Communications and basic coordination of robots in TeamChaos

Carlos E. Agüero, Francisco Martín
Vicente Matellán
Grupo de Robótica (GSyC)
Universidad Rey Juan Carlos
Móstoles (Madrid), España
{caguero,fmartin,vmo}@gsyc.escet.urjc.es

Humberto Martínez Barberá
Dept. Ing. Información y las Comunicaciones
Universidad de Murcia
Murcia, España
humberto@um.es

Abstract

How to coordinate a set of robots is still an open issue. It is a problem made up by several sub-problems. One of them is how to share information among the members of a team without compromising the basic requirements of the application. In this paper we show the communication architecture we have designed to let a team of robots communicate. We present an application of this architecture to implement a simple ball booking protocol to share the information about the ball among the robotic team members in the 4-legged RoboCup. This protocol let us to define strategies for optimizing our tactic during a match.

1 Introduction

The problem of creating a robotic soccer team is a very complex problem. There are several research areas involved (low level locomotion, perception, location, behavior development, communications, etc.). Each team has to solve all the issues in order to build a complete team.

In particular, we are part of the TeamChaos[1] that participates in the 4-legged RoboCup[**?**].

The code is splitted into two main projects: *TeamChaos* and *ChaosManager*. *TeamChaos* contains all code that runs in the robots. *ChaosManager* (figure 1) is a suite of tools for calibrating, debugging, and monitorizing different aspects of the robots and the game.

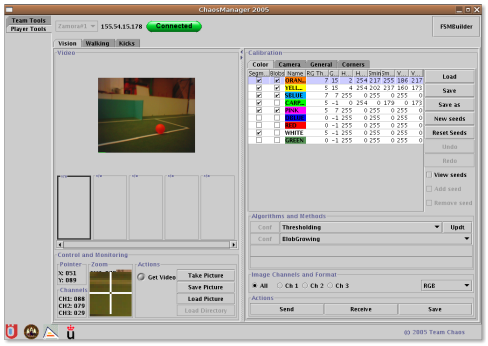

Figure 1: *ChaosManager* suite is used to help in calibration, debug and monitorization the *TeamChaos* code

Communications are an important issue in TeamChaos architecture. They let us use external tools for making laborious tasks more easily. For instance, they let us receive images and data from the robot to debug its behavior, to refine the camera parameters, to reconfigure the robot camera while the robot is running or to teleoperate the robot to check kicks or locomotion using communication between robots and *ChaosManager* too.

Besides being used as support for these tools, communication among robots are needed for sharing information among them and playing better. In particular, ball position and local belief about self location are sent periodically to the rest of team mates.

[1] http://www.aass.oru.se/Agora/RoboCup/

Once this communication architecture is available, it can also be used to coordinate the players during a match. In [**?**] a potential field approximation was developed to solve this problem. In this work, they used dynamic roles assignment to decide the influence areas of each robot. In [**?**], a behavior architecture were able to determine, in a coordinated way, the next action to be taken by each robot. Carnegie Mellon team also uses potential fields in its work [**?**] combined with a global and shared map. Other option is the work [**?**], that uses utility concept to choose the most appropriate role for each robot.

During RoboCup 2005, we implemented a Basic Ball Booking Protocol B^3P to resolve the problem of two or more robots detect the ball and try to control it. This protocol uses a coordinated mechanism to guarantee that *only one, and just one* robot goes to the ball and controls it.

In section 2 our Team-Chaos software architecture is presented. The communication mechanism is analyzed in section 3. Section 4 describes the simple ball booking mechanism design and implementation. The experiments to validate this mechanism are presented in section 5. Finally conclusions are discussed in the last section.

2 TeamChaos software architecture

TeamChaos architecture is represented in Figure 2.The lower layer commander provides an abstract interface to the sensor-motoric functionalities of the robot. The middle layer maintains a consistent representation of the space around the robot (PAM or "Perceptual Anchoring Module"). It is composed by a set of tactical behaviors (HBM or "Hierarchical Behavior Module"). The higher layer maintains a global map of the field (the GM, or "Global Map"), and makes the strategic decisions based on the current situation using an HFSM (Hierarchical Finite State Machine). Finally, radio communication is used to exchange position and coordination information with other robots via the TCM, or "Team Communication Module".

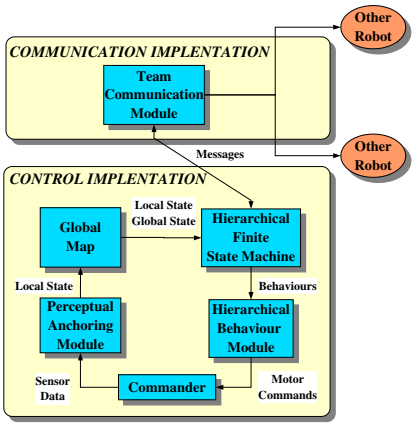

Figure 2: Overview of the TeamChaos architecture

The code running in each robot is organized in Open-R[2] objects. Each object is mono-thread. Architecture shown figure 2 is divided into three different Open-R objects: *ORRobot*, *ORTcm* and *ORGCtrl*.

ORRobot is the main object and his goals are getting local perceptions from its sensors, getting located inside the field, and generating motion commands according to a collection of behaviors.

ORTcm is the communication manager and must fulfill tasks for sending and receiving data.

ORGCtrl is an object entrusted to manage all instructions sent by the *GameController*, which is an electronic referee during the match.

Next section describes the *ORTcm*, which is the relevant module for the implementation of B^3P. Details about this architecture can be found in [**?**].

3 Team-Chaos communication architecture

ORTcm is an Open-R object which controls the radio communications between an AIBO robot and any external entities (other AIBO

[2] http://openr.aibo.com/

robots, remote computers, etc). It is a "communication center" that is used by modules when they want to communicate with other ones located in other robot.

The communication protocol developed is stateless, so there are not any connection establishment process, neither for termination. Also, transmissions are not reliable. Due to real time needs, communication protocol does not use any ACK's or retransmissions. It is better to lost some information than retransmits several times and receive old information.

OPEN-R offers both TCP/IP and UDP support but we have chosen the UDP alternative for make lighter and real time communications. At this moment, TCM opens four UDP sockets, three of them for *ChaosManager* communications, and the other one for communications among robots.

An *ExternalMessage* data structure has been developed as the exchange unit between entities. As we can see in figure 3 it has a set of fields describing the data source and destiny, both the source entity (AIBO or computer), the TeamChaos module that has sent it, data length, data transfered in the message, etc.

Figure 3: External message package with its relevant fields

Each robot has an unique IP and it is associated to his robot identifier in file *TCM.ini*. When someone sends a message to the identifier of a robot, the consequence will be an unicast message towards the IP joined to this identifier. We also allow broadcast transmissions. This feature is useful for sending debug information that is analyzed and showed by our external debugging tools (ChaosManager) and for sharing information among the members of the team.

A class called *UDPConn* has been created for implementing the TCM. This class provides an abstraction layer over the OPEN-R IP stack, showing to the developer an easy interface for using typical socket calls. Also, a *TcmClient* class has been developed for hiding all details of the communication inter-object mechanism typical in OPEN-R.

One interesting detail of the *Team Communication module* is the capacity of sending any kind of data. TCM forces the module, that wants to send/receive an specific data type, to offer its own functions to serialize and unserialize data. Maintaining this constraint, TCM solves the rest of the communication tasks.

The communication module also establishes a callback mechanism that allows a module to register with an specific data type. When this data type arrives to the robot, the callback function previously registered is invoked.

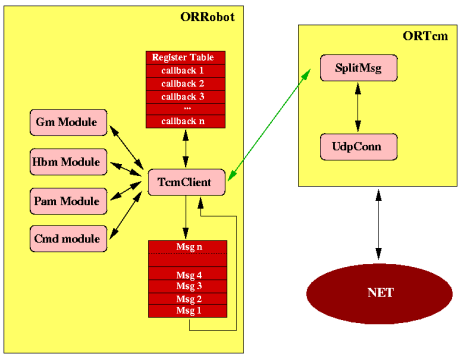

Figure 4: Communication diagram between modules and Tcm

TCM also resolves the problem of sending/receiving messages bigger than a UDP datagram. It cuts into pieces larger messages and it composes again at destination. This is useful for sending big configuration files or images that do not fit into a single UDP datagram.

Finally, a class called TcmClient has been developed with the goal of hiding OPEN-R implementation details to modules that want to communicate with the ORTcm object. As we can see in figure 4, TcmClient acts as a proxy storing all messages from the modules. It delivers messages to ORTcm when communication module is ready to attend a new request. So, TcmClient works regulating the flux of messages between modules and

TCM simplifying the communication process to them.

4 Ball booking protocol

Once the communication architecture had been implemented, we focused in solving the problem of coordinating the robots that try to control the ball. The problem arises when two or more robots detect the ball. Every robot wants to control it. If two or more robot reach the ball position at same time, the ball control was ineffective, making impossible to play in the right way.

B^3P basically consists in maintaining a local booking state about the ball. The ball can be requested by a robot, which announces this booking to other robots using the communication mechanism previously described.

Other robots update this information, and locally decides if they can book the ball or not. A robot decides its next action depending on this local information about the ball booking state.

This mechanism must accomplish the next requirements:

1. If a robot is nearer from the ball than the one which has booked the ball, the booking state must change and the nearer robot must book the ball and try to control it.

2. It is not permitted that no robot tries to control the ball.The mechanism must quickly recover from situations in which the robot which has booked the ball crash or is penalized.

3. Robots at same distance from ball should not be alternatively booking the ball. In order to solve it some distance thresholds has been used.

4. *Only one robot, and just one* of the robot which has detected the ball, must obtain the ball control. The other robots must go to optimal positions in the field, but they never should try to obtain the ball control while one of its mates has the ball booked.

The robot maintains local information about the ball booking state. This information is maintained as four different variables:

- **Booked ball**. Becomes `true` when any robot has booked the ball.
- **Robot booker**. It contains the `robot Id` which has booked the ball.
- **Distance**. Distance to the ball from the robot which has booked the ball.
- **Timestamp**. Local timestamp when the robot has booked (or re-booked) the ball.

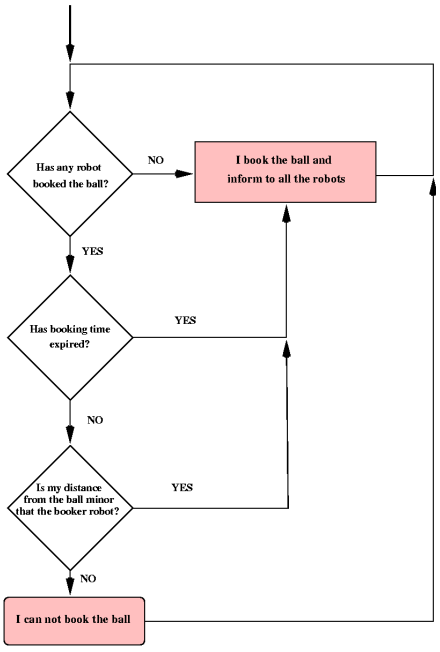

Figure 5: Basic Ball booking protocol

In Figure 5, the booking mechanism is shown. When a robot knows the ball position, it always tries to book it. The process is the next:

- The robot checks if any robot has previously booked the ball. If the ball has

not been already booked, it books the ball and informs of this decision to its team mates.

- If the ball is currently booked by another mate, the robot checks the local booking timestamp.
- If the robot which has the ball booked has not renewed the book recently, the ball is supposed to be free and able to be booked by another robot.
- If the ball is booked and this book is renewed periodically, the robot checks if its own position with respect the ball is better than the currently booker robot one. If its position is better, it books the ball and informs of this decision to its mates.
- When all these conditions have been tested and the robot has not booked the ball, the robot must do anything, excepts trying to control the ball.

5 Experiments

As with most complete systems designed to perform a task, it is difficult to produce quantitative results. Before using this booking method in a real match, we tested the system extensively in our laboratory. These tests can be downloaded from `http://gsyc.es/robotics/sbbp.mpg`. In this video, only a robot at same time goes to the ball, and the others perform another task. In this case and for clarity, they remain still.

Figures 6-10 show an example of this protocol in action. In figure 6, robot 1 and robot 3 have successfully localized the ball. They must decide which of them has to go to the ball. In figure 7, robot 1 has booked the ball and goes to the ball. The other robots have detected the ball, but their main task is remaining still while they have not booked the ball. In figure 8, the ball is pushed by an external agent. Then, robot 1 loses the ball (figure 9), and robot 2, which has previously detected the ball but was not moving, books the ball and goes for it (figure 10).

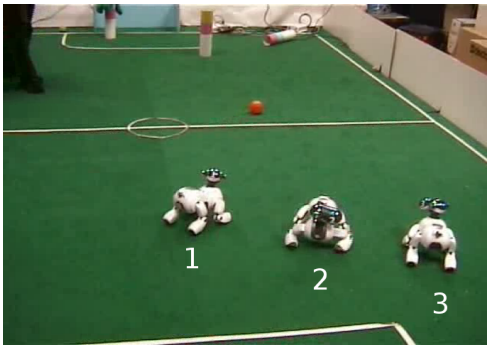

Figure 6: The ball is seen by robot 1 and 3. One of them must go to the ball and they start the book process

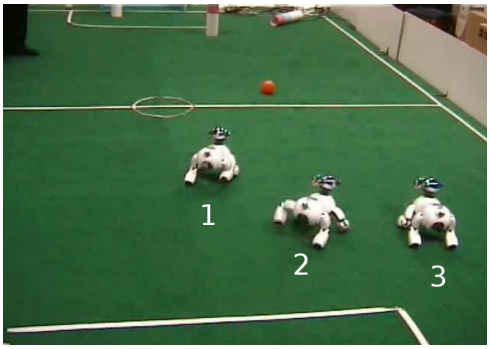

Figure 7: Only the robots 1 books the ball and goes to it. Robots 2 and 3 remain still

The tests done to validate this protocol show how the robots share the ball information for booking the ball in the right way. When a robot, which has booked the ball, goes to the ball and it is switched off, instantaneously another robot books the ball and starts moving to the ball. All the requisites presented in Section 4 are satisfied.

In the First Spanish RoboCup 4-Legged Open[3](FSR4LO) we tested the Simple Ball Booking Protocol. The robot agglomeration around the ball was greatly reduced and the robots covered the field in a more efficiently way. Due to the fact that the three teams that entered the competition this year are part

[3]http://www.rvg.ua.es/SpanishOpen05/

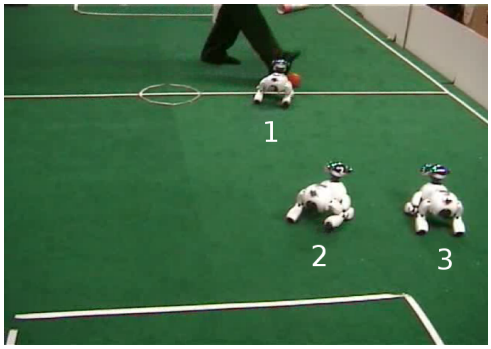

Figure 8: The ball is lost by robot 1 due to an external action

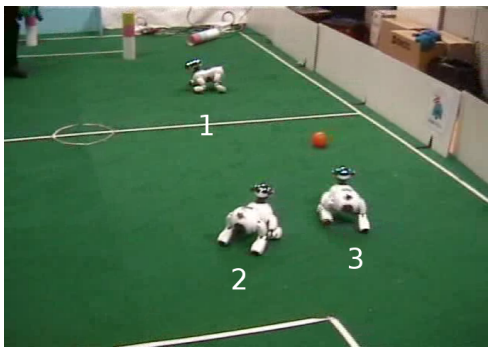

Figure 9: Robots 2 and 3 detect the robot 1 has lost the ball and book the ball. Robot 3 books the ball and goes to the ball

of the TeamChaos, the code was basically the same. The major difference among them was the use of B^3P in the URJC team. This team won the competition and did not lose any match. In the next table we the scores in the open are showed. First column corresponds to the team which used B^3P and the second column corresponds to the other teams which did not use B^3P.

6 Conclusions and Further Work

The use of communication among robots greatly enhances the players performance during a match.

In this paper we have shown the communication architecture developed in TeamChaos 4-legged team. Using this architecture, we have designed and developed a Basic Ball Booking Protocol, called B^3P, that allows sharing responsibilities among robots, without give up their functionality and without producing negative effects in adverse situations during the match.

Also, we have analyzed the results obtained, showing that this protocol really works and the team performance enhancement is improved.

Next actions are focused on developing algorithms that calculate the optimal robot positions in the field when they do not book the ball. These improvements will let us to design a team strategy which will be useful in next competitions.

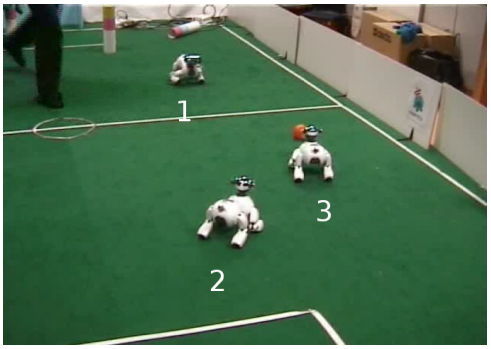

Figure 10: Robot 3 controls the ball

	B^3P	Non B^3P
1st match	2	0
2nd match	2	0
3th match	1	0

Table 1: Results of First Spanish 4-Legged Open

7 Acknowledgement

The authors would like to thank other members of TeamChaos and Robotics Lab, specially Antonio Pineda and Víctor Hidalgo for their useful help in this work.

This research has been partially sponsored by grants No. S-0505/DPI/0176 by Community of Madrid and No. DPI2004-07993-C03-01, DPI2004-07993-C03-02 by Spanish Ministry of Education corresponding to RoboCity and Acrace projects respectively.

References

[1] Humberto Martínez Barberá, Vicente Matellán Olivera, Miguel Ángel Cazorla Quevedo, Francisco Martín Rico, Carlos Agúero Durán, Víctor Manuel Gómez, Gómez, and David Herrero-Perez. Team chaos 2005 - team report. Technical report, TeamChaos 4-legged team, 2005.

[2] Brian Gerkey and Maja Mataric. On role allocation in robocup. In *RoboCup 2003: Robot Soccer World Cup VII, 2004*. Springer-Verlag, 2004.

[3] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *ICJAI-95 - Workshop on Entertainment and AI/ALIFE*, 1995.

[4] Scott Lenser, James Bruce, and Manuela Veloso. A modular hierarchical behavior-based architecture. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[5] Noriaki Mitsunaga, Taku Izumi, and Minoru Asada. Cooperative behavior based on a subjective map with shared information in a dynamic environment. *International Conference on Intelligent Robots and Systems*, pages 291–296, 2003.

[6] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty, 1986.

[7] Ashley Stroupe, Martin C. Martin, and Tucker Balch. Distributed sensor fusion for object position estimation by multi-robot systems. In *IEEE International Conference on Robotics and Automation, May, 2001*. IEEE, May 2001.

[8] Douglas Vail and Manuela Veloso. Dynamic multi-robot coordination. In *Multi-Robot Systems*. Kluwer, 2003.

[9] J. Vallejos, P. amd Ruiz-del-Solar and A. Duvost. Cooperative strategy using dynamic role assignment and potential fields path planning. In *1st IEEE Latin American Robotics Symposium, LARS2004. Mexico City. México*, 2004.