Chapter 20
The RoboCup Mixed Reality League – A Case Study

Reinhard Gerndt, Matthias Bohnen, Rodrigo da Silva Guerra, and Minoru Asada

Abstract In typical mixed reality systems there is only a one-way interaction from real to virtual. A human user or the physics of a real object may influence the behavior of virtual objects, but real objects usually cannot be influenced by the virtual world. By introducing real robots into the mixed reality system, we allow a true two-way interaction between virtual and real worlds. Our system has been used since 2007 to implement the RoboCup mixed reality soccer games and other applications for research and edutainment. Our framework system is freely programmable to generate any virtual environment, which may then be further supplemented with virtual and real objects. The system allows for control of any real object based on differential drive robots. The robots may be adapted for different applications, e.g., with markers for identification or with covers to change shape and appearance. They may also be "equipped" with virtual tools. In this chapter we present the hardware and software architecture of our system and some applications. The authors believe this can be seen as a first implementation of Ivan Sutherland's 1965 idea of the ultimate display: "The ultimate display would, of course, be a room within which the computer can control the existence of matter . . ." (Sutherland, 1965, Proceedings of IFIPS Congress 2:506–508).

Keywords Mixed systems · Robotics · RoboCup · Physical control

20.1 Introduction

In the RoboCup robotics conference and challenge [2], there are two domains that can be easily distinguished – the virtual-robot simulation leagues and real robot leagues. In the simulation leagues, typically complex scenarios are implemented

R. Gerndt (✉)
University of Applied Sciences Braunschweig/Wolfenbuettel, Wolfenbuettel, Germany
e-mail: r.gernd@fh-wolfenbuettel.de

E. Dubois et al. (eds.), *The Engineering of Mixed Reality Systems*, Human-Computer Interaction Series, DOI 10.1007/978-1-84882-733-2_20,

and often a large number of virtual "robots" or agents are used. The 2D soccer simulation league, which is the only league playing soccer with a full team size of 11 players, may serve as an example of where game strategies for an entire team can be evaluated. The other extreme is the teen-size humanoid league, with real humanoid robots. These complex robots with many degrees of freedom are hard to control, considerably expensive and a large field, or environment, is required, making it much more difficult to move beyond basic ball handling and self-balancing skills.

There have been numerous initiatives trying to combine the better of the two scenarios such as in [3, 4]. The RoboCup mixed reality system bridges the gap between virtual and real robot leagues [5–8]. It allows for complex scenarios and real devices in a same system. See Fig. 20.1 for a typical example of our system in use.

Fig. 20.1 Mixed reality five versus five real robots soccer game on a 42" display with virtual ball

There are taxonomies to classify mixed reality systems. However, we believe that the general nature of the presented system framework requires to relate the classification to specific applications. The micro-robot soccer with augmentation of the robots may be located in the augmented reality domain of the reality–virtuality continuum as presented in [9]. However, if the robots are used to augment some virtual objects, this may be located in the augmented virtuality domain.

For our mixed reality system we differentiate objects according to two bipolar criteria: we distinguish whether an object is static or dynamic and whether it is real or virtual. We consider all four types of objects to be possibly controlled by the real world or the computer. First, there are *static, virtual objects*, e.g., the soccer goals, which in our soccer application are virtual and do not change. These objects may have properties that are relevant for the application, as a ball in the goal is relevant for the scoring in a soccer game. Second, there are *static, real objects*, e.g., the physical boundaries of the display, or possibly any other real unmovable object, placed within these boundaries. The static objects do not change in position, orientation, or appearance during system lifetime. The real, static objects do have real and virtual properties, which may differ. A solid, physical obstacle may, for example, be "tunneled" by a virtual object but not by a physical one. Third, there are

dynamic, virtual objects, e.g., the soccer ball used in our typical soccer application. They can change position, orientation, appearance, and so on. Lastly, there are the *dynamic, real objects*, which in our system are typically the robots or other devices controlled by the real world or real physics, like a real soccer ball, for instance. Properties of these objects may differ, depending on the control applied by the system, if any. Robots, e.g., may behave according to virtual limitations, such as having a maximum speed lower than physically possible.

There have been proposals for active, physical objects in mixed reality systems before, e.g., like the "propelled bricks" or a "magnetic puck" [10, 11]. However, unlike the previous proposals, the use of possibly autonomous robots offers higher flexibility. The system allows to control position, orientation, and possibly additional properties of a large number of robots independently of each other and independently of their position with respect to each other and on the screen.

The target domain for the RoboCup mixed reality system is research and edutainment. Education is addressed by an easy access to the programming and low costs of the overall system with a considerably high number of robots occupying minimal space. The use of real robots makes the system much more attractive for students than the typical pure virtual multi-agent frameworks without bringing much additional burden in terms of hardware complexity or difficult programming frameworks. The system has also been used in introductory programming experience in undergraduate courses and it demonstrated extreme effectiveness in the motivation of the students, who, based on prepared templates, were able to program soccer playing behavior despite their limited experience and novice status [12, 13]. Specific applications, e.g., like soccer can also be used for entertainment. Then users may, for example, take control of some robots via gamepads.

Possible research scenarios span from typical simulation-only robotic multi-agent systems to applications, only possible in real multi-robot systems. They include the fields that emerge from the interaction of real and virtual worlds. Examples include soccer, but also traffic simulation [14], human coaching, automated role assignment, multi-robot learning strategies, and swarm applications based on local on-robot sensing and actuation. Moreover, because the robots are so small, new applications become also possible, such as the use of the micro-robots and the entire mixed reality system in insect mixed society experiments (Fig. 20.2).

As also shown in the European LEURRE project [15], insects can be deceived to "believe" the robots are members of their group. This allows the programming of exhaustively long behavioral experiments that only robots can perform.

Our first mature benchmark application combining research, education, and entertainment aspects was a two versus two robotic soccer game performed on a 20'' display. The current benchmark is a five versus five game on a 42'' display. In the future, with even larger screens, 11 versus 11 is targeted. Aside from robotic soccer there already are numerous other applications that explore research, education, and entertainment in a well-balanced way. This includes a Pacman-like game with real "Pacman" and virtual ghosts, an ice hockey game, where real robots have been "equipped" with virtual hockey sticks and a virtual ice hockey puck with "virtual" physics different from a soccer ball. In this chapter we will present a racing

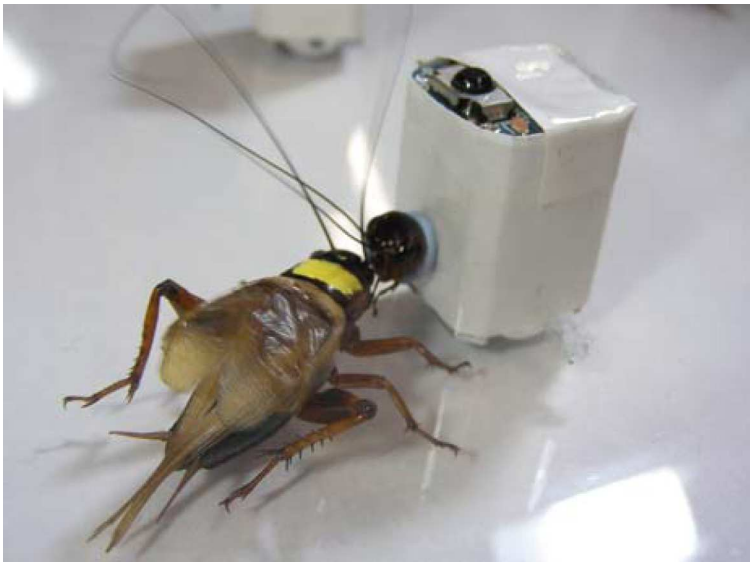


Fig. 20.2 Mixed society experiment where the head of a cricket was attached to a robot, thus deceiving real crickets because of pheromone signatures

game with virtual and real obstacles, illustrating the traveling salesman and similar problems. All applications can be implemented on the same system framework.

The basic hardware setup of the RoboCup mixed reality system consists of a horizontally mounted display and a set of micro-robots. The left part of Fig. 20.3, marked "real," sketches the hardware setup. A camera that is mounted above the

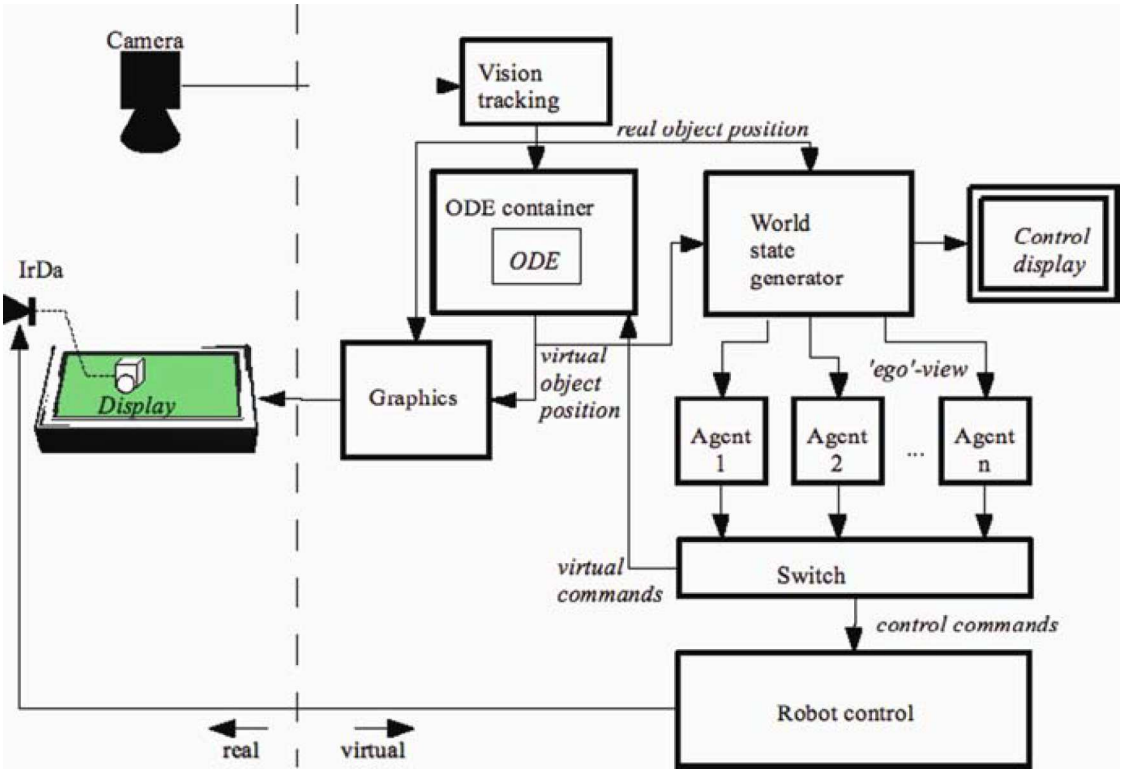


Fig. 20.3 Structure of RoboCup mixed reality system

screen captures the scene with all virtual and real objects. Vision tracking allows determining the position and orientation of the robots and other real objects on the screen. For identification, each robot can be equipped with an individual marker. The number of individually distinguishable robots depends on the markers used and on the resolution of the camera. The robots can move freely on the screen. Individual software agents control the robots. An IrDa link is used for wireless data exchange.

The right part of Fig. 20.3, marked "virtual," shows the structure of the software. There is an overall framework with a number of modules for input and output, simulation and control. The vision-tracking module captures the camera output and provides information on positions and poses of the robots and possibly other real objects to the other software modules. The world state generator generates an individual view for every single robot in the system. A control display can be attached for debugging and development purposes. The individual views are then communicated to the agents that control the robots. There is one individual agent module for every robot. The switch module separates the commands issued by the agents into commands that affect virtual objects, like kicking a virtual ball and control of real robots. The robot control module takes care of interfacing and communicating with the robots. The ODE container wraps the Open Dynamics Engine (ODE) [16] physics engine and takes care of simulation of the virtual objects. It processes data of real objects, like position and space occupied, and commands that affect virtual objects. It outputs information on new poses and positions of the virtual objects. The graphics module displays all virtual objects on the screen.

In our chapter, we will present the overall hardware and software architectures of the RoboCup mixed reality system. We will then present the results of the developments based on the system, done by a still increasing community from academia all over the world and present plans for future developments and some conclusions.

20.2 Hardware Architecture

The hardware architecture of the system is composed of four main components and their accessories, namely the micro-robots, the augmented reality display, one or multiple tracking cameras, and a computer.

20.2.1 Micro-Robots

The micro-robots are small differentially driven mobile vehicles with an approximate dimensions of a 2.5 cm cube (Fig. 20.4). The envisaged target cost for the micro-robots for series production is less than 20 •. The cube body is composed of a few milled aluminum parts screwed together and internal batteries. The only moving parts are the drives and the wheels. The diameter of the wheels and the size of the robot were designed so as to find the best balance of torque and velocity, while keeping space for batteries so as to ensure maximum autonomy and minimum weight. The metallic wheels have their axis of rotation just slightly below the

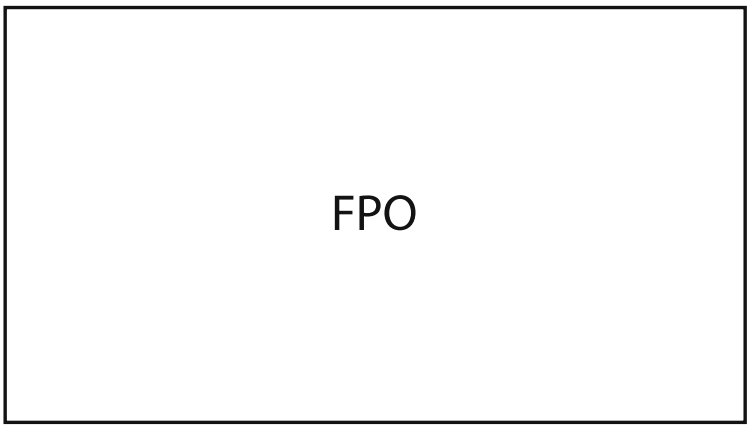


Fig. 20.4 Most recent generation of micro-robots

geometric center of the body, with the largest diameter possible so as to occupy both lateral faces almost entirely.

Designed with expandability in mind the accompanying controller board features an expansion bus and a power processor allowing the stacking of custom-designed circuit boards on top of it. Both the front and the back of the body have six extra screw holes distributed in a pattern similar to the six faces of a dice. These holes allow the attachment of external devices such as sensors, actuators, or even adjacent circuits. There are extra mounting holes for the motors allowing the wheels to be optionally assembled with their axis near the bottom edge of the body (Fig. 20.5). Four more holes are located on the bottom surface allowing the attachment of devices on the bottom of the robot.

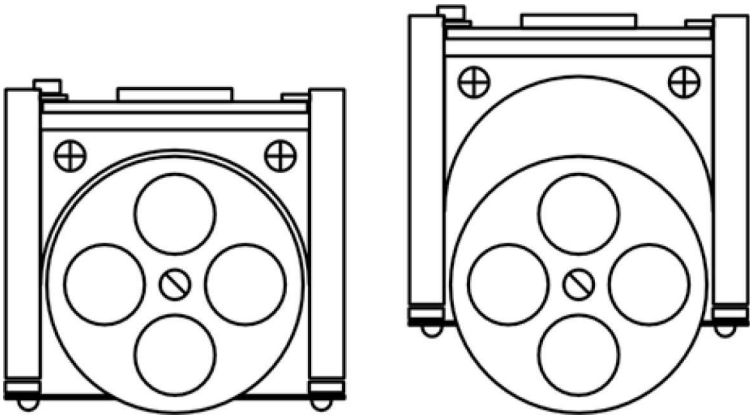


Fig. 20.5 Two types of wheel placement are possible

The robot is equipped with two micro-stepper motors (Fig. 20.6), specially modified for achieving higher torques and bi-directional actuation. These motors were originally designed for automatically adjusting the focus of the lenses of micro-cameras in mobile phones and other miniature devices. The wheels have no encoder, but approximate velocity can be managed obtained through the control of the stepping signals (Table 20.1).

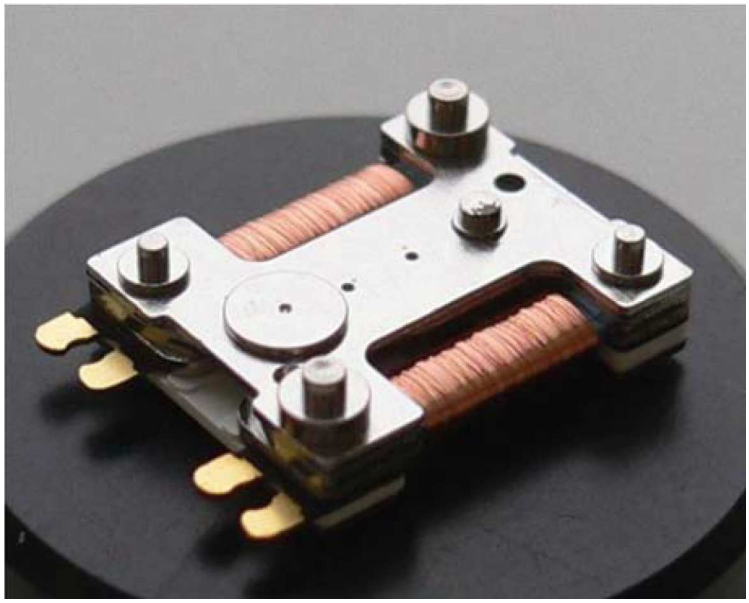


Fig. 20.6 Close-up macro-picture of the micro-stepper motor

Table 20.1 Stepper motor specification [17]

Feature	Value
Size w, d, h (mm)	7.0, 8.5, 1.9
Configuration	2 coils, 1 rotor
Gear ratio	1:240
Torque at 2.8 V (g cm)	2.0–4.0
Power consumption at 200 rps (mA)	4–12
Max. rotation speed (rpm)	12,000
Direction	Bidirectional

The detachable controller on the top of the robot (Fig. 20.7) consists of a board with a crystal, a voltage regulator, an IrDA transceiver, a connector and it contains a 32-bit ARM as the main microcontroller and an 8-bit AVR microcontroller as its slave, the later being mainly dedicated to the control of the stepper motors. Table 20.2 summarizes the main features of bothprocessors.

Table 20.2 Controller specification

Feature	Master processor	Slave processor
Architecture	ARM	AVR
Type	AT91SAM7 × 256	Atiny84
Flash memory size (kb)	256	8
RAM (kb)	65	0.5
Frequency (MHz)	48	8
Compiler	GCC	GCC
Uploading interface	JTAG, USB, Serial	SPI

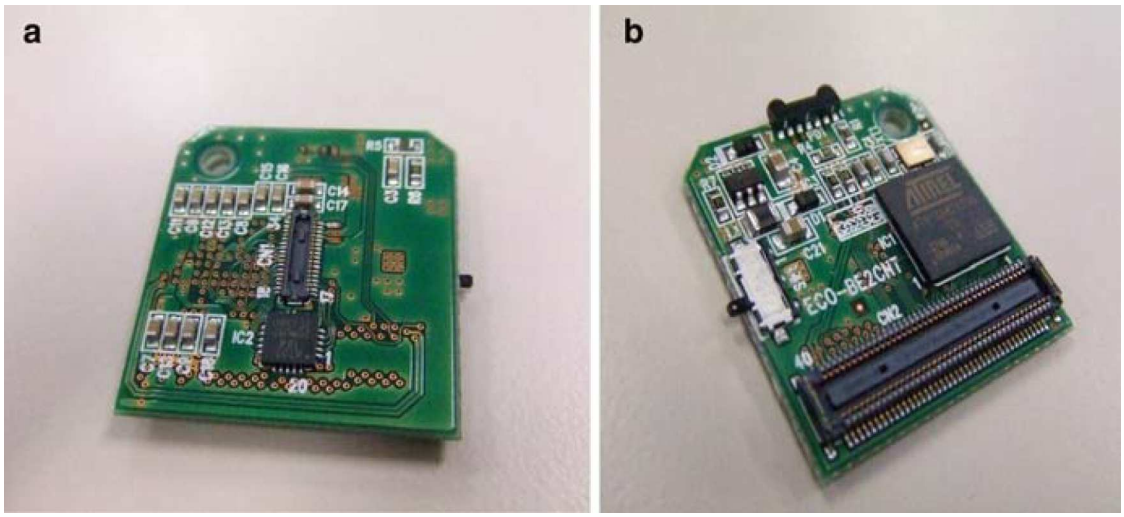


Fig. 20.7 Micro-robot:(**a**) controller board bottom view and (**b**) controller board top view

The controller itself does not come equipped with any sensors, actuators, or other specialized hardware except for the IrDA transceiver. Most pins of the ARM processor and some of the pins of the AVR processor are available on the 80-pin top connector. This means one can easily stack a peripheral board on that connector, adding extended functionality to the robot. The ARM processor has a high-performance 32-bit architecture, with periodic interval and real-time timers, 62 I/O lines, built-in communication interfaces, four 16-bit PWM channels, two-wire interface, 8-channel 10-bit AD converter, and much more. This allows the support for a myriad of features requiring only a minimum number of external components. Such features include microphone and speaker support, micro-SD card support, wireless Ethernet, LCD output, USB support, relays, DC motor controlling, and the connection of intelligent sensors, such as gyroscopes, accelerometers, and others capable of communicating through SPI or CAN ports. The ARM microcontroller controls the AVR microcontroller over one of its SPI interfaces.

In order to use the robots three additional accessories were also designed: a USB-to-infrared transmitter, a firmware uploading interface board, and a battery charger.

20.2.1.1 Battery Charger

The battery charger device consists of a platform on which six robot bodies can be attached upside down for simultaneous recharging. Each robot contains two internal lithium-ion polymer cells, with 190 mA h each. Tests show this is enough for at least 4 h of continuous operation or even more if alternating between idle and moving states, as it happens in several applications. The terminals of the two internal batteries are made available in the body's top connector in a way to allow the

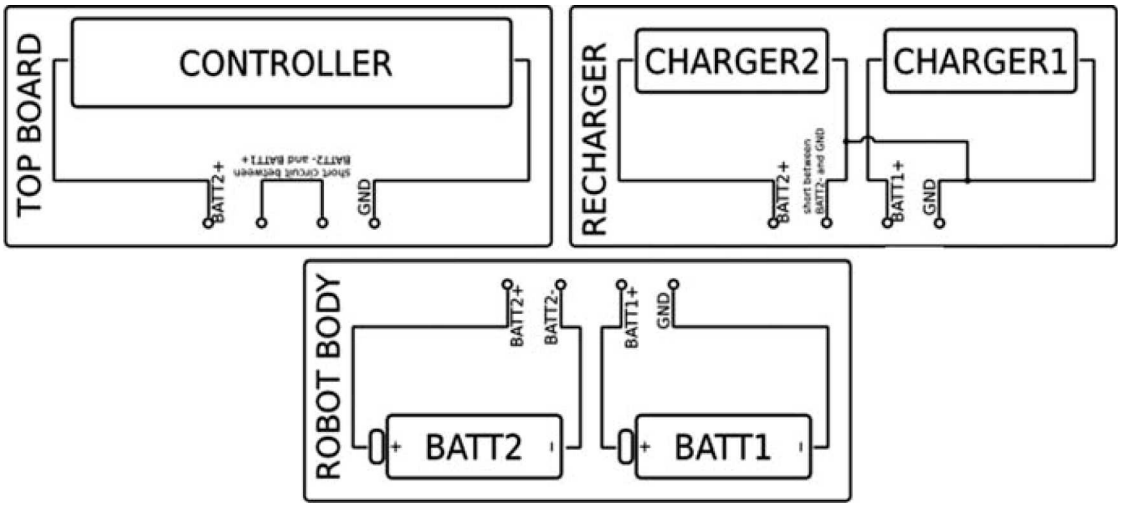
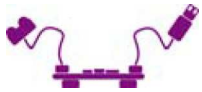


Fig. 20.8 Serial versus parallel battery configuration in operation versus charging

serial connection of the cells when robot body is attached to the microcontroller and parallel charging with two individual charging circuits when the body is connected to the charger (Fig. 20.8).

20.2.1.2 Infrared Transmitter



The infrared transmitter consists of a USB-to-serial chip converter, connected to an IrDA modulator and a FET amplifier circuit for boosting the range of the transmitted signal. Data are sent directly from the computer through the USB port and converted into short pulses in four high-power infrared LEDs distributed on each of the four corners of the field. The whole perimeter of the field is framed with reflective walls that act like mirrors so as to reflect the transmitted signal, thus widening otherwise restrict angle of coverage of the LEDs and also avoiding possible occlusions because of obstacles or other robots in the way.

20.2.1.3 Firmware Uploading Interface Board

The firmware uploading interface board is the device that allows the use of full-sized standard connectors with the small robots for programming and debugging. The board consists of a platform in which a robot body can be attached upside down and a robot controller can be attached just next to it. The connections are routed to full-sized connectors and to debugging LEDs through jumpers. The full-sized connectors include industry standard JTAG and SPI headers as well as a legacy

serial DB9 connector and a type-B USB device connector. The firmware of the AVR microcontroller can be uploaded and debugged through the SPI. For low-level debugging one has the option to either use the LEDs or a robot body. The firmware of the ARM microcontroller can be uploaded via USB or via JTAG. The ARM supports step-by-step debugging. The robot body can be powered by its internal batteries or by external power supply.

20.2.2 Augmented Reality Display

One of the original characteristics of the system and an essential part of the mixed reality environment is the horizontally mounted augmented reality display. Objects and robots can be placed on the screen with the display of virtual objects. If required, virtual objects can be virtually attached to the robots.

Some brands of LCD screens were found to emit infrared light, thus causing interference with the IR control signals. To avoid this and also reduce reflections of ceiling lights, polarizing and other optical filter sheets can be used.

Polarizing filters are also used for facilitating the automatic detection of unknown artefacts that can be randomly placed over the screen. By mode of operation, all light emitted by an LC display has the same polarization angle. Therefore, a polarization filter in front of the camera lens can be adjusted to prevent all light emitted by the display from reaching the camera. This technique helps separating virtual objects from real objects in the image processing (Fig. 20.9).

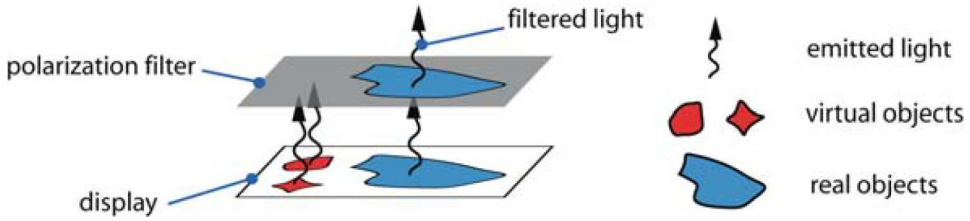


Fig. 20.9 Use of polarizing filter distinguishes between real and virtual objects on the display

20.2.3 Tracking Camera

A high-resolution tracking camera is responsible for the capturing of the detailed images necessary for the feedback of robot's position and orientation and other real objects on the screen as well as their identification (Fig. 20.10).

For sufficient accuracy the camera image has to be calibrated, in order to compensate distortion that may result from misalignment of camera and display. Initially a rectangular grid displayed on the LCD had to be matched with a checkerboard by manually adjusting the intersections of the grid. If markers with colors are used, a color calibration by manual teach-in is required in order to adopt the system to the ambient light. For future use, automated schemes have been proposed. With proper calibration of the display, static offsets are neglectable. However, the limited frame

Fig. 20.10 Camera – display setup

rate of the camera and the image processing may result in a dynamic offset between moving real objects and attached virtual tools or augmentation.

Currently, the typical setup consists of a high-resolution IEEE 1394 firewire camera capable of at least 15 fps at 1280 × 1024 or even higher rate and resolution industrial Gigabit Ethernet camera. Current prices for these cameras typically are in the range of 800 • and more. With respect to increased field size, use of multiple cameras and combining their pictures into a single image may become necessary, as indicated in the left part of Fig. 20.10. With reduction of overall costs in mind, using multiple low-cost web cams has been proposed.

20.2.4 Computer

The computer server is a central part of the system. The basic requirements are as follows: multiple core CPU with performance of a Core 2 Duo 2.3 GHz or superior, hardware OpenGL accelerated GPU for the rendering of graphics on the augmented reality display, 2 GB of RAM, port for the connection of the camera(s), and network interface for connection of clients into the system.

20.3 Software Architecture

The software architecture comprises five domains to be possibly maintained independently. These domains are vision tracking, graphics, application, robot control, and agents (Fig. 20.11). XML structures were specified to define the UDP datagram communication over Ethernet within the framework. XML was chosen as the data

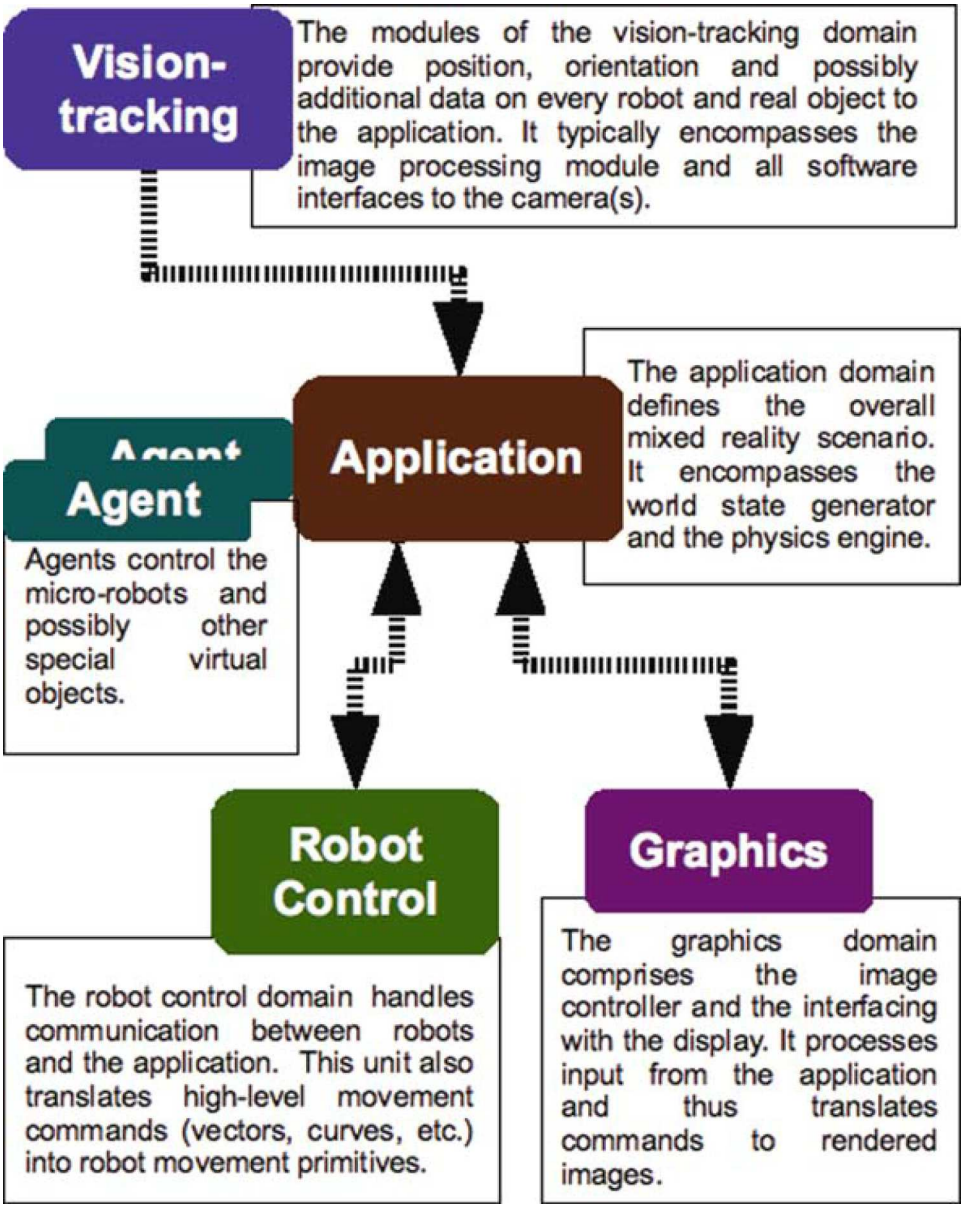


Fig. 20.11 Software domains overview

format for its suitability to encode any kind of information, human readability, and broad availability of tools to parse and validate files. UDP was chosen as a transport protocol for its simplicity and speed. Communicating over Ethernet allows the modules to run on different computers in the network.

On startup, the modules may exchange initialization data, as often needed for correct collaboration. For example, the application module might send the dimensions of the mixed reality display in millimeters to the graphics module (Listing 1).

"<type>" and "<protocol_version>" are required. The former describes the type of communication and the latter defines the protocol version. In this example "<screen_dimension>" is an extension of the basic specification that is required when the application module connects to the graphics module. Different mixed reality applications may need to exchange different data with other modules. For

```
<connect>
        <type>Graphics</type>
        <protocol_version>1.0</protocol_version>
        <screen_dimensions>
                <width>400</width>
                <height>300</height)
        </screen_dimensions>
</connect>
```

Listing 1 XML initialization example

example, the mixed reality soccer application module may want to send information about the state of the game (running, paused, kick-off, penalty, half-time) to the client module. This information is specific to soccer only. Therefore, applications may extend the basic XML specification with application-specific conventions.

20.3.1 Vision-Tracking Module

The vision-tracking module is responsible for tracking all real objects on the field. It retrieves images from the camera above the field. Two separate image-processing algorithms then extract information. The marker detection algorithm extracts the ID, position, and orientation of all markers, placed on micro-robots or other real objects. The real-object tracking algorithm extracts the ID, position, and outline of all real objects (including the robots) on the field. The real-object tracking algorithm relies on a polarization filter placed in front of the camera lens, adjusted so that all light emitted by the mixed reality LC display is absorbed.

In a first step the captured image is subtracted from a given background image, smoothed and turned into a binary image through a threshold filter (Fig. 20.12). In a second step the Teh-Chin chain approximation algorithm is applied to the binary image, resulting in a list of all contours in the binary image. This list is reduced to a reasonable amount of contour points per object, which is eight points by default.

The results of both tracking algorithms are merged, encoded in XML, and sent to the application modules. Listing 2 shows an example of an XML file, containing a list of an object and a robot. Each object is identified by its ID and shape. A shape consists of a number of x-, y-coordinates. A robot is defined by its ID, orientation, and center point.

20.3.2 Application Modules

The application modules are the central hub of the software stack. They form the core of the mixed reality server and are responsible for managing connections to all other modules and providing the mixed reality functionality. The application

Fig. 20.12 Vision tracking: (*upper left*) background image, (*lower left*) image as seen by the camera, (*upper right*) shapes of artifacts, and (*lower right*) image with identified real objects

```
<worldData>
    <object>
        <id>42</id>
        <shape>
            <pos><x>156</x><y>349</y></pos>
            <pos><x>207</x><y>291</y></pos>
            <pos><x>199</x><y>325</y></pos>
        </shape>
    </object>
    <robot>
        <id>7</id>
        <orientation>180</orientation>
        <pos><x>146</x><y>120</y></pos>
    </robot>
</worldData>
```

Listing 2 Example XML description of object and robot

modules receive the real-object information from the vision-tracking module. They then merge information of real and virtual objects and forward this combined world data to the agent modules in the clients' domain. They also send commands to the graphics modules to display the virtual objects. Further on, they process control commands from the client modules that affect virtual objects or forward them to the robot control module.

Two implementations of the application modules exist. The first implementation is a mixed reality soccer server and the second is a generic C++ framework that provides developers with an API to develop any type of mixed reality application.

20.3.3 Graphics Module

The graphics module is a 3D hardware accelerated, generic module used to display any kind of information on the mixed reality display. It is separated into two threads: The first thread listens for incoming XML files from the application modules and translates them into a list of primitives that are to be drawn. The second thread constantly iterates over the latest list of drawing commands and translates them into OpenGL calls. The available primitives are as follows: 2D lines, polygons, textures, text, 3D objects, light sources, and sound files. For playing sound files, the OpenAL library is used. Each graphical command consists at least of the name of the object to be displayed and the 2D position on the screen. The name of the object refers to the name given in the theme pack. A theme pack is an archive containing a set of primitives. As the graphics module is intended to be generic, applications can transmit theme packs to the graphics module. After activation, all primitives from the theme pack are available for display. Some parameters are only relevant for certain types of primitives, such as the pitch for sound files. Other parameters are optional, such as the layer. The graphics module subdivides the 3D space provided by OpenGL into 1000 2D layers.

The UDP protocol used to transmit the XML files gives no guarantee that every file is transmitted. Therefore, the graphics module was designed to be stateless. This means that each XML file has to contain a complete list of all objects that are to be drawn. If there were commands to enable/disable the display of objects this could lead to accumulation errors if some XML files are lost in transmission.

20.3.4 Robot Control Module

The robot control module receives high-level control commands for the micro-robots from the application module. It translates these commands into low-level commands for the micro-robots and sends them via USB to the IrDA transmitter.

The robots may be addressed individually or in a broadcast mode. The commands are composed of one mandatory command keyword and multiple optional arguments for that command. Commands are sent to the robot separated by commas. A command can be wrapped within parenthesis, and this itself can be used as

an argument, allowing the sending of nested commands. The format of the robot commands can better be understood through the following examples:

Example 1, command ‘go forward with speed 10’:
fw,0a
Example 2, nested commands:
lst,(fw,14),(slps,01),stop
Example 3, loop command:
lp,03,(lst,(fw,14),(slps,01),stop,(slps,01))

In Example 1, the command takes one argument (a number in hexadecimal notation). Other commands, e.g., the “stop” takes no argument. The command in Example 3 allows a list of commands to be executed in sequence. The command line of Example 3 is a nested loop command. The first argument tells how many times to run the loop and the second argument is the command to be executed in the loop. This command makes the robot go forward for 1 s, stop for 1 s, and repeat that three times.

All the examples previously described are broadcasted and executed by all robots. In order to control the robots independently, the robots need to be addressed by their IDs. There are two types of IDs in the robots, a fixed firmware ID (fid) and a software ID (sid) that can be changed dynamically. Both can be used for addressing the robots. The following examples show some examples:

Example 4:
fid,00,(fw,14)
Example 5:
sid,00,(chsid,01)
Example 6:
sbkt,(fw,14),stop,(sl,14),(bw,05)

Example 4 shows the usage of the firmware ID 00. Example 5 shows how to change the software ID of a specific robot. Example 6 shows the software ID bucket command, by which one can send several different commands to different robots in a single message. When this command is issued, all robots of ID 00 will execute the first argument, robots with ID 01 will execute the second argument, and so on.

The command-parsing infrastructure was designed for usability and expandability. Commands are implemented as simple C functions using the same argument passing convention as the typical main function of a console application. The following line shows a typical example of one such function prototype for a firmware program:

```
int cmd_set_forwardvelocity(int argc, char **argv)
```

The returning integer indicates error if different than 0 and the arguments are passed as a pointer to char strings and an integer saying how many arguments are there.

20.3.5 Agents

The agents implement the artificial intelligence that impersonates the micro-robots. Several instances of the agents may connect to the application module, depending on how many micro-robots are part of the application. Each agent typically only issues commands for its specific robot. However, they may also take control of virtual objects. For example, there is a kick command to kick the virtual ball, if the robot is close enough and there are move commands to control virtual tools, like a virtual hockey stick.

20.4 Experience

20.4.1 Development Process

Currently the RoboCup mixed reality robots are in their third generation. Around year 2005 it started with a small cube robot for demonstration purpose. From this demonstrator the second generation of mixed reality robots, with all necessary support circuitry, like charger, was developed in 2006. They had dimensions of about 1×1 cm and a height of 2.5 cm. Different from the current third-generation robots, used since the beginning of 2008, they only had a single processor to handle IR communication and the stepper drives. Batteries also were smaller, as were the wheel diameters. The second-generation robots have been used for a series of tournaments and development contests all over the world. Eventually a redesign with a number of improvements was carried out, leading to the current robot generation. As with the software development, the hardware development work also was shared among the teams of the RoboCup mixed reality community.

Initially a software framework with basic features and an example agent was provided to the teams. Similar to an open-source approach the software was developed by cooperation. However, this did not include the agents that control the robots, since they represent the intelligence of the soccer robot teams, which were to play each other during the RoboCup soccer tournaments.

20.4.2 Soccer System

Different soccer scenarios have been implemented, all of them were making use of the mixed reality features of the system. The basic system has been described throughout this chapter. It makes use of a virtual soccer field and a virtual soccer ball that are virtually “kicked” by real robots. The robots are controlled by software agents, which implement different behaviors and strategies. Among others, test with dynamically changing goalkeepers against a team with fixed behavior in a two versus two robots soccer game has been carried out. Human player have been asked to control individual robots via game pads, playing with and against computer-controlled robots.

There have also been tests with a real ball and real goals. However, with a real ball, control became very cumbersome. Only very short “kicks” with a high degree of inaccuracy were possible.

20.4.3 Racing Application

In order to introduce static real objects to our mixed reality system, we developed a mixed reality racing game called beRace! (Fig. 20.13). In this application, random everyday objects are placed on the display. The game then generates a virtual racing track around these obstacles. The track consists of a number of checkpoints. Each checkpoint is a line drawn between two real objects. Depending on the game mode, the goal for the robots is to pass through these checkpoints in a predefined order. Some real objects might be light enough for the robots to move them. In this case, the virtual racing track will be adjusted to the change in reality.

To introduce further interaction between real and virtual objects we place virtual items on the screen that can be picked up by the robots by driving over them. Items are virtual rockets, mines, boosters, and oilcans. Items can be used and may change the properties of the robots. For example, a robot may lay down a puddle of oil. If another robot passes through this puddle, any incoming steering commands for this robot will be exaggerated for 3 s to simulate slipperiness.

This application demonstrates how real objects (robots, obstacles) and virtual objects (racing track, items) can interact with each other in various ways. Virtual rockets may explode upon impact with real obstacles and upon impact with virtual mines. Real robots may move real and virtual obstacles but are temporarily disabled when touching virtual mines.

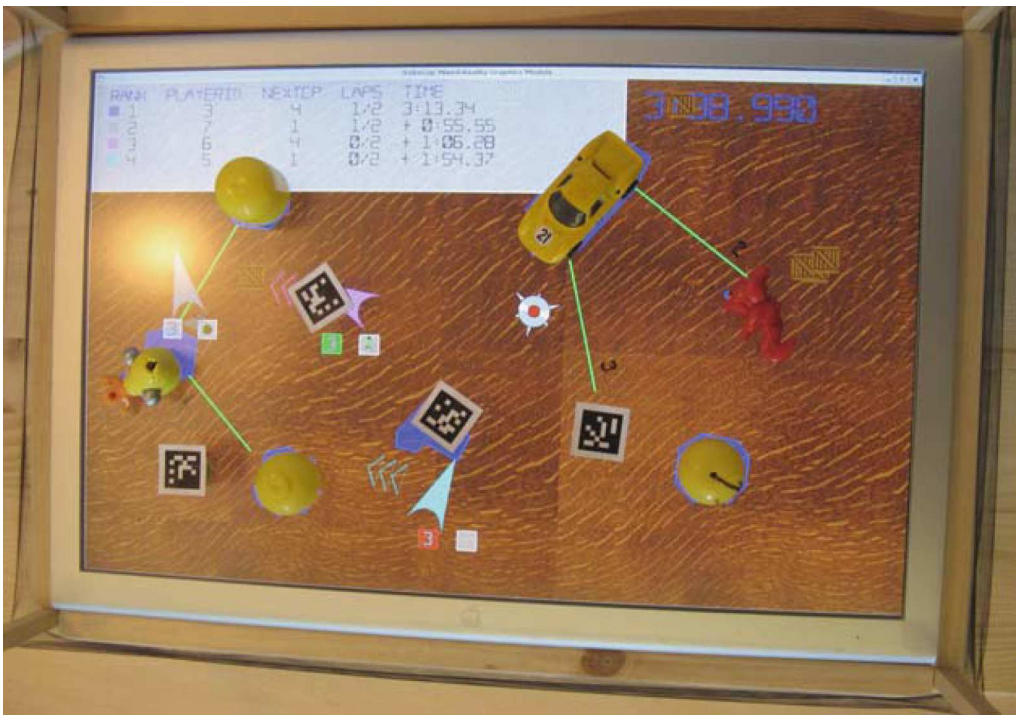


Fig. 20.13 beRace! Application, markers done with ARToolKit [18]

20.4.4 Future Developments

A number of ideas for further improvements have been presented during the RoboCup mixed reality tournaments. With low system costs as major objective, a number of concepts have been presented to overcome the problem of the still considerably expensive camera. One approach has been to use a number of low-cost webcams instead of the expensive Firewire or Ethernet cameras. Using a number of webcams not only alleviate the cost problem but also widens the area that can be used as a mixed reality arena. However, there still are a number of technical challenges, like combining the individual images into an overall picture and sustaining a sufficiently high frame rate. Another approach, going even further, was making use of low-cost cameras that are mounted to the robots. The cameras will provide an egocentric view of the robots and may be combined with the images from static cameras. However, this may lead to an incomplete coverage of the arena, e.g., if all mobile robot cameras point into a certain direction.

Aside from cameras, robots may be equipped with other sensors and actuators, e.g., to serve as tangible interfaces, which not only allow a physical input to a system but also, by moving on the screen, a physical output of a system. Further actuators may allow to change color or shape, thus allowing a true two-way interaction between virtual and real world.

20.5 Summary and Conclusions

In this chapter we presented a hardware and software architecture, suitable for a mixed reality system with far-reaching two-way interaction with multiple dynamic real objects. We gave a detailed description of the hardware architecture and of the robots, which are the key components to allow the mixed reality system to physically interact with the real world and of the software architecture. We provided real-world examples to illustrate the functionality and limitations of our system.

We showed how the overall system is structured in order to allow students to easily gain access to programming of robots and implementing artificial intelligence applications.

The RoboCup mixed reality system was used for education and for research, even in non-technical domains like traffic control. A system with tangible components as well as a virtual part offers an ideal platform for the development of future user interfaces for a close and physical interaction of computer and user.

References

1. Sutherland I E (1965) The Ultimate Display. Proceedings of IFIPS Congress 2:506–508.
2. The RoboCup in the Internet: www.robocup.org
3. Boedecker J, Guerra R da S, Mayer N M, Obst O, Asada A (2007) 3D2Real: Simulation League Finals in Real Robots. Lecture Notes in Computer Science 4020:25–34.

4. Boedecker J, Mayer N M, Ogino M, Guerra R da S, Kikuchi M, Asada M (2005) Getting Closer: How Simulation and Humanoid League Can Benefit from Each Other. Symposium on Autonomous Minirobots for Research and Edutainment 93–98.
5. Guerra R da S, Boedecker J, Mayer N M, Yanagimachi S, Hirosawa Y, Yoshikawa K, Namekawa K, Asada M (2008) Introducing Physical Visualization Sub-League. Lecture Notes in Computer Science.
6. Guerra R da S, Boedecker J, Yanagimachi S, Asada M (2007) Introducing a New Minirobotics Platform for Research and Edutainment. Symposium on Autonomous Minirobots for Research and Edutainment, Buenos Aires.
7. Guerra R da S, Boedecker J, Asada M (2007) Physical Visualization Sub-League: A New Platform for Research and Edutainment. SIG-CHALLENGE Workshop 24:15–20.
8. Guerra R da S, Boedecker J, Mayer N M, Yanagimachi S, Hirosawa Y, Yoshikawa K, Namekawa M, Asada M (2006) CITIZEN Eco-Be! League: Bringing New Flexibility for Research and Education to RoboCup. SIG-CHALLENGE Workshop 23:13–18.
9. Milgram P, Takemura H, Utsumi A, Kishina F (1994) Augmented Reality: A class of displays on the reality-virtuality continuum Paper presented at the SPIE, Telemanipulator and Telepresence Technologies, Boston.
10. Fitzmaurice G W (1996): Graspable User Interfaces. PhD at the University of Toronto.http://www.dgp.toronto.edu/~gf/papers/PhD%20-%20Graspable%20UIs/Thesis.gf.html.
11. Pangaro G, Maynes-Aminzade D, Ishii H (2002) The Actuated Workbench: Computer-Controlled Actuation in Tabbletop Tangible Interfaces. In Proceedings of UIST'02, ACM Press, NY, 181–190.
12. Guerra R da S, Boedecker J, Mayer N M, Yanagimachi S, Ishiguro H, Asada M (2007) A New Minirobotics System for Teaching and Researching. Agent-based Programming. Computers and Advanced Technology in Education, Beijing.
13. Guerra R da S, Boedecker J, Ishiguro H, Asada M (2007) Successful Teaching of Agent-Based Programming to Novice Undergrads in a Robotic Soccer Crash Course. SIG-CHALLENGE Workshop 24:21–26.
14. Dresner K, Stone P (2008) A Multiagent Approach to Autonomous Intersection Management. Journal of Artificial Intelligence Research, March 2008, 31:591–656.
15. Caprari G, Colot A, Siegwart R, Halloy J, Deneubourg J -L (2004) Building Mixed Societies of Animals and Robots. *IEEE Robotics & Automation Magazine*.
16. Open Dynamics Engine (ODE) in the Internet: www.ode.org.
17. Guerra R da S, Boedecker J, Yamauchi K, Maekawa T, Asada M, Hirosawa T, Namekawa M, Yoshikawa K, Yanagimachi S, Masubuchi S, Nishimura K (2006) CITIZEN Eco-Be! and the RoboCup Physical Visualization League. Micromechatronics Lectures – The Horological Institute of Japan.
18. ARToolKit. In the Internet: http://www.hitl.washington.edu/artoolkit.