EyeBot: A Family of Autonomous Mobile Robots

Thomas Bräunl
Department of Electrical and Electronic Engineering
Centre of Intelligent Information Processing Systems (CIIPS)
The University of Western Australia, Perth
http://www.ee.uwa.edu.au/~braunl

Abstract

We designed a new and versatile family of small autonomous mobile robots. All these robots share a common controller and operating system, both developed from scratch. The robots are powerful enough to perform simple image processing tasks on-board and in real-time. This enables us to perform a number of research projects for individual mobile robots and for groups of robots on these systems.

1 Introduction

While small, inexpensive mobile robots are an enormous motivation in teaching beginner level lab courses, we found most of them insufficient for serious research work. Due to shortcomings in mechanics and electronics, establishing higher level goals in autonomous vehicle guidance is virtually impossible with too simple robots. Therefore, we decided to develop a new system from scratch.

Our goal was the development of an extendable base platform (*EyeBot Controller*) for a mobile robot system. The system was developed around the key requirements of image processing. It therefore features a digital camera and an LCD graphics display. It should also provide a sufficient number of I/O ports for the connection of various sensors and actuators or any future extensions.

While most robot vision systems are either tethered or remote-controlled [1], on-board real-time vision is feasible for large and expensive mobile platforms. However, it seemed very difficult to implement real-time vision on a small and inexpensive system. *EyeBot* is a controller for mobile robot systems that accomplishes this goal. It is an 8.6 cm × 8.6 cm board built around the Motorola M68332 microcontroller with a number of input/output ports, connected to a digital camera and a graphics LCD display. *EyeBot* has been successfully used in the construction of several wheel-driven vehicles, a 6-legged walking machine, and a biped walker. It is currently considered for the project of a flying robot.

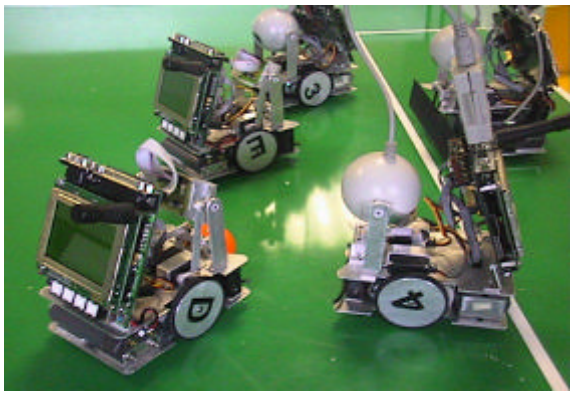

Fig 1. SoccerBot mobile robots

Although the controller runs at moderate speed (35 MHz), it is fast enough to compute basic image operations on a low resolution image in real time. E.g. image acquisition, Sobel edge detection and display on the LCD for a 80 × 60 grayscale image can be performed at a rate of about 10 frames per second.

EyeBot's graphics LCD is essential for interaction between the robot and the programmer. He needs to see the robot's view in order to set camera parameters or orientation. Although the camera provides gray scale images at 6bit and color images at 24bit at a higher resolution, the display can only show low resolution black/white images. This is sufficient as a feedback to the programmer when running the robot, but not for program development, which is done on a workstation using the tool *Improv*, which we developed as a separate project [2].

The robots displayed in Figure 1 are called SoccerBots for their application in the RoboCup robot soccer competition (see section 4). However, contrary to most other robot soccer teams, these robots are completely autonomous and can execute a variety of other tasks as well. Each robot is equipped with two shaft encoders, three infrared range sensors, and a digital color camera. We use differential steering for driving plus two additional servo actuators for tilting the camera and kicking the ball. A wireless transmission allows the robots to exchange information to other robots or a PC.

Fig 2. EyeBot Controller

2 EyeBot

The *EyeBot Controller* is a based on a Motorola M68332 microcontroller [3] with interfaces for our *EyeCam* digital CMOS camera and a 64×128 black/white graphics LCDisplay. We are operating on images of size 60×80, which we found have sufficient resolution for typical navigation tasks of small mobile robot systems. The controller is powerful enough to perform real-time onboard image processing, depending on the complexity of the operation.

The controller hardware was developed at Univ. Stuttgart and The Univ. of Western Australia, while Univ. Kaiserslautern (Germany) and Rochester Institute of Technology (USA) contributed to the system software.

A version of the *gnu* C-compiler and library has been adapted for *EyeBot*, so program development can be made in a high level language, using assembly routines for time-critical passages. In addition, a multi-threading scheduler has been developed, which is essential for robotics applications. The microcontroller's timing processor unit (TPU) is being used for servo control with pulse width modulation (PWM), for sound synthesis and sound playback, as well as the control of infrared distance sensors.

On top of the operating system, we developed the integrated tool *Rock&Roll* (**ro**bot **c**onstruction **k**it and **ro**bot **l**ocomotion **l**ink) [4]. This system allows a "click-and-connect" construction of robot control structures. In this data flow model, sensors are sources and actuators are sinks, both representing system-defined module boxes. User-defined control boxes can be added,

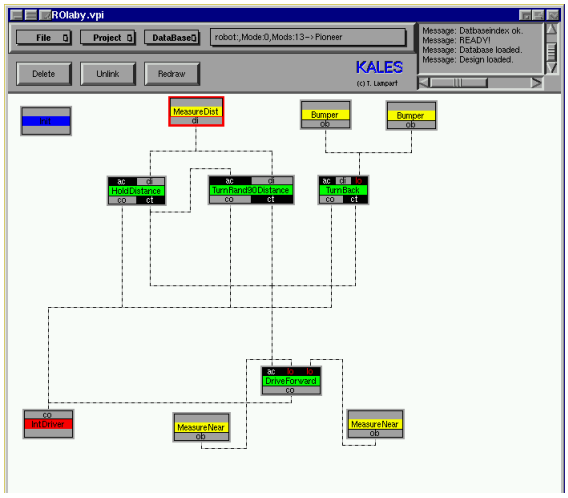

Fig 3. Rock and Roll Visual Prog. Tool

together with interconnection links between all modules, representing data flow.

3 Operating System

The operating system RoBIOS (**ro**bot **b**asic **i**nput **o**utput **s**ystem) has been written in C plus m68k assembly language, using the *gnu* C compiler and assembler tools [5]. RoBIOS comprises a small real time system with multi-threading, libraries for various I/O functions, and a number of demonstration applications.

The C low level text input and output routines have been adapted for *EyeBot*. This enables us to use the standard "clib" I/O library together with the *EyeBot* system for user application programs. E.g., a user can call `getchar()`, in order to read a key input and use `printf(..)`, in order to write text on the screen.

Special care has been taken to keep the RoBIOS operating system flexible among several different hardware configurations, because the same system is to be used for wheeled robots *and* for legged robots. Therefore, a hardware description table has been included into the system design.

The *EyeBot* operating system *RoBIOS* relies on the **h**ardware **d**escription **t**able *HDT*, in order to find out which hardware components are currently connected to the system. These hardware components can be sensors or actuators (motors or servos), whose control routines are already available in the general *RoBIOS* system. *HDT*

allows easy detection, initialization, and use of hardware components.

Entries in the *HDT* table define the current hardware configuration of the system. *RoBIOS* contains access routines to find and use the device drivers corresponding to the table entries. The *HDT* contains a list of all entries, together with their semantics. E.g. in the following, the first element line specifies an object of group `MOTOR`, called `MOTOR_RIGHT`, and a pointer to the motor data structure.

```
HDT_entry_type HDT[] =
{
  MOTOR,MOTOR_RIGHT,(void *)&motor0,
  MOTOR,MOTOR_LEFT,(void *)&motor1,
  PSD,PSD_FRONT,(void *)&psd1,
  INFO,INFO,(void *)&roboinfo,
  END_OF_HDT,UNKNOWN_SEMANTICS,(void
   *)0
};
```

Any program which needs to access a sensor or actuator object defined in the *HDT*, can now do so by defining a handle. Each group entry has to provide the handle data type and an initialization function.

4 Application: Robot Soccer

RoboCup is a five-a-side soccer tournament for mobile robots [6]. Different leagues exist for different size robots. Our robots qualify for the small-size league, but unlike most of the other teams, we do not make use of global sensors, especially we are not using a global overhead camera. Instead, we are interested in developing locally intelligent, autonomous mobile agents, which can be programmed for a number of tasks and are not restricted to a single specific application like the soccer game.

The special challenges of robot soccer are:

- real-time
  the ball has to be tracked quick enough and reached before the opponent
- cooperation
  five robots have to cooperate on the task of soccer with a common plan
- opponent
  each team has an opponent team with competing aims
- fault-tolerance
  each team has to operate if it loses a player, each player has to operate if it loses part of its functionality (e.g. a sensor) or has faulty or incomplete data

The main problem for each robot is to identify the ball (orange golf ball) and the opponents goal (color coded in yellow or blue). If possible, locating walls, team mates and opponent players is an advantage. For solving this problem, we use a digital camera on-board each robot together with on-board infrared distance measurement sensors. From the robot's color image, the ball position is determined and then translated via a table in x,y-coordinates. Since we are using local vision with local (on-board) image processing only, there is of course a limit in processing speed. Currently we achieve about 4 frames per second in 24bit color, 60×80 pixels resolution.

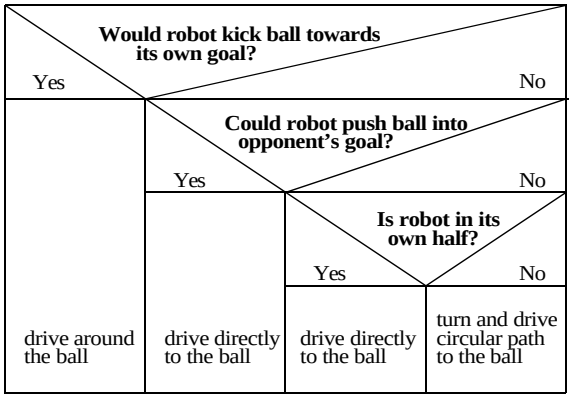

Fig 4. Ball approach strategy

Figure 4 shows the decision process involved once a robot has recognized a ball. It becomes clear that for a robot knowing its exact orientation is crucial (otherwise it might score an own goal), while knowing its exact position can be somewhat relaxed. Since we do not have global vision, a robot can easily lose its orientation when being pushed by an opponent. To overcome this problem, we currently incorporate a local compass system into each robot.

Assuming robot position/orientation and ball position are known, we can now calculate the robot's desired trajectory. For this purpose we use Hermite splines with two control points: the robot's position as start point and the ball's position as end point. The initial orientation is given by the robot, while the final orientation is selected to intersect the opponent's goal, so when the robot will finally hit the ball at the end of its trajectory, the ball will roll towards the goal.

Hermite splines are simple to calculate and have the desired property to pass all control points/orientations exactly. The Hermite blending functions $H_0 \ldots H_3$ with parameter u are defined as follows:

$$H_0 = 2u^3 - 3u^2 + 1$$
$$H_1 = (-2)u^3 + 3u^2$$
$$H_2 = u^3 - 3u^2 + u$$
$$H_3 = u^3 - u^2$$

The current robot position is then defined by:

$$P(u) = p_k H_0(u) + p_{k+1} H_1(u) + D p_k H_2(u) + D P_{k+1} H_3(u)$$

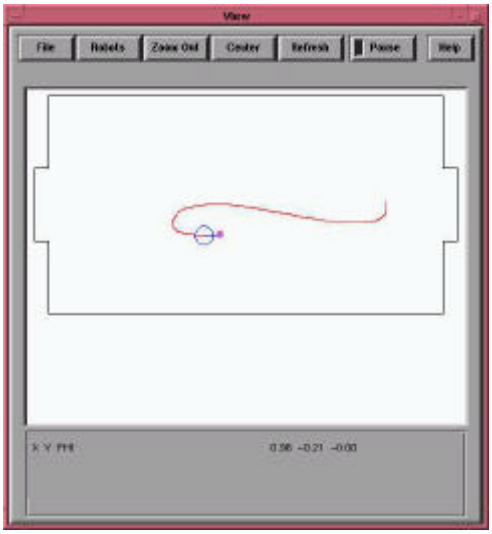

Fig 5. EyeSim screen display

A PI controller is used to calculate the linear and rotational speed of the robot at every point of its way to the ball trying to get it as close to the spline as possible. The control function is called a hundred times per second.

As shown in Figure 5, this strategy has first been designed and tested on the *EyeBot* simulator *EyeSim*, before running on the actual robot. Our approach is described in more detail in [7].

5 Application: Map Generation

Accuracy and speed are the two major criteria for map generation. Although the quad-tree representation seems to fit both criteria, it was unsuitable for our setup with limited accuracy sensors. Instead, we used the approach of visibility graphs [8] with configuration space representation. Since the typical environments we are using have only few obstacles and lots of free space, the configuration space representation allows more efficient obstacle mapping then the free space representation.

The task planning structure for the robot performing the mapping task is specified by the following structogram.

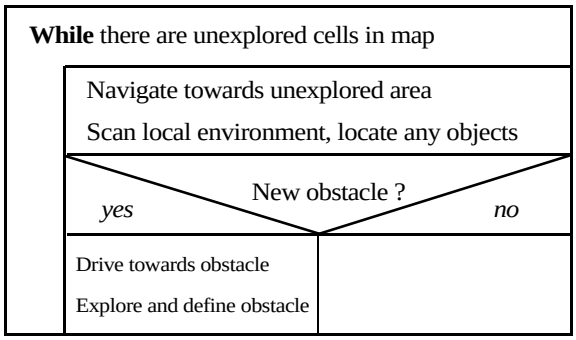

Fig 6. Map generation strategy

The robot starts in one point of its environment, then subsequently drives to each obstacle or wall in its near vicinity and enters it into the map. A grid structure is used for the map's data representation, in order to reduce memory space and smooth the generated map. After an obstacle is encountered, the robot tries to surround it as close as possible, using a variation of the *DistBug* algorithm [9].

Results of the mapping process are shown in Figure 7. On top the given environment is shown, on the bottom the generated map. Analysis of the results shows that the position error is within 20% of the robot's size. This is a very good result considering the robot's low-precision infrared sensors. The algorithm does not make any assumptions about straight walls or right angle corners. If our environment was restricted to these two assumptions, even more accurate maps could be generated.

6 Application: Omni-Directional Driving

Another mobile robot design using the same *EyeBot* controller is *Omni*, an omni-directional (holonomous) mobile robot.

Omni has four driven wheels, built after the *Mecanum* principle. Each wheel's driving sur-

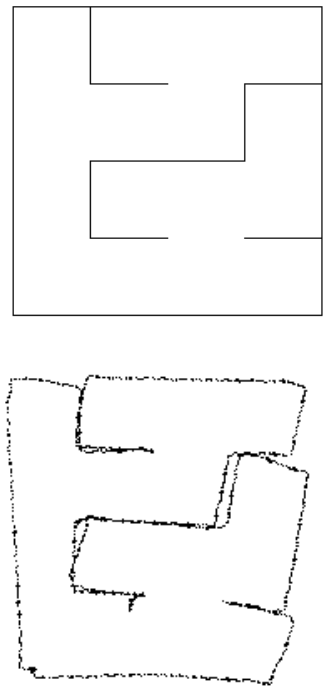

Fig 7. Mapping results

face is covered by free rollers, set at an angle of 45 degrees. Applying different speeds to the four individual wheels will then allow the robot to drive forward/backward, slide to the left/right or turn on the spot.

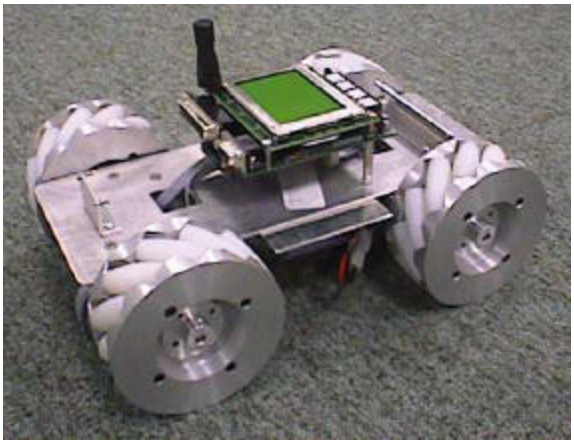

Fig 8. Omni-directional vehicle

7 Application: Walking Machines

We have constructed a number of legged robots over the years. These were mostly six-legged robots (see Figure 10). Six-legged robots have the advantage that they hardly ever fall over, so balance can almost completely be ignored as long as the walking patterns stay within a certain range. It turned out that a single EyeBot controller had sufficient computing power to drive six-legged robots with 12 degrees of freedom (two per leg).

We constructed two biped walking robots as a challenge. In order to keep cost low, we used servos instead of higher quality DC motors. Each android walker has four infrared proximity sensors in its feet (as feedback for surface contact) and two acceleration sensors in its upper body (as feedback for balancing and body attitude).

Balancing is the fundamental problem that had to be solved first by a feedback loop with active control. After that we examined different walking strategies.

Fig 9. Bipedal walking machine

8 EyeBot Robot Family

Recent robot developments based on the *EyeBot* controller are the two-wheeled *SoccerBot* vehicles, a six-legged walking machine *Crab*, two-legged android robots *Johnny Walker* and *Jack Daniels*, and the omni-directional four-wheel-driven *Omni*. Constructions of underwater and flying robot are currently being considered.

All these robots use the same *EyeBot* controller with the same *RoBIOS* operating system. The only difference is in the hardware description table, where robot specific settings and drivers are listed, e.g. different mechanics, motors, and sensors.

All robots use a digital camera and three infrared distance measurement sensors to the front, left and right. Motor-driven robots (*SoccerBot*: 2 motors, *Omni*: 4 motors) have encapsulated encoders associated with each motor, while servo-driven robots (*Crab*: 12 servos, *Johnny+ Jack*: 9 servos), unfortunately do not have these feedback sensors.

Fig 10. EyeBot family

Summary and Future Research

We have discussed the *EyeBot* mobile robot family. The *EyeBot* controller is a hardware/software platform for mobile robots of any kind, allowing real-time vision control. Current research concentrates on an integrated multi-robot console allowing wireless control and monitoring of a group of robots. Future research will be on intelligent behaviour-based systems and intelligent group behaviour of mobile robots.

More information about these projects is available on the Internet:

`www.ee.uwa.edu.au/~braunl/eyebot/`

Acknowledgments

The author wants to acknowledge the work of his colleagues and students who participated in robotics projects. These are especially Ivan Neubronner and Klaus Sautter for electronics design, Richard Meager and Jörg Henne for robot mechanics, Klaus Schmitt and Thomas Lampart for operating system software and programming tools, Birgit Graf for the soccer application program and Nicholas Tay for the map generation application program.

References

1. H. Bayer, Th. Bräunl, A. Rausch, M. Sommerau, P. Levi, *Autonomous Vehicle Control by Remote Computer Systems*, Proceedings of the 4th International Conference on Intelligent Autonomous Systems, IAS–4, Karlsruhe, March 1995, pp. 158–165 (8)
2. Th. Bräunl, S. Feyrer, W. Rapf, M. Reinhardt, *Parallele Bildverarbeitung*, Addison-Wesley, 1995
3. Th. Harman, *The Motorola MC68332 Microcontroller*, Prentice Hall, 1991
4. P. Levi, M. Muscholl, Th. Bräunl, *Cooperative Mobile Robots Stuttgart: Architecture and Tasks*, Proceedings of the 4th International Conference on Intelligent Autonomous Systems, IAS–4, Karlsruhe, March 1995, pp. 310–317 (8)
5. The GNU Project, *GNU Documentation*, online, Delorie Software, www.delorie.com/gnu/docs/
6. M. Asada (Ed.), *RoboCup-98: Robot Soccer World Cup II*, Proc. of the second RoboCup Workshop, Paris, July 1998
7. T. Bräunl, B. Graf, *Robot Soccer with Local Vision*, 5th Pacific Rim International Conference on Artificial Intelligence, Nov. 1998, Singapore, pp. 14–23 (10)
8. P. Sheu, Q. Xue (Ed), *Intelligent Robotic Planning Systems*, World Scientific Publishing, Singapore, pp. 111-127, 231-243
9. I. Kamon, E. Rivlin, Sensory-Based Motion Planning with Global Proofs, IEEE transactions on Robotics and Automation, vol. 13, no. 6, Dec. 1997, pp. 814-822 (9)