboldt from Humboldt University Berlin, the Nao-Team HTWK from Leipzig University of Applied Sciences, the Nao-Devils from TU Dortmund, and B-Human from Bremen University. We mainly learned about concepts and algorithms from them and want to thank them again at this point. We did not use any code of them so far.

References

1. Endert, H., Wetzker, R., Karbe, T., Heßler, A., Brossmann, F.: The dainamite agent framework. Technical report, Dai-labor TU Berlin (2006)
2. Endert, H., Karbe, T., Krahmann, J., Trollmann, F., Kuhnen, N.: The dainamite 2008 team description. RoboCup 2008 (2008)
3. Endert, H., Karbe, T., Krahmann, J., Trollmann, F.: The DAInamite 2009 team description. RoboCup 2009 (2009)
4. Hessler, A., Berger, M., Endert, H.: Dainamite 2011 team description paper. Robocup 2011 (2011)

5. Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., Hirukawa, H.: Biped walking pattern generation by using preview control of zero-moment point. In: ICRA. (2003) 1620–1626
6. Reinhardt, T.: Kalibrierungsfreie Bildverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfußball. Master's thesis, Hochschule für Technik, Wirtschaft und Kultur Leipzig (2011)
7. Quinlan, M.J., Middleton, R.H.: Multiple model kalman filters: a localization technique for robocup soccer. In Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S., eds.: RoboCup 2009. Springer-Verlag, Berlin, Heidelberg (2010) 276–287
8. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. Software - Practice and Experience **30**(11) (2000) 1203–1233