VIRTUAL ROBOT - ADAPTIVE RESSOURCE-MANAGEMENT IN ROBOT TEAMS

Jens Ziegler [*,1] **Ingo Dahm** [**,1] **Mathias Hlsbusch** [*]
Jochen Kerdels [*]

[*] *Department of Computer Science, University of Dortmund,*
[**] *Computer Engineering Institute, University of Dortmund,*

D-44221 Dortmund, Germany

Abstract: In this article, the design of a global control architecture for teams of soccer playing robots is presented. Therefore, the metaphor of a "virtual robot" is introduced. This meta-computing based concept of a virtual robot - a collection of a dynamically varying number of robots - is used in order to offer a simple method to share sensor information and processing power efficiently in a team of autonomous walking robots, additionally opening the possibility of "call-for-action" requests. Sensor fusion allows for the calculation of a coherent global world model which in turn will be processed locally to get optimal action sequences of the robots in the team. Coordination of the robots is achieved by local schedulers which use the global world model as a basis for action selection.

Keywords: Control, Sensor Integration, Legged Robots, Multiple Vehicle Systems, Networks of Autonomous Vehicles

1. INTRODUCTION

Since 1999, the SONY Legged League is an official RoboCup League and the first one with walking robots [2] . Each team consists of several four legged robots – SONY ERS-7, ERS-210 or ERS-210 supercore. The specific situation in the Legged League results in a special behavior-control:

1) It is not allowed to add nor change hardware components. Only one type of robot is allowed on the playing field. Thus, the robots of one team are almost identical. (Participants, 2002). It is not possible the scale processing-power or memory capacity to the users needs. Therefore, the ressources are under massive load. Even the lately introduced super-core didn't solve this problem. A teame-wide ressource sharing as done in meta-computer networks is a capable solution to handle jobs that are to big for one computing unit (Hamscher *et al.*, 2000; Schwiegelshohn and Yahyapour, 2000).

2) The ability to walk influences alle modules that depend on locomotion directly or indirectly. Exemplary, a odometry-data-based navigation (Chenavier and Crowley, 1992; Borenstein *et al.*, 1997), has to deal with massive problems like ¡schlupf¿, ¡kippeln¿, and ¡verhakeln¿. Even Localization does mainly depend on vision-systems quality (Fox *et al.*, 1999; Fox, 1998; Utz

[1] Partially supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Ba 1024/11-1
[2] http://www.robocup.org

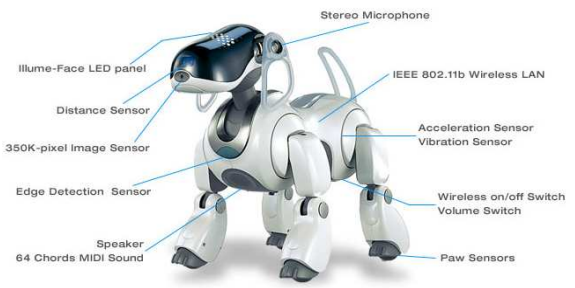

Fig. 1. Sony's ERS-7. This 20 degree-of-freedom, fully autonomous four-legged robot is used in RoboCup robot soccer games.

et al., 2002). A low reliabilty of the observed data worsens the accuracy of localization. The consideration of more than one robots observation does improve the world models quality significantly (Dahm and Ziegler, 2002; Stroupe *et al.*, 2001; Stroupe *et al.*, 2000).

3) Participants of the League are encouraged to freely distribute their software-code. This makes sense as the robot components are identical. Therefore, the participating teams will show a comparable performance in the next future as good methods and skills of winning teams can be captured (e.g. walking engine (Uther *et al.*, 2002)). In future, the individual skills of the robots can be assumed to be similar/equal. The use of cooperating methods and algorithms is traded as the next main performance enhancement. **REF** In the SONY League, we expect this to be a way to perform coordinated team-actions.

All these problems can be solved by a global ressourcemanagement and a job Scheduler. The situation is highly comparable to a Metacomputing project, whereas each robot is a ressource of the grid (C.Bitten and Yahyapour, 2000). Moreover, the Real-Time constraint is a hard side-constraint in Robot Soccer. Robots can demand ressources and provide other ressources and information at the same time. Scheduling strategies and ressource management rules must reflect those constraints.

2. THE "VIRTUAL ROBOT" METAPHOR

A collection of a dynamically varying number of robots is called a "virtual robot" (VR). A VR contains all available ressources and information of all individual robots [3] and a special management architecture allows to access all elements of the VR as if they were parts of a single robot.

To use this metaphor has some important advantages: (i) it allows to control all robots coherently. This is in contrast to current techniques, which achieve a purposeful team behavior only with mutually coordinated actions. (ii) A ressource management system can use all capabilities of the robots efficiently by calculating an optimal load balancing. This scheduler can resort to well known algorithms in the area of meta-comptuting. (iii) It is user independent, i.e. it is not necessary for the system designer to assign a specific robot to specific tasks in advance [4]. (iv) The concept does not rely on a fixed architecture of the VR, instead is designed to handle a variable number of heterogeneous devices.

The main problem of controlling a VR is the identification and allocation of available ressources, the splitting of complex, high-level tasks into different smaller problems and the subsequent assignment of these small tasks to appropriate ressources.

2.1 Ressources in a team of robots

All available sensors, actuators, and processors of all robots that are part of the VR will be considered as ressources. A VR consisting of several different robots has a multitude of different sensors, processors and actuators. If the configuration of the VR changes, the composition of ressources changes, too. Every change (e.g. addition/deletion of robots due to decisions of a referee) therefore has to communicated by its causer to the existing VR via a well defined registration/check out protocoll.

2.2 Worldmodel

All local information is integrated into one global world model by sensor fusion techniques. The reliability of local informations can be enhanced by cross-validation. This reduces the impact of erroneous sensor data.

The world model reflects the VR's view of the actual situation and is the basis for its action selection mechanism.

2.3 Atomic actions - Tasks

In contrast to other grid-computing applications, where the character of distributed processes is potentially infinite and depends on the application itself, the set of different tasks that have to be solved in robot soccer in order to get a coherent

[3] A VR is not necessarily restricted to consist only of robots of a single type as in the four-legged league. It is possible to integrate different robot architecures into a single VR.

[4] This role mechanism is commonly used in team robotics.

team behavior can be limited. This has consequences for the scheduling algorithm.

There are three different types of tasks which are specific for robot soccer applications:

2.3.1. Sensor tasks Sensor tasks are scheduled to a robot if the VR is in need of information that can be obtained by the sensory system of that robot. An example for a sensor task is to examine the position of the ball, to calculate the position, and send the information along with a validity value.

2.3.2. Processing tasks A processing task requires a computation. If the VR consists of robots with different CPUs, a complex computation can be scheduled to a robot with a more powerful CPU. An example might be the color segmentation of a high-resolution color image which can not be done in realtime by other robots.

2.3.3. Do this: Action tasks Action tasks are the main element in which the concept of a VR differs from an ordinary grid-computer. The possibility to *do something* allows the VR to directly manipulate its environment, which in turn has an impact on the dynamics as a whole. The fact that a VR consists of several independent autonomous robots entails the problem of *how to do something*, because the redundancy of actuators makes it possible to achieve a certain aim in different ways. A human can pick up a ball with its right or with its left hand. Which hand to use depends on what the further plans with the ball are. If it just wants to look at it, the choice of the hand does not really matter. If the goal is to throw the ball far away, then the human should pick it up with its stronger arm, in order to reach the goal sufficiently.

In this context, an action is defined as a small sequence of directed movements of a robot. A typical example is a movement to a certain position given in 2D-world coordinates (x, y). This coarse grained definition of actions allows to compose complex team behaviors without goinig ttoo much into the details of robots hardware programming. In section 4, more examples of actions typical for robot soccer are presented.

3. SCHEDULING

Having described the different tasks, it is now the job of the scheduling mechanisms, to coordinate, which physical robot of the VR has to perform which task at a given time. Using a simple client/server approach as the distribution mechanism would lead to complicated error handling procedures due to possible communication failures between the robots.Therefore we designed a scheduling mechanism that just needs some broadcasted data and which lets every robot decide for itself locally, which task to perform.

In order to achieve a coherent behavior of the whole team, we present the concept of "task rating".

3.1 Task Rating

Basically, the task rating is a function f_r mapping the worldmodel w (the robots view of the current situation) and a specific task t to $\mathbb{R}^+$:

$$f_r(t, w) \longrightarrow \mathbb{R}^+. \quad (1)$$

The return value of this function represents how "good" a robot is able to perform the specified task. The function f_r is evaluated locally on every robot (thus depending on robot specific capabilbities) and the return value is zero, if the robot is by no means able to solve that task in the actual situation.

For example, if the task is "$move(x, y)$", robots that cannot get there, return $f(move(x, y), w) = 0$ and a robot that can get to (x, y), but its own position is far away from (x, y) returns a quite small value of f. A robot that is already close to (x, y), returns a high value of f.

If the list of all tasks (sensor, processing, and action tasks) is limited and known, every robot can calculate the value of f_r qfor every task in a certain situation. After a broadcasting of this task rating, we assume that every robot has a complete list of task ratings of all the other members of the VR.

3.2 Configurations

A *configuration* c is a collection of tasks or task-classes [5] that are reasonably to be performed at the same time by different robots in order to get a coordinated and purposeful team behavior. The number of tasks in a configuration c is in $[1, n]$, with n indicating the number of robots in a team. Obviously, it has to be specified which tasks belong to a which task-class.

Further on, every task or task-class in a configuration c has a special value $w_c(task)$ that presents how important the special task is to reach the VR's aim.

[5] A task class is a symbol for tasks whose realization depend on which robot of the VR executes a task of that class. An example is the *kick()*-task that can be realized differently (left foreleg, right hindleg, head, etc) depending on the actual situation of the game. The task rating of a task class is the maximum rating of all tasks within the task class.

Finally, every configuration $c_i \in C$, where C is the set of all configurations and $\mid C \mid = m$, has a value uc_i that represents how important the whole configurationc_i is to reach the VR's aim. Every robot in the system has the same set C of configurations.

Every c_i defines a vector $\boldsymbol{v}_{c_i}(C)$ of values which modify the $u_{c_j}, \quad j \neq i$, values of other configurations. This makes it possible to change the weights of configurations such that in subsequent steps evaluations are biased depending on the actual decision.

Additionally, the whole system is prevented from being trapped in a single configuration c_i if the weight u_{c_j} of another configuration is incremented by repeated choices of c_i.

3.3 The scheduling algorithm

The scheduling algorithm realizes a function that uses all the data described above to calculate a matching from robots and tasks with maximum score. Because there are $n!$ permutations to assign the tasks in c_i to the n robots of the team, the s_i values have to be calculated for every permutation.

The basic score $s(c_i)$ of a configuration c_i is defined as

$$s(c_i) = max[u_{c_i} \sum_{t \in c_i} f_r(w,t) w_{c_i}(t) | \, r = 1, ..., n] \quad (2)$$

We define $\boldsymbol{S}$ as $\boldsymbol{S} = (s_{c_1}, \ldots, s_{c_m})^T$ with

The scores $s_i \in \boldsymbol{S}$ can additionally be modified by multiplication with one ore more vectors x of weights. An example is the weighting of configurations supporting offensive play. The respective elements of x will have values greater than one. The final scores are computed with

$$\boldsymbol{S'} = \boldsymbol{S} \cdot \boldsymbol{v}_{c_i} \cdot (x_1, \ldots, x_m), \quad (3)$$

where c_i is the last selected configuration. The configuration with the maximum s-value will be selected and the tasks of this configuration are assigned to the robots according to the permutation which yielded the maximum $s(c_i)$ value in eq. (2). In Fig. 2, a schematic view of the scheduling mechanism is shown.

3.4 Broadcast of results

We have seen that all data that needs to be exchanged between the robots is the task-rating for all tasks. We expect this to be a quite small amount of data (like one UDP-packet), so it is no problem to broadcast this data often. Each task-rating-packet has a time-stamp, which enables the

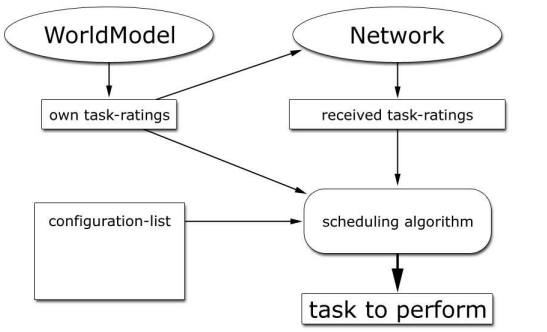

Fig. 2. Schematic view of the scheduling machanism.

scheduler to give a lower weight to older task-ratings. This makes it possible to handle communication failures or synchronisation problems, because the important tasks are implicitly scheduled to robots, which were known to be reachable a short time ago: configurations get high scores if high task weights $w(t)$ coincide with high task ratings $f_r(w,t)$. It is unlikely that important tasks are scheduled to either robots with small task ratings or robots whose ratings are old, because of the ageing of task ratings.

This means that if some robots are unavailable, the remaining robots perform the high weighted tasks in the according configuration. In case of a complete communication breakdown, each robot just executes the task with the highest $f_r(w,t)$ value.

There is a wide range of possibilities to extend this scheduling mechanism.

3.4.0.1. Learning Scheduler It is possible in principle to learn *good* configurations at runtime. During a robocup football game, for example, the $u(c)$ values of the past configurations can be modified (increased), if a goal was scored. For coming scheduling events, these configurations are slightly preferred. The $u(c)$ values can be decreased, if a series of configurations led to a oppnonent goal.

3.4.0.2. Global Tendencies You can express global tendencies with weight vectors. The football example again: You can favor more offensive or defensive team play by increasing or decreasing the $x(c)$-values which modify the scores s_c of those configurations which supports offensive playing or defensive playing respectively.

3.4.0.3. Intermediate Objectives If the VR's aim is too complex to be defined by one big configuration-list, it is a good solution to split

the complex aim into smaller intermediate objectives. For each of this intermediate objectives a configuration-list can be designed separately and by a worldmodel-dependend rating of these objectives it is possible to weight the $x(c)$-values which modify the scores s_c of the appropriate configurations, so that the intermediate objectives can be processed consecutively to finally fulfill the complex aim of the VR.

4. APPLICATION

A simple example may clarify the scheduling mechanism. It assumes a robot soccer application with four autonomous legged robots. The complete list of task is

t_1: go forward with ball
t_2: go forward
t_3: withdraw
t_4: stay in goal
t_5: score a goal

The list of configurations C is defined as

$$c_1 = \left\{ \begin{array}{ll} \text{go forward with ball} & , 5 \\ \text{go forward} & , 3 \\ \text{withdraw} & , 0.3 \\ \text{stay in goal} & , 0.1 \end{array} \right\}, u_{c_1} = 1.0$$

$$c_2 = \left\{ \begin{array}{ll} \text{score a goal} & , 25 \\ \text{go forward} & , 1 \\ \text{withdraw} & , 2 \\ \text{stay in goal} & , 0.5 \end{array} \right\}, u_{c_2} = 1.0. \quad (4)$$

Configuration c_1 suggests a team behavior that pushes the ball forward, along with a supporting robot, whereas configuration c_2 favors a kick to the goal, while, at the same time, another robot should withdraw in order to block a counter of the opponent in case of a failing kick. The task-ratings in Tab. 1 have been calculated by the robots, assuming that robot 2 has the ball and robot 4 is the goalkeeper. Robot 1 and 3 are somewhere in the middle of the field.

Table 1. Task ratings of all five tasks computed by all four robots in the team.

	Robot 1	Robot 2	Robot 3	Robot 4
t_1	0	10	0	0
t_2	9	9	4	3
t_3	3	3	6	6
t_4	0	0	0	10
t_5	0	2	0	0

According to eq. (2), the maximum basic score for configuration c_1 is

$$s(c_1) = 10 \cdot 5 + 9 \cdot 3 + 6 \cdot 0.3 + 10 \cdot 0.1 = 79.8 \quad (5)$$

for the following mapping

$$\begin{aligned} t_1 &\longrightarrow \text{robot 2} \\ t_2 &\longrightarrow \text{robot 1} \\ t_3 &\longrightarrow \text{robot 3} \\ t_4 &\longrightarrow \text{robot 4.} \end{aligned} \quad (6)$$

Configuration c_2 gets a basic score of

$$s(c_2) = 2 \cdot 25 + 9 \cdot 1 + 6 \cdot 2 + 10 \cdot 0.5 = 76.0 \quad (7)$$

for the same mapping. This results in a decision for configuration c_1. Since every robot has made the calculations, the scheduling of the tasks to the robots is clear.

After some time, the task ratings change (see Tab. 2).

Table 2. New task ratings of all five tasks computed by all four robots in the team.

	Robot 1	Robot 2	Robot 3	Robot 4
t_1	0	10	0	0
t_2	9	9	4	3
t_3	4	3	4	6
t_4	0	0	0	8
t_5	0	5	0	0

The configurations now get basic scores of

$$\begin{aligned} s_{c_1} &= 10 \cdot 5 + 9 \cdot 3 + 4 \cdot 0.3 + 10 \cdot 0.1 = 79.2 \\ s_{c_2} &= 5 \cdot 25 + 9 \cdot 1 + 6 \cdot 2 + 0 \cdot 0.5 = 146.0 \end{aligned} \quad (8)$$

for the same mapping (see eq. (6)). Configuration c_2 gets the highest score, so that the team behavior changes from c_1 (forward motion) to c_2 (attempt to score a goal). In this special case, another mapping of tasks to robots is possible ($t_1 \rightarrow$ robot $2, t_2 \rightarrow$ robot $1, t_3 \rightarrow$ robot $4, t_4 \rightarrow$ robot 3 gets the same score. Due to the fact that identical schedulers are executed on the robots, the team will come to the same conclusion, so no extra handling of this situations has to be implemented.

5. CONCLUSION

ACKNOWLEDGEMENT

The authors wish to thank the students involved in this project: **Claudius Rink, Andreas Rossbacher, Frank Rossmann, Bernd Schmidt, Pascal, Jrn Hamerla, Manuel, Hyung-Won Koh, Damien, Christop Richter, Carsten Schuman, Norbert**. We additionally thank our sponsors Microsoft Corp. and Lachmann & Rink for financial support.

REFERENCES

Borenstein, J., H. Everett, L. Feng and D. Wehe (1997). Mobile robot positioning: Sensors

and techniques. *Journal of Robotic Systems* **14**(4), 231–249.

C.Bitten, J. Gehring, U. Schwiegelshohn and R. Yahyapour (2000). The NRW-Metacomputer. Building Blocks for A Worldwide Computational Grid. In: *International Parallel and Distributed Processing Symposium 2000*.

Chenavier, F. and J. Crowley (1992). Position estimation for a mobile robot using vision and odometry. In: *Proceedings of the IEEE Int. Conference on Robotics and Automation (ICRA92)*. IEEE Press,Piscataway, NJ. pp. 2588–2593.

Dahm, Ingo and Jens Ziegler (2002). Using artificial neural networks to construct a metamodel for the evolution of gait patterns of four-legged walking robots. In: *Proc. Fifth Int'l Conf. Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR 2002)* (P. Bidaud and F. Ben Amar, Eds.). Professional Engineering Publ.. Bury St. Edmunds, U.K.. pp. 825–832.

Fox, D. (1998). Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation. PhD thesis. Institute of Computer Science III, University of Bonn.

Fox, D., W. Burgard, F. Dellaert and S. Thrun (1999). Monte carlo localization: Efficient position estimation for mobile robots. In: *Proceedings of AAAI'99*. pp. 343–349.

Hamscher, V., U. Schwiegelshohn, A. Streit and R. Yahyapour (2000). Evaluation of Job-Scheduling Strategies for Grid Computing. *Lecture Notes in Computer Science* **1971**, 191–202.

Participants, Sony Legged League (2002). Sony four legged robot football league rule book. Technical report. Sony Legged League.

Schwiegelshohn, U. and R. Yahyapour (2000). Fairness in parallel job scheduling.

Stroupe, A., M. Martin and T. Balch (2000). Merging probabilistic observations for mobile distributed sensing.

Stroupe, A., M. Martin and T. Balch (2001). Distributed sensor fusion by for object position estimation by multi-robot systems. In: *Proceedings of the IEEE/RAS International Conference on Robotics and Automation (ICRA)*. IEEE Press, Piscataway, NJ.

Uther, W., S. Lenser, J. Bruce, M. Hock and M. Veloso (2002). Cm-pack'01: Fast legged robot walking, robust localization, and team behaviors. In: *RoboCup-2001: The Fifth RoboCup Competitions and Conferences* (A. Birk, S. Coradeschi and S. Tadokoro, Eds.). Springer Verlag, Berlin.

Utz, H., A. Neubeck, G. Mayer and G. Kraetschmar (2002). Improving vision-based self-localization. In: *Proceedings of the 6th international RoboCup Symposium* (G. A. Kaminka, P. U. Lima and R. Rojas, Eds.). pp. 17–32.