DAInamite

Team Description 2016

Yuan Xu, Martin Berger, Qian Qian and Erdene-Ochir Tuguldur
{yuan.xu, martin.berger, tuguldur.erdene-ochir}@dai-labor.de

DAI-Labor, Technische Universität Berlin, Germany
http://www.dainamite.de

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 [1,2,3,4]. The team is active in the SPL since 2012, making it to the quarter finals in RoboCup 2013 and finishing 2nd place in Iran Open 2016.

Our code is mainly written in C++ and Python. The main advantage for using Python is having a flexible programming language for rapid development of new ideas and prototypes. The time-critical components for motion, vision and localization are implemented in C++. The remaining modules such as 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

The architecture of our code is following the principles of agent architecture, implementing for example the sense – think – act loop. Furthermore, due to the nature of parallelism of sensors, processors and actuators, the program runs in different threads for better performance, see Figure 1.

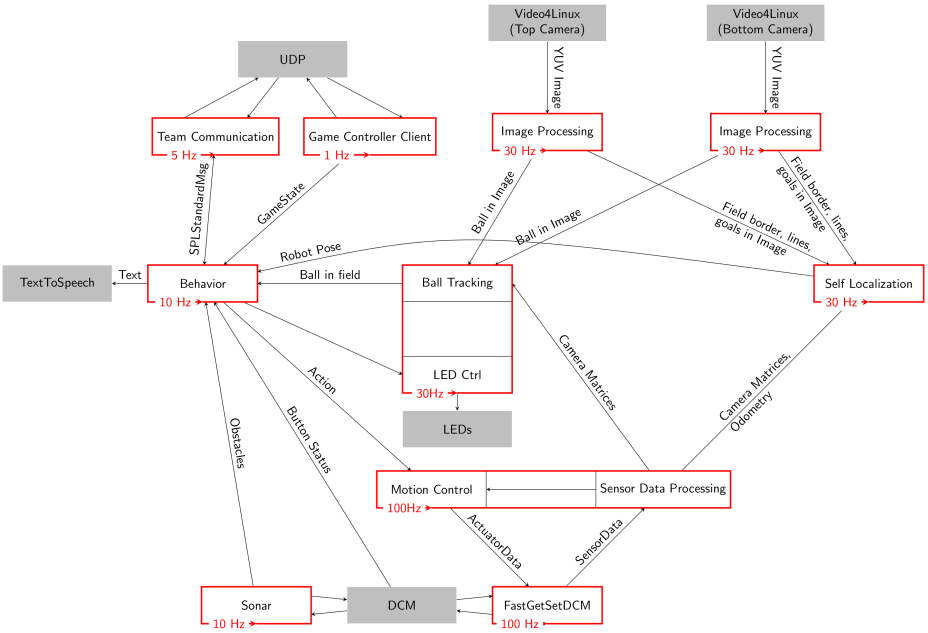

Fig. 1. Process of DAInamite's software architecture. The modules in gray are provide by Linux system or NaoQi. The other modules are developed for RoboCup soccer, and run in different frequencies.

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 for experiments or debugging.

To achieve 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.

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 the popular *Linear Inverted Pendulum* [6].

Our motion module is API-compatible to the original motion module *ALMotion*, developed by Aldebaran. The basic API and functionalities (such as Self-collision avoidance, Smart Stiffness, etc.) of ALMotion are replicated, so our motion module can serve as an enhanced replacement. Additionally, our motion module provides 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. 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 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 acquisition, we are using *Video4Linux* to capture images from both cameras in parallel.

Currently, the vision module is able to detect goal posts, ball, field lines, field border, and obstacles. Visual obstacle detection is done on a down sampled 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.

In order to detect the ball which does not have a unique color anymore, a Convolutional Neural Networks (CNN) is used to classify image pathes as ball. First, the detected visual obstacles are used as region of interest (ROI) for possibly containing a ball, then the downsampled ROI is passed to CNN. If the ROI is classified as ball, the ball's contour is found by fitting a circle onto the white and green edges using RANSAC. Furthermore, the confidence of the ball is calculated by the size of the ball projected on the robot frame. We use Caffe[8] to train the CNN, and tiny-cnn[9] for prediction. For performance reason, we are using a small network at the moment, see Figure 3.

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

3.3 Localization

Since IranOpen 2016, a multi-hypotheses Kalman filter based localization was used instead of the previous particle filter based localization. For performance consideration, it is implemented as a C++ library with Python bindings. The multi-hypotheses Kalman filter based localization surpasses the particle filter in terms of position accuracy and also gives smoother robot walking trajectory during position tracking. In addition to landmarks like field lines, field's border and

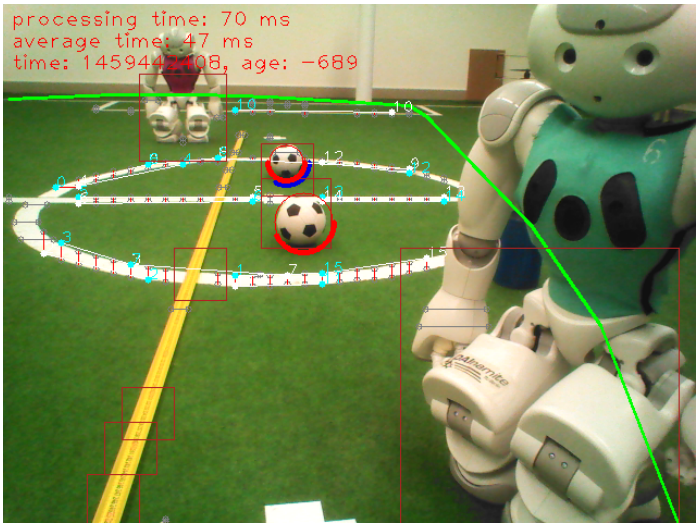

Fig. 2. Visualization of the processing results for a sample image. Detected elements are highlighted: Field border (green), line-segments (white), visual obstacles (red box) and ball (red circle). The red and blue circle around ball represent ball confidence in image and robot frame respectively.

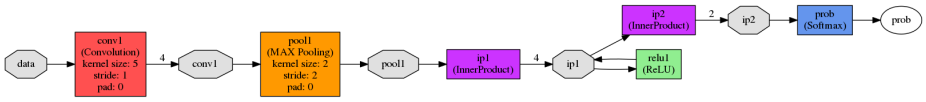

Fig. 3. The network architecture of Convolutional Neural Networks used for ball detection.

L, T, X corners, the detection of center circle and penalty areas are also implemented, and they are serving as less ambiguous features improving localization accuracy. Similar to a sensor reseting step in particle filter, multiple samples are generated when ambiguous landmarks are seen, which allows faster positon recovery when the positon is lost. The confidence of each sample is calculated based on the matching error of the global features.

We are not using any active means to break field symmetry, so the robots have to 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 information communicated by the most stationary player, the goalie, is used by the attacking field player to correct its orientation if necessary. This only works under the assumption, the goalie is localized and roughly in the correct position.

3.4 Behavior

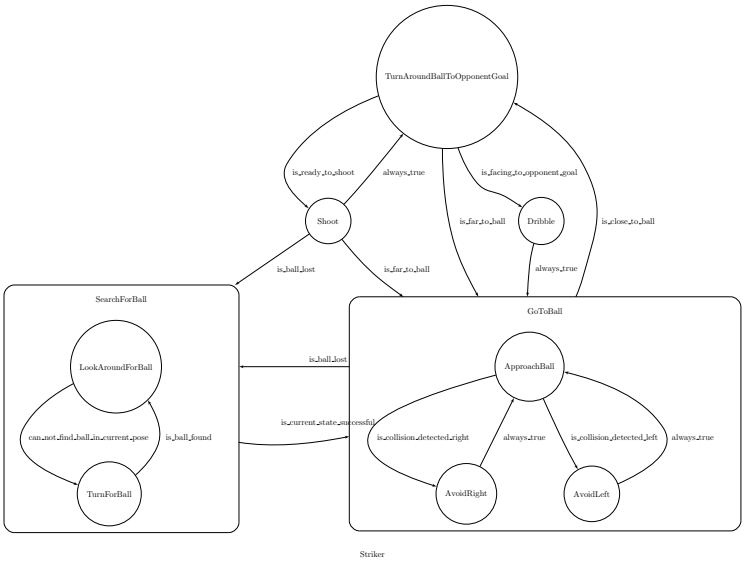

Fig. 4. Generated visualization showing a part of the active striker's behavior during gamestate Playing

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* [12]. Figure 4 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.

Based on the shared information unsing the standard message format, field players coordinate based on who is closest to the ball, as well as their currently active behavior state (if available) or the communicated intentions.

Some recent changes in the face of rule changes that more heavily punish repeating the same rule offenses (up to removal from the half), we improved the overall performance of the stand-up routine and are testing some improved decisionmaking in cornercases to reduce the number of avoidable infractions.

4 Tools

We have a number of different small tools for particular purposes.

For quick experiments with Nao, we created an extension to IPython, named 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 (mainly 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 in a small tournament.

Fig. 5. Team Viewer showing a replay of a situation in a game.

During games the Team Viewer (figure 5) 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 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 current architecture and approaches to the major algorithmic challenges that have to be faced in humanoid robotic soccer. We are continuously working on automating calibration and reducing setup efforts. Also we are still working on improving our localization and overall game performance.

References

1. Endert, H., Wetzker, R., Karbe, T., Heßler, A., Brossmann, F.: The dainamite agent framework. Technical report, Dai-labor TU Berlin (2006)

[2] `https://youtu.be/3e69bmgiH7I`
[3] `https://github.com/BerlinUnited/SimSpark-SPL`

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. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
9. tiny cnn: A header only, dependency-free deep learning framework in c++11. `https://github.com/nyanp/tiny-cnn` (2016)
10. RoboCup Technical Committee: Robocup standard platform league (nao) rule book (2013)
11. 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
12. 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