DAInamite

Team Description 2014

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 document describes the progress of the team DAInamite from DAI-Lab, TU-Berlin (Germany) regarding the SPL. We give a brief overview of the team's history, constitution, our general architecture, its relevant components (motion, vision, and behavior), as well as employed tools, and point out recent work.

1 Introduction

Team DAInamite's origin lies within the 2D soccer simulation league, in which it participated a few times since 2006 [1–4]. DAInamite's first appearance in the SPL league was at the German Open 2012 in Magdeburg. And its first participation in the world championship in the SPL was in RoboCup 2013 in Eindhoven [5], reaching the quater finals.

In addition to C++, Python is mainly used in the team's code. The time-critical components for motion, and vision are implemented in C++. The remaining modules such as localization, behavior, and ball-tracking are implemented in Python.

2 Team Constitution

The team is constituted of undergradutes, graduate students, and postdocs of TU Berlin, mainly from faculty IV (CS&EE) [1]. The team is hosted at the chair Agent Technologies in Business Applications and Telecommunication (AOT) and the DAI-Laboratories (DAI-Lab) at TU Berlin. Prof. Sahin Albayrak is the head of chair AOT and founder and head of the DAI-Lab. Concrete research interests of the team members are agent-oriented software engineering, autonomous systems, cooperation & coordination, and human-robot-interaction.

[1] http://www.tu-berlin.de, http://www.dai-labor.de, http://www.aot.tu-berlin.de/, http://www.dainamite.de

3 Architecture

We are using Aldebaran's NAOqi architecture as the basis, to benefit from the following features: Call functions in both C++ and Python, and the ability to execute functions also remotely over the network.

To archive higher computational performance during soccer games, when the system is run as a whole, we embed the Python code (using Boost Python) as a local module into NAOqi's process to remove communication overhead.

Our main policy was to first add missing functionality as necessary and then replace modules to imporve performance. Depending on performance requirements these modules are written in either Python or C++.

3.1 Motion

We developed our own motion module in C++ for better performance in soccer games. Especially, we are using a fast (20 cm/s) omni-directional walk based on *Linear Inverted Pendulum* [6].

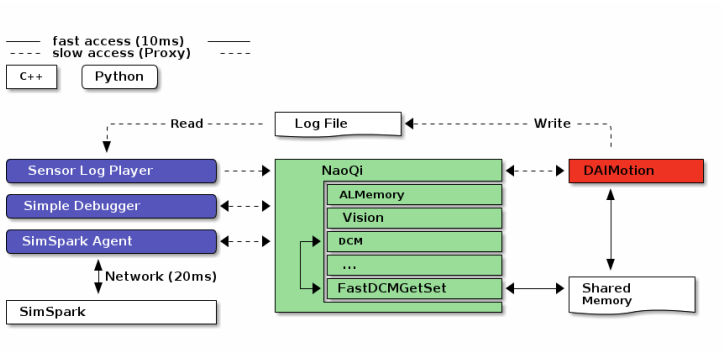

Fig. 1. Architecture of motion module.

The motion module is compatible with the API of the original motion module, shipped by Aldebaran. The basic API and functionalities (such as Self-collision avoidance, Smart Stiffness, etc.) of ALMotion are implemented, so our motion module can serve as an enhanced replacement of ALMotion. Additionally, our motion module provides to other modules odometry data, and homogenous transformation matrices for both cameras, derived from the robot's configuration.

In order to test and debug our motion module easily, we divided it into several different sub-modules, see Figure 1. As the core part, *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; and data is communicated via NAOqi when it is run with recorded logfiles or the SimSpark simulator.

3.2 Vision

Our team uses a vision algorithm strongly inspired by [7]. 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* to capture images from both cameras in parallel.

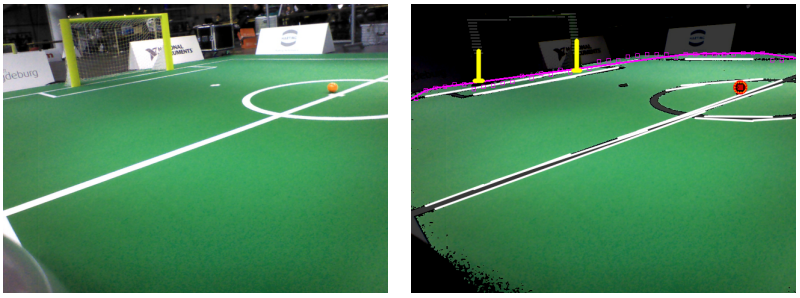

Fig. 2. Original image (left) and a visualization of the processing results (right) for a sample image. Detected elements are highlighted: Field border (magenta) and field pixels, goal posts (yellow), line-segments (white), and ball (red).

Currently, the vision module is able to detect goal posts, ball, field lines, field border, and obstacles. Visual obstacle detection is done using blob detection on a downsampled image, treating regions with a color that differs from the detected field color as obstacles, ignoring areas of prior detected objects (i.e. goal posts and lines). An exemplary result of the vision processing is pictured in figure 2, where detected entities are highlighted in different colors. These results are then written into the agent's blackboard 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 Ball tracking

Like other high level components, the ball tracking is also implemented in Python. Our vision module fails to detect a ball if its present in the image, not further away than six meters, and not heavily occluded. Because of the magenta jerseys introduced in RoboCup 2013 [8], we adjusted the ball detection to cope with multiple red peaks on the field. Within the area of each detected obstacle the ball detection is run additionally and these hypotheses are checked and filtered afterwards.

If no ball is present in the current image, the vision may occasionally return false positives. To cope with this situation, we are using a multi-model Kalman

filter (i.a. as described in [9]). The detected balls' positions are tracked in 2D, relative to the robot. These measurements are then integrated into the best fitting hypothesis or spawn new hyptoheses, if deemed outliers for the existing hypotheses.

3.4 Localization

For localization, a particle filter is implemented in Python using NumPy and SciPy. 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 is provided by the motion module.

At the beginning of each kick-off and when re-entering the game after having been penalized, for example, players assume they are within their own half of the field, according to the rules. Symmetry is not actively broken, so the robots continuously track their position, to distinguish the opponent's goal from their own.

To reduce the risk of scoring on our own goal in case a field player becomes delocalized, the goalie informs all players when the ball is approaching its goal, so players can correct their orientation if necessary. This only works under the assumption, the goalie is localized and in the correct position.

3.5 Behavior

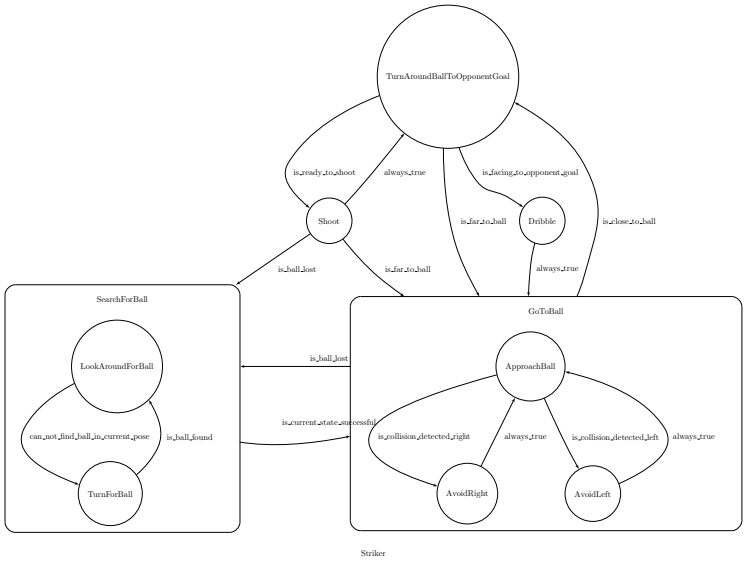

Fig. 3. Generated visualization showing a part of the active striker's behavior during gamestate Playing

The behavior is implemented in Python using hierarchical state machines. A visual representation of the agents behavior can be generated from these state machines using *DOT* [10]. Figure 3 shows an examplary image of a part of our striker's behavior. The *Team viewer* can display and log all the communicated data from robots and *Game Controller*. This helps to analyze game situations post mortem.

The robots communicate with each other and share their own position, the ball's position, and their behaviour's currently active state.

4 Tools

We mostly use different small tools for particular purposes.

For quick experiments with Nao, we created an extension to IPython, termed inao. It provides an enhanced interactive Python shell to interact with NAO, providing fast access to all the module's proxies (motion, memory, vision). Please watch the video [2] to get an impression of its use.

To help assessing the effects of code-changes (primarilly for behavior) to the team's in-game perfomance, we are using a modified version of the SimSpark 3D soccer simulator from Nao-Team Humboldt[3] and continuous integration to let recent versions play against their predecessors.

Fig. 4. Team Viewer showing a replay of a situation in a game.

During games the Team Viewer (figure 4) helps in evaluating the overall state of the robots as a team. It also records the team-communication send by the robots and packets sent by the game controller. To a limited extend this

[2] `http://youtu.be/3e69bmgiH7I`

[3] `https://github.com/BerlinUnited/SimSpark-SPL`

enables us to replay certain situation for the robot's behavior and evaluate its response.

5 Conclusion and Future Work

We gave an overview of our architecture and approaches to the major algorithmic challenges that have to be faced in humanoid robotic soccer. Recent work has been started in robot detection and automating calibration (for joints, camera matrix, and color) to reduce setup efforts. Also we are currently working on improving our localization, since it is still fragile in the face of visual occlusions.

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. Hessler, A., Xu, Y., Tuguldur, E.O., Berger, M.: DAInamite team description for robocup 2013. RoboCup-2013: Robot Soccer World Cup XVII (2013)
6. 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
7. Reinhardt, T.: Kalibrierungsfreie Bildverarbeitungsalgorithmen zur echtzeitfähigen Objekterkennung im Roboterfußball. Master's thesis, Hochschule für Technik, Wirtschaft und Kultur Leipzig (2011)
8. RoboCup Technical Committee: Robocup standard platform league (nao) rule book (2013)
9. 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
10. 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