final version:
Advanced Robotics, Volume 14, Number 1, pp 27 - 36, VSP International Science Publishers, 2000

On-board Control in the RoboCup Small Robots League

Andreas Birk and Holger Kenn and Thomas Walle
Artificial Intelligence Laboratory
Vrije Universiteit Brussel
{cyrano,kenn,thomas}@arti.vub.ac.be

Abstract

We present the most recent version of our RoboCube system, a special robot-controller hand-tailored for players in the small robots league. The RoboCube is conceptualized to implement players with as many on-board features as possible in an extremely flexible way. For this purpose, the RoboCube provides significant computation power and memory as well as a multitude of I/O-interfaces within the space-constraints. As it facilitates the use of many sensors and effectors, including their on-board processing, the RoboCube allows to explore a large space of different robots and team set-ups.

1 Introduction

The investigation of on-board control within RoboCup is an extremely interesting scientific issue for two major reasons. First, it is directly related to one of the greatest intellectual challenges of our time, namely the quest for a constructive understanding of intelligence. This aim to foster AI and related disciplines has been a major goal of RoboCup right from its beginning [KAK$^+$97, KTS$^+$97].

Second, the exploitation of on-board features is also related to an important technological development, namely the emerging field of *autonomous systems* as networked embedded devices, i.e., computerized systems with sensors and effectors as well as standalone and communication capabilities. Information Technology has significantly influenced our daily lives in the last decades and it will continue to do so. But the potential of the increasing dominance of computer and networking facilities has its limits as IT is restricted to the transport, provision, and processing of mere information. In addition, there is the enormous possibility and the actual need to free devices, which engage in physical perception and manipulation, from explicit and permanent human supervision. For example e-commerce can only rise to its full potential if a multitude of autonomous systems collaborates in an automated ordering, delivery, and reception of goods.

The two major properties of the small robots league, global sensing and severe size restrictions, discourage to some extent the important investigation of on-board control. But they also have positive effects. First of all, the global sensing eases quite some perception problems, allowing to focus on other important scientific issues, especially team behavior. An indication for this hypothesis is the apparent difference in team-skills between the small robots league and the midsize league, where global sensing is banned. In RoboCup'98, teams in the small robots league managed to demonstrate some real team behaviors, like e.g. passing of the ball. The teams in the midsize league seemed not yet to be that far.

The size restrictions as a second point also have a beneficial aspect for the investigation of team-behavior. The play-field of a ping-pong-table can easily be allocated in a standard academic environment, facilitating games throughout the year. It is in contrast difficult to embed a regular field of the midsize league into an academic environment, thus the possibilities for continuous research on the complete team are here limited. The severe size restriction of the small robots league has another advantage. These robots can be much cheaper as costs of electro-mechanical parts significantly increase with size. Therefore, it is more feasible to build even two teams and to play real games throughout the year, plus to include the team(s) in educational activities.

The rest of this article is structured as follows. In section 2, we introduce a classification of team approaches based on possible basic components for players. With the help of this background, we motivate the importance of fostering on-board features for the players. The third section deals with our RoboCube system, a special robot-controller hand-tailored for players in the small robots league. RoboCube's basic properties, its design, and its implementation are described. It is shown how RoboCube is conceptualized to implement players with as many on-board features as possible in an extremely flexible way. Section 4 presents software aspects related to the RoboCube. First, its BIOS is introduced. Furthermore, it is demonstrated with the example of path-planning that highlevel control aspects can actually be implemented on board of the RoboCube. Section 5 concludes the article.

2 On-Board Features and their Motivations

2.1 Classification of Team-Approaches

An informal classification of approaches is proposed in [Asa98]. There, a first distinction is based on the type of vision (local, global or combined) and the number of CPUs (one or multi). It is also mentioned that in the case of multiple CPUs a difference between systems with and without explicit communication between players can be made. Though this scheme is useful, it is still a first, quite rough classification. Therefore, we propose here to make finer distinctions based on a set of crucial components for the players.

In general, a RoboCup team consists of a (possibly empty) set of host-computers and off-board sensors, and a non-empty set of players, each of which consist of a combination of the following components:

1. minimal components
   (a) mobile platform
   (b) energy supply
   (c) communication module
2. optional components
   (a) computation power
   (b) shooting-mechanism and other effectors

(c) basic sensors

(d) vision hardware

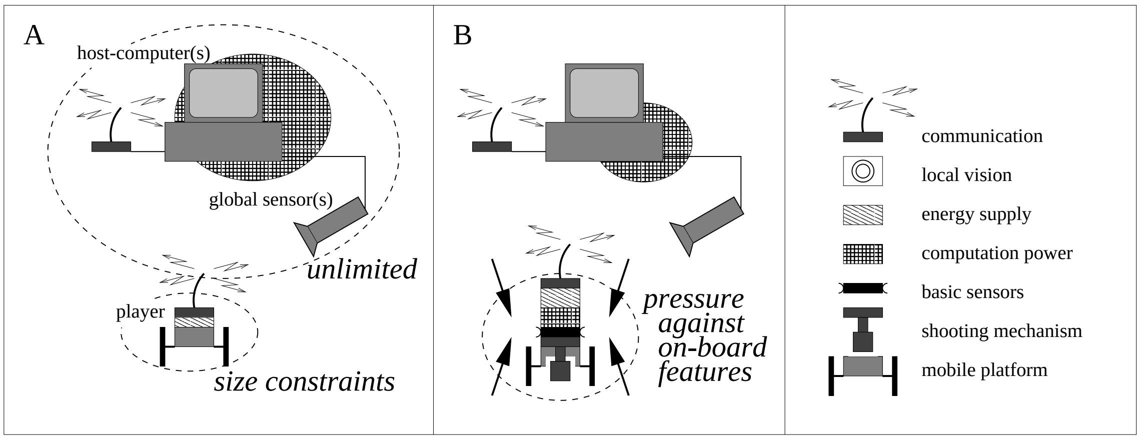


Figure 1: There are several basic components which can be, except the minimal ones, freely combined to form a player. Situation A shows the most simple type of player, a radio-controlled mobile platform. Situation B shows a more elaborated player.

Note, that the most simple type of player, consisting of only minimal components, is hardly a robot. It is more like a "string-puppet" in form of a radio-controlled mobile platform without even any on-board sensors or computation power (though it could well be possible that this type of device has an on-board micro-controller for handling the communication protocol and the pulse-width-modulation of the drive motors). The actual control of this type of players completely takes place on the off-board host(s).

Based on this minimal type of player, the optional components can be freely combined and added. In doing so, there is a trade-off between

1. on-board sensor/motor components,
2. on-board computation power, and
3. communication bandwidth.

A player can for example be built without any on-board computation power at the cost of communication bandwidth by transmitting all sensor/motor-data to the host and back. So, increasing on-board computation power facilitates the use of a smaller communication bandwidth and vice versa. Increasing sensor/motor channels on the other hand increases the need of on-board computation power and/or communication bandwidth.

2.2 The Importance of On-Board Features

The exploitation of on-board features has, as already indicated in the introduction, several interesting scientific aspects. The two major ones are

- its relation to the constructive investigation of intelligence, and
- its importance for the emerging technological field of autonomous systems.

2.2.1 On-Board Features for Robotics and AI

On-board features are important for research in robotics as well as AI and related disciplines for several reasons. Mainly, they allow research on important aspects which are otherwise impossible to investigate, especially in the field of sensor/motor capabilities. For effector-systems for example, it is quite obvious that they have to be on-board to be within the rules of soccer-playing. Here, the possibilities of systems with many degrees of freedom, as for example demonstrated in the SONY pet dog [FK97], should not only be encouraged in special leagues as e.g. in the one for legged players, but also within the small robots league.

In the case of sensors and perception, the situation is similar to the one of effector-systems, i.e., certain important types of research can only be done with on-board devices. This holds especially for local vision. It might be useful to clarify here the often confused notions of local/global and on-/off-board. The terms on- and off-board are easy to distinguish, general properties. They refer to a piece of hardware or software, which is physically or logically present on the player (on-board) or not (off-board). The notions of local and global in contrast only refer to sensors, i.e., particular types of hardware, or to perception, i.e., particular types of software dealing with sensor-data. Global sensors and perception tell a player absolute information about the world, typically information about its position and maybe the positions of other objects on the playfield. Local sensors and perception in contrast tell a player information about the world, which is relative to its own position in the world. Unlike in the case of on- and off-board, the distinction between local and global is fuzzy and often debatable. Nevertheless, it is quite clear that the important issue of local vision can only be investigated if the related feature is present on-board of the player.

Hand in hand with an increased use of sensor and motor systems on a player, the amount of on-board computation power must increase. Otherwise, the scarce resource of communication band-width will be used up very quickly. Note, that there are many systems using RF-communication at the same time during a RoboCup tournament. Especially in the small robots league, were only few and very limited off-the-shelf products suited for communication exist, transmission of large amount of data is impossible. It is for example quite infeasible to transmit high-resolution local camera images from every player to a host for processing.

2.2.2 Autonomous Systems

In addition to the relevance of on-board features for research in robotics as well as AI and related disciplines, they are also of high interest for research on autonomous systems, i.e., computerized systems with sensors, motors and some on-board intelligence, which allows them to interact with similar devices and which frees them from permanent and explicit human supervision. Research on this type of devices is a vastly growing field and they are predicted to be a key technology of the starting millennium. Often, single mobile robots are explicitly or implicitly used as defining example for an autonomous system, see e.g. [Ste95, Mae90, HG87]. Note, that here a slightly wider notion of an autonomous system is sketched. It includes the interaction of multiple systems and it is not only limited to robots in a classical sense.

The main reason for the commercial relevance of this type of autonomous system is the continuously increasing amount of networking. With increasing availability and capacities, the use of networking facilities goes more and more beyond the mere transport, processing and provision of information. Starting with the current boom in "simple" teleoperation, actual physical manipulation through networked devices gets more and more important. The full potential of electronic commerce for example can only be reached, if a multitude of physical devices cooperate autonomously on the delivery of goods. Take for example an autonomous household, where the refrigerator and other devices keep accounts of goods, re-order if necessary, schedule transports from the "goods-port" at the front-door, and so on.

An advantage of the study of autonomous systems over the field of mere software agents [JW98] is the relation of autonomous systems to the real world. Autonomous systems are and have to be grounded in physical concepts, leading to several concrete benefits. First of the all, the real world and especially some "limited" environments as e.g. households, offices, and factories establish a better defined context than some abstract information space. Different scientists, designers, and manufactures of devices are much more likely to share common ideas and principles, which are reflected in the theory and technology of these systems. Second, for basic issues of autonomous systems, for example sensing, manipulation, mobility, there is the possibility to profit from the advanced status of already existing fields, like especially robotics. Last but not least, the real world provides, in combination with the sensors and motors, a supplementary and especially unambiguous information channel. Imagine for example the possibility of autonomous devices to schedule their activities to save energy. This is can range from a simple exploitation of lower electricity prices in certain periods of the day to an elaborated adaptation to an unpredictable supply from solar energy[1]. In this context, devices already can coordinate their activities just by individual adaptation based on measuring supply and consumption.

For autonomous systems, there are two major tasks that have to be accomplished. On the technological side, embedded devices have to be developed that provide sufficient sensor/motor interfaces and that allow for network connections. On the theoretical side, coordination of system interactions, including the issues of cooperation, communication, and group formation, have to be investigated. So, RoboCup is an ideal testbed for these two purposes.

The feature of including wireless components adds, apart from its other research issues, further attractiveness to RoboCup as testbed for autonomous systems. The development of large-scale networks for domestic and other private use will include quite some wireless components, as indicated by most recent home-networking initiatives. Wireless networking makes life easier for the common user, but on the system side, still many problems need to be solved.

3 The RoboCube as a "Universal" "Special-Purpose" Hardware

RoboCup is not laid out as a single event, but as a long-term framework for fostering significant scientific research. Within this framework, it is expected that robots, concepts and teams co-evolve trough iterated competitions. As explained before, on-board control has to play an important role

[1]The changing, partially unpredictable supply is a major problem for domestic exploitation of solar energy. The standard solution of using a large buffer in form of many lead-acid-batteries is expensive, needs quite some storage room, and is especially not very environment-friendly.

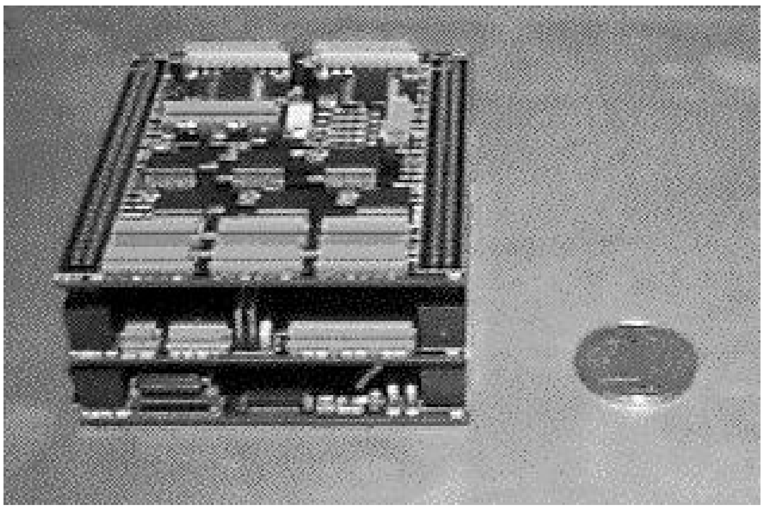


Figure 2: A picture of the RoboCube

within this process. For this purpose it must be possible to not only explore a multitude of team and control approaches, but also the vast space of physical implementations of the robot players. General body features — like speed, mobility, and so on — as well as special body features — like e.g. shooting and dribbling capabilities — are like in real soccer the basis for successful performance. It follows that the architecture of the robots must be flexible enough to allow physical changes and add-ons without requiring substantial re-engineering. Unlike in commercial robots, it must be possible to upgrade or add motors and other effectors, to use different sensors, and so on. Short, it must be possible to explore the whole space of physical interactions between the robots, the ball, and the rest of the environment.

But this flexibility has to be provided within the concrete limitations of the RoboCup regulations, especially size constraints. Therefore, a careful design of a "*universal*" "*special-purpose*" hardware is needed. The *RoboCube* is our attempt to provide the cornerstone for a conceptual framework and the technological implementation of a system allowing the fruitful and effective investigation of on-board control in the RoboCup small robots league.

The RoboCube evolved out of pure robot-control hardware developed in the VUB AI-lab. Beginning in the mid-eighties up to now, various experimental platforms for behavior-oriented architectures have been build. In doing so, experiences with approaches based on embedded PCs and different micro-controllers were gathered and lead to the *Sensor-Motor-Brick II (SMBII)*[Ver96]. The SMBII is based on a commercial board manufactured by Vesta-technology providing the computational core with a Motorola MC68332, 256K RAM, and 128K EPROM. Stacked on top of the Vesta-core, a second board provides the hardware for sensor-, motor-, and communication-interfaces. The RoboCube is an enhanced successor of the SMBII. In RoboCube the commercial computational core is replaced by our own design, also based on the MC68332, which saves significant costs, and the architecture is simplified. In addition, the physical shape of RoboCube is quite different from the one of the SMBII. First, board-area is minimized by using SMD-components in RoboCube. Second, three boards are stacked on each other leading to a more cubic design compared to the flat but long shape of the SMBII.

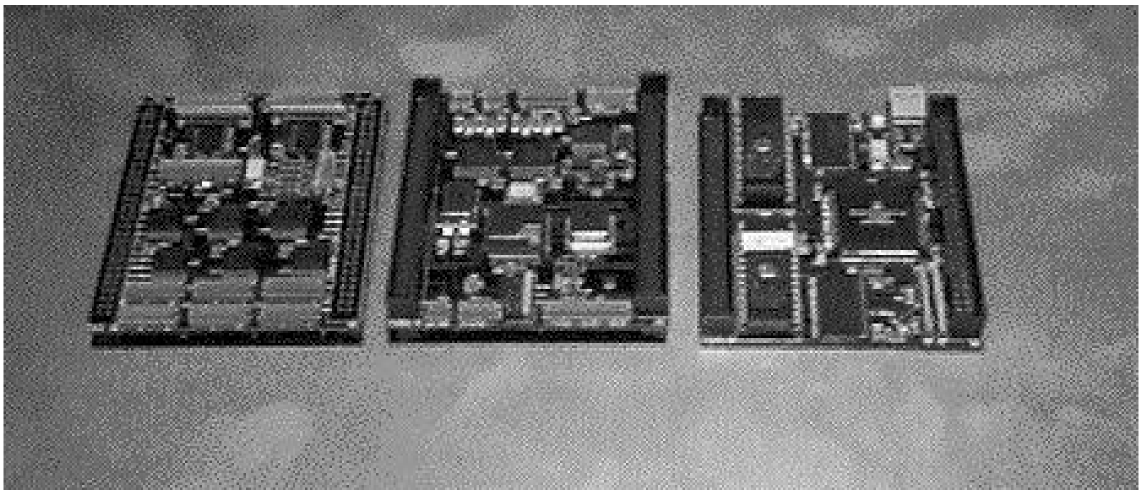


Figure 3: A picture of the main boards of the RoboCube

We demonstrated the versatility of RoboCube with our team that competed at RoboCup'98 [BWB+98]. This team consists of a heterogeneous set of robot-players, based on quite different components. We combined for example shooting-mechanisms and fast agile drives for an offensive type of player and a more precise drive-platform with good grip for a more defensive type of player. In addition to these electro-mechanical components, it is also possible to exchange sensor-systems and communication-facilities between the robots in a plug-and-play manner, or to add completely new ones.

3.1 Overview of the Architecture

In order to have a very flexible architecture we chose for an open bus system. As shown in fig. 5 there is a global bus that comprises several sub-buses which are managed from different sources.

The system is logically divided into 6 subsystems:

1. The processor subsystem contains the Motorola MC68332 microcontroller. It provides the address bus, data bus, special timing channels (TP), interrupt lines (/IRQ), several chip selects (/CS), an SPI bus and an ordinary RS232 serial bus. The capabilities of the processor are further described in subsection 3.3. Furthermore, a 1 MByte Flash-EPROM and a 1 MByte SRAM are included in this basic system.

2. The memory subsystem is separated from the processor because it then can easily be adjusted to the necessary size. In its maximum configuration, it consists of 13 MByte of DRAM memory.

3. The optional FPU subsystem provides a 25 MHz floating point unit.[2]

[2]There is also a 40 MHz version available, but at a rather high cost.

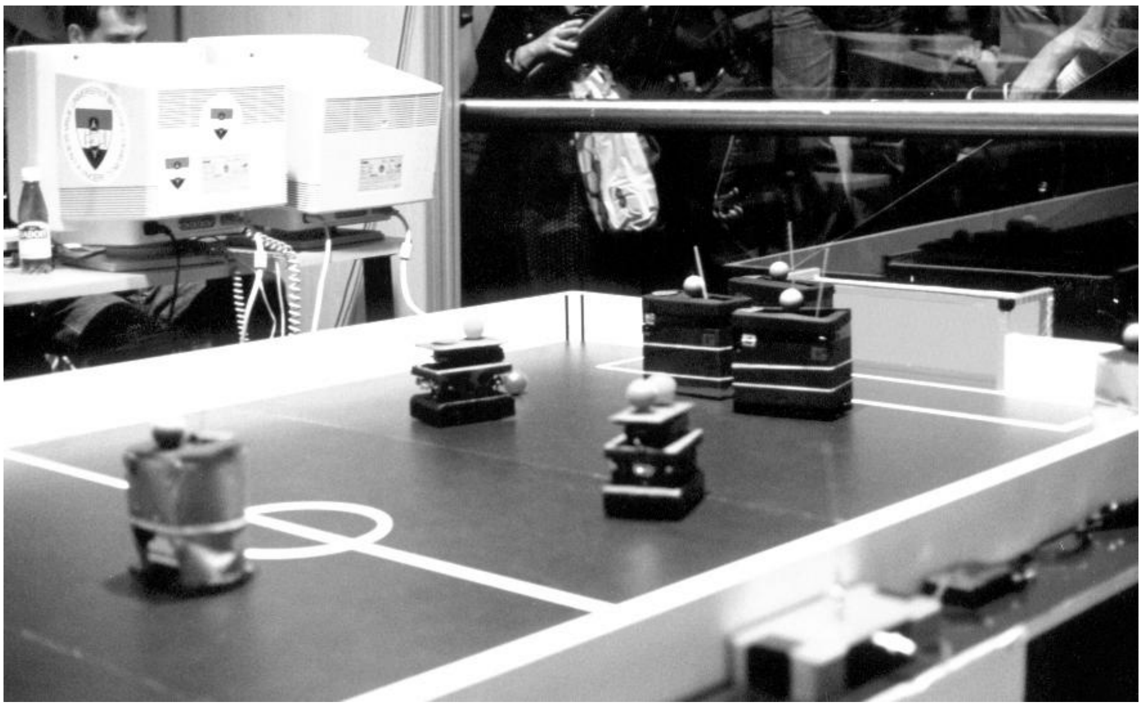


Figure 4: Cambridge University, UK, playing against the VUB AI-lab team at RoboCup'98. The two robots in the center are part of the Cambridge team, the one in the front and the three in the back are members of the VUB AI-lab team.

4. The extension busmaster subsystem provides two serial RS232 buses and two I2C buses for which a broad variety of sensors and actuators ICs are available.
5. The I/O subsystem contains all the interfaces needed in our current control for the robots.

In the current setup, there are the following I/O extensions with their proposed use available:

- binary inputs and outputs: switches, bumper, LEDs, hardware encoding of the 'name' of a robot.
- infrared transceiver: simple obstacle avoidance, slow data transfer
- UHF transceiver: medium bandwidth data transfer (40 kBit/s, very small in size)
- analog-to-digital converters: measurement of light intensity, magnetic and sound sensors, ...
- special connectors decouple the devices from the central control:
  - Motors: direct plug ins for motor controllers and shaft encoders
  - SPI: extension
  - TP: enables control of far away devices with possibly complicated timing

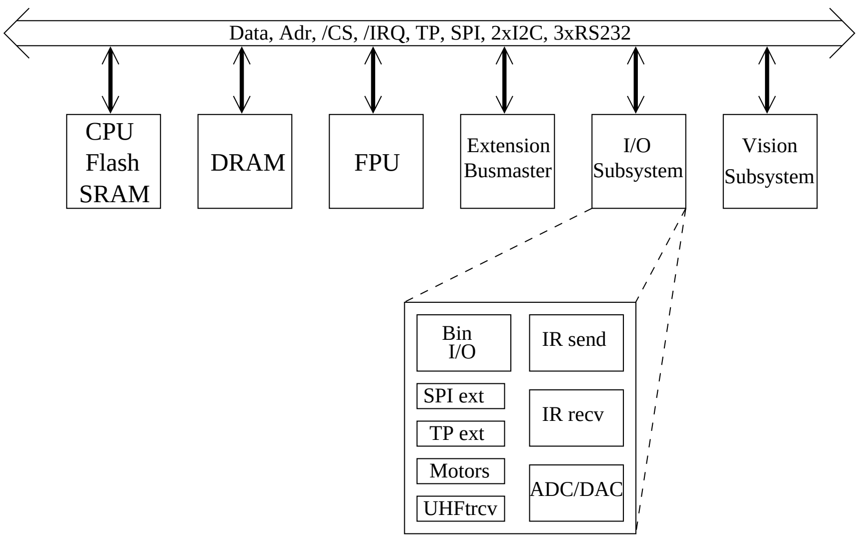


Figure 5: Block diagram of the RoboCube architecture

3.2 Features and Capabilities

The system boots out of a 1 MByte Flash-EPROM which holds a basic input/output operating system (BIOS) and offers space for a small file system. A huge part of the BIOS is dedicated to the efficient handling of different actuators and sensors.

In the basic configuration the main memory consists of a 1 MByte low power SRAM. No wait states have to be generated for that type of RAM. With the 24 bit address bus, the main memory can be extended up to 16 MByte. Since 2 MByte are allocated by SRAM and EPROM and 1 MByte is reserved for the I/O-space, the DRAM subsystem allows 13 MByte of additional main memory. Only when the DRAM performs a refresh cycle, the access to it causes some wait states. But this will happen once every 200th processor cycle and thus, not slow down its performance.

There are 3 types of serial buses in the system. Serial buses allow to bridge long distances and are usually simple to handle, but they offer normally only poor throughput. The main advantage here is the small number of wires. First of all, we have 3 standard RS232 serial busses. Hence, connecting a board to a host-computer is very simple. This is especially useful for debugging and downloading software at startup. One of the RS232 connections is dedicated to the UHF transmission system, which enables the communication to the board at run time.

The microcontroller offers a synchronous serial bus — the SPI bus. The SPI bus is fully duplex and allows multiple bus masters. It has 4 address lines encoding the destination of the transfer. Two additional lines carry the data in each direction (from and to the master). The relatively large number of wires and the fact that only a few devices are available for this bus type make the SPI bus unattractive for general usage. Nevertheless, there is the interesting option to attach a 64 bit CCD line segment, which implements a simple visual perception module, via this bus.

A very powerful bus is the inter IC bus from Philips (I^2C). It is a synchronous bus that uses only two lines. There are many devices available. The major features are:

- multiple masters
- bidirectional operation
- hardware bus arbitration
- hardware bit and byte level synchronization. This gives a slow device the opportunity to adjust the transfer speed to its own capabilities and can force wait states during the transfer.
- 7 bit addresses
- maximum 100 kBit/s

On one of our I/O subsystem board based on the I^2C, there are 24 analog input, 6 analog output channels and 16 general purpose binary in/outputs available. The board uses only one of the implemented I^2C-buses. Thus, by stacking a second I/O subsystem board into a RoboCube, the above numbers can be doubled.

One of the most striking features of the RoboCube is its size. With the size of 50mm x 60mm x 80mm, it is very small and compact. This layout relies on two special facts. First, all ICs are in SMD packages. Second, we use a special stacking connector from AMP which builds the global bus perpendicular to the boards plane. Hence, the system can be very easily extended by stacking additional boards on top of the others. Finally, due to these two connector blocks the whole layout gets mechanically very stable and guarantees secure connections.

3.3 The Motorola MC68332 Microcontroller

The MC68332 is a 32-bit integrated microcontroller in our case running at 25 MHz, combining high-performance data manipulation capabilities with powerful peripheral subsystems. The subsystems work independently from the main CPU32 instruction processing unit, thus allowing for a high overall system performance.

The two most striking features of the microcontroller are:

- the very low power consumption. It consumes a maximum of 150mA at 5V. Measurements of the current of the whole processor board (including Flash-EPROM and SRAM) turned out a consumption of 60mA only. In standby mode, the processor is even specified for 0.1mA.
- the time processor unit (TPU). 16 microcoded channels for performing time related activities from simple input capture or output compare to complicated motor control or pulse width modulation are provided.

The MC68332 and its features are described in detail in [Mot95, Mot96b, Mot96a, Mot96c].

4 The Software Aspects of RoboCube

4.1 The RoboCube's Basic Input/Output Operating System (BIOS)

RoboCube's BIOS has a quite special design to address the specific needs for robotic control [Ver96]. Besides the usual BIOS functionality like serial I/O for program download and debugging, it provides the following additional core functions:

- Access to sensors and actuators
- Wireless communication interface management

The BIOS consists of a c-runtime environment, a set of low-level drivers for sensors and actuators and a protocol engine for radio communication. The runtime environment provides a cooperative multitasking scheme to run several threads in a round-robin schedule. These tasks consist of system threads to service sensors and actuators and user threads. Each round-robin cycle is triggered by a single timer interrupt that is triggered every 40 msec.

To have a simplified and uniform access to the different actuators and sensors, a system thread regularly reads the sensors, pre-processes the sensor data and stores the results in its RAM. Then, several application threads can access these values as a special type of local variable, a quantity, without any additional delay since the sensor values are already stored in memory.

Since the actual read/write operations to the hardware are also triggered by the timer interrupt every few milliseconds, sensors are read out in well-defined synchronized time intervals. This is especially important for reading out pulse accumulators to get well-defined results. The actuators are updated the same way, also in the same intervals.

The wireless communication interface provides radio communication between several stations without any need for a designated master station. Each station is identified by a unique number (ID). The data is transmitted in packets, which can be directed to only one other station (unicast), several other stations (multicast) or all stations in the cell (broadcast). At the moment, no routing is performed and all stations in a cell are assumed to be directly reachable.

The radio communication protocol provides two communication channels, one for reliable stream communication and one for unreliable direct communication, comparable to TCP and UDP in the TCP/IP protocol suite. The reliable stream communication channel provides automatic retransmission of lost packets and proper packet reordering on reception. Since the protocol is only used for direct station-to-station communication, no adaptive windowing has been implemented. In a later version, adaptive windowing and multi-cell-routing could be implemented as well. Packets are CRC-checked upon reception and are discarded, if the CRC-check fails. Each packet in the reliable stream communication channel has to be acknowledged, non-acknowledged packets are assumed to be lost and will be resent after a timeout.

To optimize the throughput of the communication channel without neglecting the reliability aspect, the protocol engine uses two methods for obtaining transmission right on the shared medium, Time Division Multiple Access (TDMA) and Carrier Sensed Multiple Access (CSMA). Each station has its own time slot for transmission, and each station tries to identify its proper timeslot by listening to packets from other parties, identifying their IDs and synchronizing the protocol's time slots. Then the station waits for its own timeslot. But before transmission really starts, the presence of a carrier is checked. If no carrier is present, then the station starts to transmit. If, however, the transmission gets interrupted by another station, a CRC mismatch will occur and the packet is discarded upon reception. The receiving station will not acknowledge that packet and the packet will be resent. If a packet-acknowledge is lost, the sender will send the well-received packet again, but the recipient will discard the packet because a packet with the same packet sequence number has already been received.

The wireless communication interface has its own UART , interrupt service routine and server thread, therefore the communication is hidden from the application program, it only has to empty the receive queue from time to time.

On the hardware side, the wireless communication interface is implemented with a Radiometrix Bim433-F UHF Transceiver directly connected to the UART. This low power device can transmit half-duplex serial data with 40kBit/s over about 30m and provides on-board carrier detection and signal decoding circuits.

Although the protocol has been specially implemented for this architecture, it does not rely on any special features of the hardware. In a student project, a implementation of the protocol engine for windows 95 has been designed. This protocol engine serves as an application level gateway to the Internet.

4.2 Using the RoboCube for Highlevel Control

Though the RoboCube has quite some computation power for its size, its capabilities are nevertheless far from those of desktop machines. So, it is not obvious that interesting behaviors in addition to controlling the drive-motors and shooting can actually be implemented on the RoboCube, i.e., on board of the robots. Therefore, we demonstrate in this section that for example path-planning with obstacle avoidance is feasible.

24	23	22	21	20	19	18	17	16	15	14	15	16	17	18
23	22	21	20	19	18	17	16	15	14	13	14	15	16	17
22	21	20	19	18	17	16	15	14	13	12	13	14	15	16
21	20	19	18	17	16	15	14	13	12	11	12	13	14	15
20	19	18	17	16	15	14	13	12	11	10	11	12	13	14
19	18	17	16	15	14	13	12	11	10	9	10	11	12	13
18	17	16	15	14	13	12	11	10	9	8	9	10	11	12
19	18	17	16	[X]	[X]	[X]	[X]	[X]	8	7	8	9	10	11
18	17	16	17	[X]	[X]	[X]	[X]	[X]	7	6	7	8	9	10
17	16	15	16	[X]	[X]	[X]	[X]	[X]	6	5	6	7	8	9
16	15	14	15	[X]	[X]	[X]	[X]	[X]	5	4	5	6	7	8
15	14	13	14	[X]	[X]	[X]	[X]	[X]	4	3	4	5	6	7
14	13	12	[X]	[X]	[X]	6	5	4	3	2	3	4	[X]	[X]
13	12	11	[X]	[X]	[X]	5	4	3	2	1	2	3	[X]	[X]
12	11	10	[X]	[X]	[X]	4	3	2	1	0	1	2	[X]	[X]
11	10	9	8	7	6	5	4	3	2	1	2	3	4	5
12	11	10	9	8	7	6	5	4	3	2	3	4	5	6
13	12	11	10	9	8	7	6	5	4	3	4	5	6	7
14	13	12	11	10	9	8	7	6	5	4	5	6	7	8
15	14	13	12	11	10	9	8	7	6	5	6	7	8	9

Figure 6: A potential field for motion-control based on Manhattan distances. Each cell in the grid shows the shortest distance to a destination (marked with Zero) while avoiding obstacles, which are marked with '[X]'.

Path planning is with most common approaches rather computationally expensive. Therefore, we developed a fast potential field algorithm based on Manhattan-distances. Please note that this algorithm is presented here only to demonstrate the computing capabilities of the RoboCube. A detailed description and discussion of the algorithm is given in [Bir99].

Given a destination and a set of arbitrary obstacles, the algorithm computes for each cell of a grid

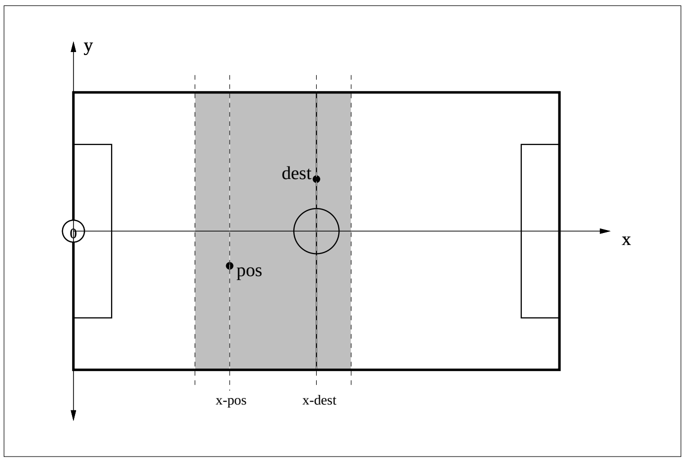


Figure 7: The potential field (grey area) is not computed for the whole soccer-field. Instead, it is limited in the x-direction to save computation time.

the shortest distance to the destination while avoiding the obstacles (figure 6). Thus, the cells can be used as gradients to guide the robot. The algorithm is very fast, namely linear in the number of cells. The algorithm is inspired by [Bir96], where shortest Manhattan distances between identical pixels in two pictures are used to estimate the similarity of images.

The basic principle of the algorithm is region-growing based on a FIFO queue. At the start, the grid-value of the destination is set to Zero and it is added to the queue. While the queue is not empty, a position is dequeued and its four neighbors are handled, i.e., if their grid-value is not known yet, it is updated to the current distance plus One, and they are added to the queue.

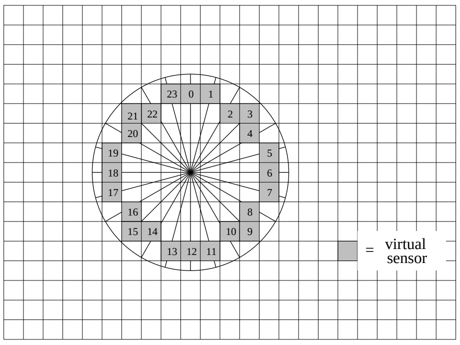


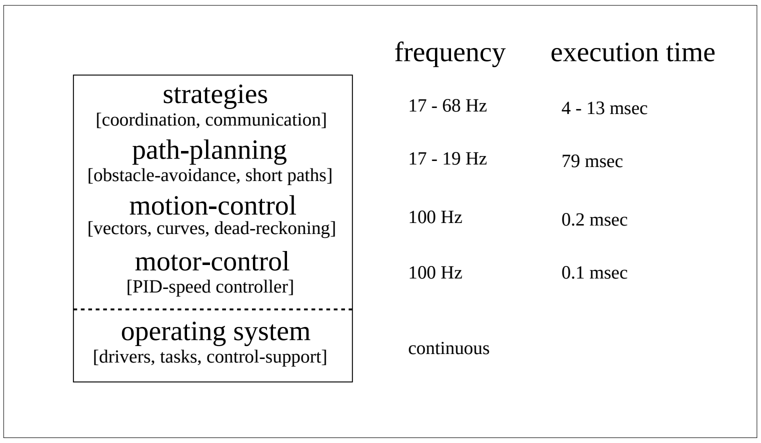
Figure 8: Twenty-four so-called virtual sensors read the potential values around the robot position on the motion grid. The sensor values can be used to compute a gradient for the shortest path to the destination, which can be easily used in a reactive motion-control.

For the experiments done so far, the resolution of the motion-grid is set to 1cm. As illustrated in figure 7, the potential-field is not computed for the whole soccer-field to save computation time.

Given a robot position *pos* and a destination *dest*, the field is restricted in the x-direction to the difference of *pos* and *dest* plus two safety-margins which allow to move around obstacles to reach the destination.

The motion-grid is used as follows for our soccer-robots. The global vision detects all players, including opponents and the ball, and broadcasts this information to the robots. Each robot computes a destination depending on its strategies, which are also running on-board. Then, each robot computes its motion-grid. In doing so, all other robots are placed on the grid as obstacles.

Robots have so-called virtual sensors to sample a motion-grid as illustrated in figure 8. The sensor values are used to calculate a gradient for a shortest path to the destination, which is ideal for a reactive motion control of the robot. In doing so, dead-reckoning keeps track of the robot's position on the motion-grid.



	frequency	execution time
strategies [coordination, communication]	17 - 68 Hz	4 - 13 msec
path-planning [obstacle-avoidance, short paths]	17 - 19 Hz	79 msec
motion-control [vectors, curves, dead-reckoning]	100 Hz	0.2 msec
motor-control [PID-speed controller]	100 Hz	0.1 msec
operating system [drivers, tasks, control-support]	continuous	

Figure 9: The path-planning is part of a four-level software architecture which controls the robots players. It runs, together with the CubeOS operating system, completely on board of the RoboCube.

Of course, the reactive control-loop can only be used for a limited amount of time for two main reasons. First, obstacles move, so the motion-grid has to be updated. Second, dead-reckoning suffers from cumulative errors. Therefore, this loop is aborted as soon as new vision information reaches the robot, which happens several times per second, and a new reactive controller based on a new motion-grid is started.

Figure 9 shows performs-results of the path-planning algorithm running on a RoboCube as part of the control-program of the robot-players. The different tasks of the control-program proceed in cycles. The execution time refers to a single execution of each task on its own (including the overhead from the operating system). The frequency refers to the frequency with which each tasks is executed as part of the player-control, i.e., together with all other tasks.

The control-program consists of four levels which run together with the CubeOS completely on-board of the RoboCube. The two lowest levels of motor- and motion-control run at a fixed frequency of 100 Hz. Single iterations of them are extremely fast as the TPU of the MC68332 can take over substantial parts of the processing. The strategy and path-planning level run in an “as fast as possible”-mode, i.e., they proceed in event-driven cycles with varying frequencies.

The execution of the pure strategy-code, i.e., the action-selection itself, takes up only a few milliseconds. Its frequency is mainly determined by wether the robot is surrounded by obstacles or not, i.e., wether path-planning is necessary or not. The computation of the motion-grid takes most of the 79 msec needed for path-planning. As two grids are used, one still determines the motion of the robot while the next one is computed, the cycle-frequency is at least 17 Hz. So, in a worst case scenario where the player is constantly surrounded by obstacles, the action-selection cycle can still run at 17 Hz.

5 Conclusion

In this article, we presented the most recent version of our RoboCube, a "universal" "special-purpose" robot-controller. It is somehow universal as it allows an easy and flexible construction of a multitude of players. For this purpose, the RoboCube provides quite some computation power and memory as well as a multitude of I/O-interfaces. It is at the same time a kind of special-purpose solution as it is tailored to fit the particular needs and constraints of the small robot league. It facilitates the use of many sensors and effectors, including the support from its BIOS to access them in high-level programs.

Concretely, RoboCube has a open bus architecture which allows to add "infinitely" many sensor/motor-interfaces (at the price of bandwidth). But for most applications, including playing soccer, the standard set of interfaces should be more than enough. RoboCube's basic set of ports consists of 24 analog/digital (A/D) converters, 6 digital/analog (D/A) converters, 16 binary Input/Output (binI/O), 5 binary Inputs, 10 timer channels (TPC), 3 DC-motor controller with pulse-accumulation (PAC), and a wireless 40 kBit communication channel.

As mentioned before, the access to these ports is supported for high-level programming languages, especially C. In addition, software modules exist to ease the direct usage of complete sensor- and effector-systems, like e.g. for motion-control. But not only lowlevel sensor-motor behaviors can run on board of the RoboCube. To demonstrate its capabilities with an example, we described the implementation of a path-planning algorithm with obstacle-avoidance on board of the robots.

Acknowledgments

Many thanks to all current and former members of the VUB AI-lab, especially Dany Vereertbrugghen who developed the SMBII.

Andreas Birk is a research fellow (OZM-980252) of the Flemish Institution for Applied Research (IWT). Research on the RoboCube is partially funded with the TMR-grant "Development of an universal architecture for mobile robots" (ERB4001GT965154).

The VUB AI-Lab RoboCup team thanks Sanders Birnie BV as supplier and Maxon Motors as manufacturer for sponsoring our motor-units.

References

[Asa98] Minoru Asada. Posting on august 4. RoboCup small robots mailing-list, 1998.

[Bir96] Andreas Birk. Learning geometric concepts with an evolutionary algorithm. In *Proc. of The Fifth Annual Conference on Evolutionary Programming*. The MIT Press, Cambridge, 1996.

[Bir99] Andreas Birk. A fast pathplanning algorithm for mobile robots. Technical report, Vrije Universiteit Brussel, AI-Laboratory, 1999.

[BWB+98] Andreas Birk, Thomas Walle, Tony Belpaeme, Johan Parent, Tom De Vlaminck, and Holger Kenn. The small league robocup team of the vub ai-lab. In *Proc. of The Second International Workshop on RoboCup*. Springer, 1998.

[FK97] Masahiro Fujita and Koji Kageyama. An open architecture for robot entertainment. In *Proceedings of Autonomous Agents 97*. ACM Press, 1997.

[HG87] L.O. Herzberger and F.C.A. Groen, editors. *Intelligent Autonomous Systems*. Elsevier Science Publishers, 1987.

[JW98] Nicolas R. Jennings and Micheal R. Wooldridge, editors. *Agent Technology; Foundations, Applications, and Markets*. Springer, 1998.

[KAK+97] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proc. of The First International Conference on Autonomous Agents (Agents-97)*. The ACM Press, 1997.

[KTS+97] Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. The robocup synthetic agent challenge 97. In *Proceedings of IJCAI-97*, 1997.

[Mae90] Pattie Maes, editor. *Designing Autonomous Agents*. The MIT Press, 1990.

[Mot95] Motorola. Mc68332 user's manual. Technical Report http://mot-sps.com/mcu/documentation/pdf/332um.pdf, Motorola Literature Distribution, 1995.

[Mot96a] Motorola. M68300 family cpu 32, reference manual. Technical Report http://mot-sps.com/mcu/documentation/pdf/cpu32rm.pdf, Motorola Literature Distribution, 1996.

[Mot96b] Motorola. Mc68332 technical summary. Technical Report http://mot-sps.com/mcu/documentation/pdf/332tsr2.pdf, Motorola Literature Distribution, 1996.

[Mot96c] Motorola. Time processing unit (tpu) reference manual. Technical Report http://mot-sps.com/mcu/documentation/pdf/tpurm.pdf, Motorola Literature Distribution, 1996.

[Ste95] Luc Steels, editor. *The Biology and Technology of Intelligent Autonomous Agents*. NATO ASI series. Springer, 1995.

[Ver96] Dany Vereertbrugghen. Design and implementation of a second generation sensor-motor control unit for mobile robots. Technical Report Thesis, Tweede Licentie Toegepaste Informatica, Vrije Universiteit Brussel, AI-lab, 1996.