社団法人 人工知能学会
Japanese Society for
Artificial Intelligence

人工知能学会研究会資料
JSAI Technical Report
SIG-Challenge-A701-4 (5/4)

Successful Teaching of Agent-Based Programming to Novice Undergrads in a Robotic Soccer Crash Course

Rodrigo da Silva Guerra [1], Joschka Boedecker [1], Hiroshi Ishiguro[1,2], Minoru Asada [1,2]
[1] Graduate School of Engineering, Osaka University, Osaka, Japan
[2] JST Erato Asada Project
{rodrigo.guerra,joschka.boedecker,ishiguro,asada}@ams.eng.osaka-u.ac.jp

Abstract

This work describes a method for introducing agent based programming concepts as a hands-on experience targeted at inexperienced students learning C as their first programming language. The approach is based on three main features: (a) a simplified C interface to the RoboCup 2D soccer simulation framework; (b) a strict agent-centered polar vector arithmetics approach for describing soccer skills and agent behaviors; and (3) on a tournament at the end of the course to give students an opportunity to evaluate their programs against one another in a fun environment. We present data of our experience performing a five-day (15h) course with sixty second-year engineering bachelor students who had just barely learned the basics of computer programming. We show how, based on the described scheme, students with very limited computer programming experience became able to develop their own teams of autonomous agents, in some cases including concepts such as dynamic role-assignment, multi-agent coordination, and team formation. We defend the method presented here helps exposing the students to valuable experience ahead of their normal schedule, and boosting overall motivation and performance.

1 Introduction

Even though using real and virtual robots in engineering related courses is known to boost motivation of students [1, 12, 5, 4], care needs to be taken in the course design to minimize the many practical problems of working with the robots or simulation environments. This is especially true for students with little programming experience. In this paper, we present the contents and results from a very short introductory level programming course for undergraduate engineering majors using simulated robotic soccer as a framework to teach agent-based programming, and to give students an opportunity to get hands-on experience putting to work the programming skills they had just barely learned. The approach we chose corresponds to what Lund [8] terms *guided constructionism*, i.e., a combination of traditional constructionist approaches [10, 11] and explicit guidance in forms of lectures and coaching by more experienced students.

We built the course around the 2D soccer simulator [9] which has been widely accepted as a standard research and educational platform for multi-agent applications. Numerous works using this tool have been carried out, including both scientific level research papers (e.g. [7, 14, 13]) and agent programming courses (see e.g. [2, 5]). This helped making this system a robust platform for testing ideas in multi-agent disciplines. However, despite of its wide-spread and general acceptance, the two-dimensional simulation framework is still rather complex, requiring the kind of specialized knowledge typical of programmers with rather mature experience. While such complexity often regards features without which interesting multi-agent problems could not be properly attacked, this same complexity also comes as an obstacle to the non-experienced programming beginners as pointed out in [2, 5].

We believe the limited programming skills of students should not prevent them from experimenting with the basic concepts of agent-based programming. On the contrary, we think allowing these students to experiment with their first codes already in such a dynamic and motivating environment helps boosting their learning of programming concepts as they become necessary for producing more elaborate autonomous soccer teams. Aimplified interface to program the agents was created making it possible to cope with a very narrow time frame of only 5 sessions (each of 3 hours duration) with lectures and programming work plus an additional separate session for a class tournament.

The rest of the paper is organized as follows: Section 2 describes the simplified C-interface around which the course was formulated. Section 3 shows in detail the strict vectorial approach which worked as the cen-

tral conceptual tool for programming the agents. Section 4 present specific details about the organization of the course. Section 5 gives some qualitative impressions from the authors based on the obtained results. Finally, sections 6 summarizes the main contributions of the paper and discusses directions future works.

2 The Simplified C-Interface

We prepared a set of wrapping functions allowing students to implement their code in a very compact and simplified way requiring only the use of standard C code. The idea is similar to that of RoboSoc [6], but with a much stronger focus on simplicity – at the price of loosing generality. These wrapping functions were constructed around the Trilearn base code published in 2002 [3].

The whole environment provided by the Trilearn base code was wrapped into four control functions and a few sense/act functions. Some of the sense/act functions were simple wrapping of existing C++ functions of the original code while others were implemented from scratch or heavily simplified. The four control functions are described below:

- `pv_init` – Includes all initialization procedures that should happen before the main loop in order to prepare the agent and including initial communication with the server. The only argument passed to this function is the desired team name;
- `pv_update` – Includes all the necessary routines for parsing messages received from the server and updating the internal world model of the Trilearn base code. This function is called inside the main loop right at the beginning, so that actualized sensor values can be assured;
- `pv_flush` – Takes all accumulated action commands, assembles them into messages and sends them to the server so that the agent can actually perform them. This eliminates the burden of sending the commands every time an agent takes a decision. Should be called in the last instruction inside the main loop;
- `pv_close` – Performs all procedures necessary for finishing the program, de-allocating resources.

The listing 1 shows an example of a complete agent which is capable of passing the ball to a teammate (variable names are consistent with figure 1). Together with our strict vector approach (which is detailed in section 3) this simplified interface enabled the students with very little programming experience to program a variety of soccer playing behaviors in a very clear and intelligible way.

3 The Vector Arithmetics Approach

Most people learn basic operations with vectors already at school. Vector arithmetics are visually intuitive, especially in the two dimensional space, where calculations can be approximated by sketching simple strokes on a piece of paper. More than that, the analogy of the soccer field as a two-dimensional space gives a very straight

Listing 1: Example of a simple "passing" agent

```
#include <cinterface.h>
int main(int argc, char *argv[])
{
        struct strct_vector c;
        struct strct_vector e;
        pv_init("MyTeam");
        while (pv_update())
        {
                c = pv_getball();
                e = pv_getteammate(1);
                if (pv_cankick())
                {
                        pv_kick(e);
                } else {
                        pv_steerto(c);
                }
                pv_flush();
        }
        pv_close();
}
```

forward interpretation for directions and magnitudes of two-dimensional vectors which can represent forces, accelerations, velocities of players and ball. In fact, this approach is so straight forward that it is very common to see implementations involving two-dimensional vector arithmetics of some kind across the various RoboCup Soccer leagues.

In our approach we developed a complete framework where all essential elements necessary for making a team of soccer playing agents could be implemented by the exclusive use of two-dimensional vector arithmetics and nothing else. Moreover, we took care of describing all necessary components relative to the self in an agent-centered approach excluding completely explicit global coordinates of any sort. This agent-centered perspective allows a deeper understanding on the agent perspective embodied with sensors and immerse in the environment.

In the figure 1 we illustrate how one can take advantage of the two-dimensional vector representation for a very visual strategy planning. Suppose, for instance, your agent were to mark an opponent by placing itself between the opponent and the ball. The vector from the ball to the opponent, according to the figure 1, is given by the expression $\mathbf{c} - \mathbf{d}$. One could simply scale down this vector, let's say, half-way ($\frac{\mathbf{c}-\mathbf{d}}{2}$), and sum to the vector representing the direction to the ball, yielding the expression $\frac{\mathbf{c}+\mathbf{c}-\mathbf{d}}{2}$. This last expression would be the direction the agent should go. Similarly, if an agent were to kick the ball ($\mathbf{c}$) into the center of the goal ($\frac{\mathbf{a}+\mathbf{b}}{2}$), it would need to go towards $\mathbf{c} + (\mathbf{c} - \frac{\mathbf{a}+\mathbf{b}}{2})(r_r + r_b)$, where r_r and r_b are the radius of the robot and the ball respectively (compare with the previous expression).

A web-based applet was developed in Java as a tool for helping the students draw their vector arithmetics on the screen of the computer and test their ideas on different game situations. See figure 2 for a screen shot.

To our understanding this simplified vector approach provides a powerful unifying interpretation which can be used across for a variety of very different soccer robots.

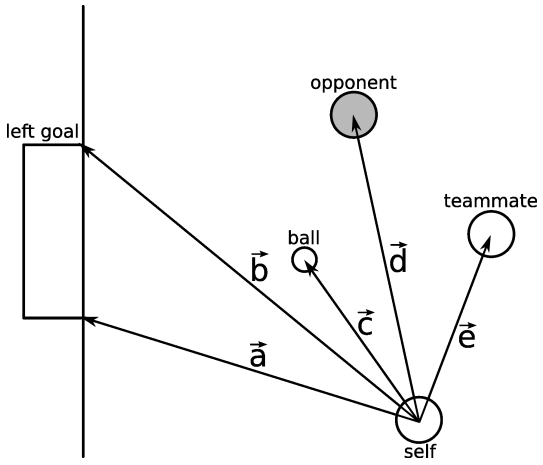

Figure 1: An example of game situation where all necessary elements can be visualized in terms of two-dimensional vectors

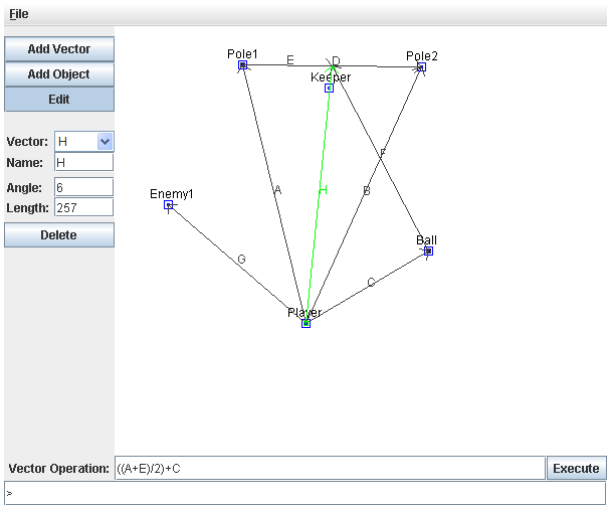

Figure 2: Screen shot of the Java vector applet developed for enabling students to visualise their different strategies and check resulting expressions.

See below some examples of very straight forward interpretation in the case of the most popular platforms.

Omni-directional camera: Suppose the vertical axis of revolution of the omni-camera is normal relative to the horizontal plane (of the field) and fixed in the robot (which is often the case). The center of the revolved image is the origin for the two-dimensional vector representation. Radial distances from the origin in the image have direct mapping into distances from the robot body.

Projective camera: For the sake of simplicity lets assume square pixels, pinhole model, and center of projection on center of image (usually very practical approximation when it comes to non-precision robotics). One can assume a simple mapping from the distances of arbitrary image points to the center of the image into the corresponding horizontal and vertical angles of their respective rays (taking the pinhole as the vertex). See figure 3. The horizontal angle α give the direction of the two-dimensional vector while the vertical angle β gives

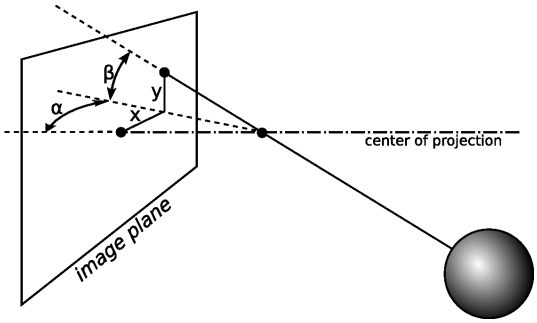

Figure 3: Example of simplified pinhole camera. One can find a simple mapping from the distances x and y to the corresponding angles α and β respectively. In the two-dimensional vector approach the angle α gives the direction and the magnitude is given by $d = f(\beta)$. This relation is derived by simple trigonometry

a direct mapping into distances on the floor (assuming camera is at constant high).

Pan & Tilt: Consider, images on the center of projection of the camera. Pan angles can be used directly as the directions of the two-dimensional vectors while tilt angles map into distances in the floor. The pinhole camera attached to the pan & tilt mechanism can be approximated by simply summing pan and tilt angles with horizontal and vertical camera angles respectively.

Control theorists experienced in mobile robotics would probably still argue it is not so straight forward to derive control laws when you have non-holonomic restrictions in the mobility of the robots (e.g. two wheeled differentially driven robots). Such problems have been focus of constant research in the past years due to the challenging control problem they represent (i.e. closed-form solution does not exist). But here again, this should not come as an obstacle to the learner who is eager to put a robot into movement into a simplistic and practical experiment – otherwise all beginners in robotics would require robots equipped with omni-wheels.

In the original two-dimensional simulation framework agents are moved by the sucessive use of `dash` & `turn` commands, for, respectively, turning and moving forward the agent's position. We implemented an interface which mimics the effect of having differential wheels (i.e. transforming the fictitious wheel velocitys into a corresponding series of `dash` & `turn` commands). This was done in an effort for keeping the virtual agent as closely related as possible to the most typical robotic architectures – at the price of restricting the original (less realistic) maneuverability of the agents.

In our simulated differential driven robot the velocity of each wheel could be set into three different constant values, both backward and forward, or zero (stopped). In such case, for example, $(+3, +3)$ would make the robot go forward, $(+2, -2)$ would make the robot spin clockwise and $(+3, +1)$ would make the robot go in an arc-trajectory to the direction forward/right. In order to face the problem of steering the robot into arbitrary

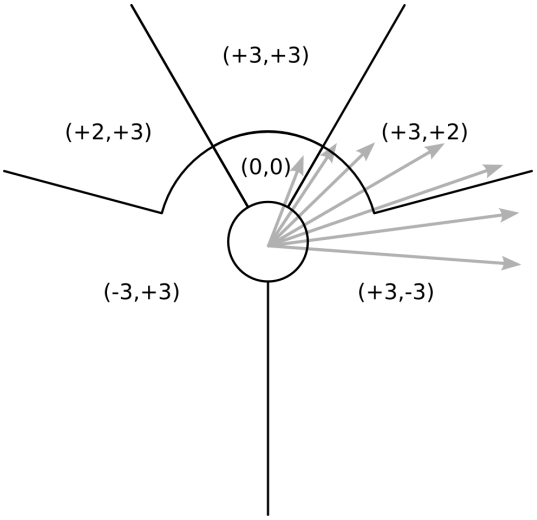

Figure 4: State machine implementing a simple steering algorithm for a two-wheeled differential driven robot

locations we implemented a simple state-machine. See figure 4. This approach was chosen for being simple and effective enough so that the student with their limited experience in programming (and robotics) could easily understand and eventually improve their agent's navigatibility by customizing their own steering methods. This way they not only could understand and use very realistic robots in their experiments but also had contact with a state-machine for decision making in their code.

4 Course Format

The course was performed at the Osaka University during the month of March of 2007 to an audience of 62 second-year engineering students (amoung which only 4 were females). It was composed by five separate lectures of three hours each, given twice a week plus an extra sixth lecture where the tournament was realized. The overall program of the class is summarized in the table 1.

Except for the first class, which was almost completely theoretical, all the other classes were build around the student exercise and practice. In the end of each class a homework was a assigned, which would involve and stretch concepts learned in class, but also bring up issues which would only be formally introduced in the next class. This was done in order for them to experience the needs before trying the solution, so that when the solution was presented its value could be more promptly understood. For instance, students would compile using the command prompt in the first class, but be introduced to Makefiles in the second class, and they would implement different skills (such as passing) inside their main loop in the second class but learn how to create more generic parameterized functions in the third class, and so forth. Despite all the theory being not completely new to them (they came fresh from a theoretical introduction to programming), it takes quite a lot of practice and experimentation until they can really put their knowledge into work in such a dynamical situation.

Day	Content summary
1st	Introduction, review of concepts of vector arithmetics, basic agent programming concepts
2nd	Introduce the most elementary code, explain the use of a simple Makefile, start working on very atomic behaviors (e.g. kick to goal and run to the ball)
3rd	Start generalizing these atomic behaviors with the introduction of simple functions (e.g. find closest teammate)
4th	Make the agent even more versatile introducing dynamic role assignment and general changes of behavior according to things such as own-id, side of play, etc.
5th	Divide the class in groups of four students and start developing their own teams for the final tournament.
6th	Tournament

Table 1: Summary of the course program

5 Results

A questionnaire was formulated in order to help evaluating the evolution of the interests and degree of confidence of the students in three main aspects: (a) programming, (b) robotics and (c) soccer. The questionnaire was distributed twice, firstly before the first class and again later, after the first round of games during the tournament. This questionnaire was extensive and composed by several multiple choice questions. In these questions students had to classify their interests, previous experience and self-confidence on many criteria regarding themes directly or indirectly related to the contents of the course. The speculations discussed in this section are based on the results of this questionnaire, together with the collected homework, the final team code and a final report.

To our surprise, all students were unanimous in that they had never even heard about the term "agent programming" before. On the other hand, when asked to list from the top of their heads names of robots and soccer players, robots like ASIMO and AIBO were cited more often than the most often cited soccer player (which was Ronaldinho). The authors interpret this as an strong indication that robotics is a rather popular subject among young engineering students in Japan.

At the end of the course, despite the fact that we didn't talk about real robots during any of the practical classes, students showed that they have increased significantly their confidence about how much they believed they could make a real robot play soccer. See figure 5.

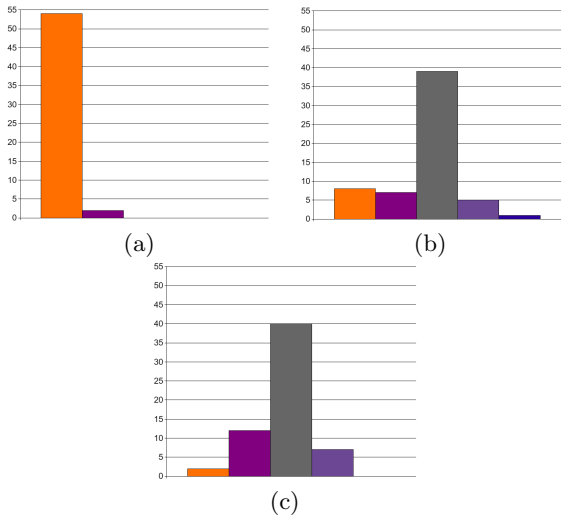

Figure 5: Results of the pool where students were invited to self-evaluate to what extent they believed they could make a real robot play soccer (a) when asked before and (b) when asked after the course, and similarly for making (c) an artificial agent play soccer after the course. The five columns in each chart represent the total number of students that chose each of the corresponding five options, which were, from left to right: (1) can nothing at all, (2) can do almost nothing, (3) can do a little, (4) can do well, (5) can do very well. (the neutral option was explicitly omitted)

Furthermore a much bigger number of students declared to have become more interested in both robot soccer and agent programming. Moreover, in the end of the course we found a very strong correlation between their confidence on their own agent programming skills with their confidence on how much they believed they could make a real robot play soccer (compare charts in figure 5-b and 5-c). On our interpretation the above indicates that, although we worked only with a very simplistic two-dimensional simulation environment, the students could still relate this to a broader concept applicable to real robots.

Furthermore, the authors noticed the positive effects of the tournament on the motivation of the students, which were often verbally expressed on their final reports, and also rather evident when reviewing the videos recorded during the tournament.

6 Discussion

This work presented an effective approach for exercising basic programming skills in the very dynamic environment of soccer simulation in a very short period of time. The method here presented together with the observed results support the idea that limitations on programming skills do not prevent students achieving their goals in the complex and dynamic multi-agent environment.

During the class we collected some strong evidence of the popularity of robotics among the students. For future work we plan to investigate how the (eventual) use of real robots in class would influence their performance if compared to the data collected during this course in which we used simulation only. Moreover, we plan to make our evaluation methods conform to suggested [12] standards thus enabling easy comparison of results.

7 Acknowledgements

The authors would like to thank to Chisato Yoshida for her help on the empirical portions of the work, to Matthias Bohnen for developing the nice Java applet for helping on vector arithmetics, to the teaching assistants for their great help both during the classes and later on analysing the collected data, and to the students themselves for their collaboration and understanding. The authors also want to thank JSPS and JST and the Japanese Ministry for Sports and Education for their financial support.

References

[1] John Anderson and Jacky Baltes. An agent-based approach to introductory robotics using robotic soccer. *International Journal of Robotics and Automation*, 21(2):141 – 152, April 2006.

[2] S. Corradeschi and J. Malec. How to make a challenging ai course enjoyable using the robocup soccer simulation system. In *RoboCup-98: Robot Soccer World Cup II*, Lecture Notes In Artificial Intelligence. Springer, 1998.

[3] Remco de Boer and Jelle R. Kok. The incremental development of a synthetic multi-agent system: the uva trilearn 2001 robotic soccer simulation team. Master's thesis, University of Amsterdam, 2002.

[4] Barry S. Fagin and Laurence Merkle. Quantitative analysis of the effects of robots on introductory computer science education. *ACM Journal of Educational Resources in Computing*, 2(4):1–18, December 2002.

[5] Frederik Heintz, Johann Kummeneje, and Paul Scerri. Simulated robocup in university undergraduate education. In *RoboCup 2000: Robot Soccer World Cup IV*, Lecture Notes In Artificial Intelligence, pages 309 – 314. Springer, 2001.

[6] Fredrik Heintz. Robosoc, a system for developing robocup agents for educational use. Master's thesis, Dept. of Computer and Information Science, Linkopings Univ., 2000.

[7] Jelle R. Kok and Nikos A. Vlassis. Using the max-plus algorithm for multiagent decision making in coordination graphs. In *RoboCup 2005: Robot Soccer World Cup iX*, Lecture Notes in Artificial Intelligence, pages 1–12. Springer, 2006.

[8] Henrik Hautop Lund. Robot soccer in education. *Advanced Robotics*, 13(8):737–752, 1999.

[9] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent

systems. *Applied Artificial Intelligence*, 12:233–250, 1998.

[10] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.

[11] S. Papert. Constructionism: A new opportunity for elementary science education. A proposal to the National Science Foundation. Massachusetts Institute of Technology, Media Laboratory, Epistemology and Learning Group., 1986.

[12] Elizabeth Sklar, Simon Parsons, and Peter Stone. Using robocup in university-level computer science education. *ACM Journal on Educational Resources in Computing*, 4(2), 2004.

[13] Frieder Stolzenburg, Oliver Obst, and Jan Murray. Qualitative velocity and ball interception. In *Proceedings of the 25th Annual German Conference on AI: Advances in Artificial Intelligence*, Lecture Notes In Computer Science, pages 283 – 298, 2002.

[14] Peter Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.