Autonomous Mobile Robots with On-Board Vision and Local Intelligence

Thomas Bräunl, Birgit Graf
Department of Electrical and Electronic Engineering, CIIPS
The University of Western Australia
Nedlands, Perth, WA 6907, Australia
http://www.ee.uwa.edu.au/~braunl

Abstract

We designed a family of completely autonomous mobile robots with local intelligence. We developed our own controllers with a variety of digital/analog I/O facilities and our own operating system RoBIOS, which allows maximum flexibility. Our robots have a number of on-board sensors, including vision, and do not rely on global sensor systems. The on-board computing power is sufficient to analyse several color images per second. This enables our robots to perform several different tasks such as navigation, map generation, and intelligent group behavior.

1 Introduction

After experience with large and heavy commercial mobile robot systems [3], we decided to develop completely new mini-robots which should be suitable for a variety of tasks in research and education. The robot mechanics were designed to comply with the RoboCup [1] robot soccer rules in order to be able to play in this competition, but should still be versatile for a number of different perception/action tasks and not be too specialised for any event in particular.

Each robot is equipped with two shaft encoders, five infra-red range sensors, and a digital color camera. We use differential steering for driving plus two further actuators for tilting the camera and moving the front bar, e.g. for kicking a ball. A wireless transmission allows the robots to exchange information regarding their positions, sensor data, as well as their intended actions and plans. Each robot is told its starting position and uses its shaft encoders plus infra-red sensors to keep track of its current position and orientation. We do not use a global positioning system or any global sensors.

We incorporated a digital color camera and a graphics display to our basic controller system, based on a Motorola 68332 controller (32 bit). All image processing is done on-board without using remote computing power.

We believe that most essential mobile robot research and education can be performed using small mobile robots with vision, as opposed to large, heavy, expensive and potentially harmful robot systems.

2 EyeBot Controller

The *EyeBot* controller is the central piece of hardware we developed for a number of mobile robot projects. This means the controller has to be flexible enough to account for the needs of e.g. a 4-wheel-drive omni directional vehicle and a 9-degree-of-freedom biped walking machine.

2.1 Design Criteria

The controller was designed to perform onboard image processing. This means it has to have an interface to a digital camera and also the computing power to process several images per second.

To the best of our knowledge, *EyeBot* is so far the only mobile robot controller for small size mobile robots which allows onboard vision (the controller size is 8.7cm $\times$ 9.9cm). All other systems use either the "remote-brain" approach, e.g. a video signal is relayed to a host workstation and driving commands are sent back - tethered or remote-controlled, or they do not use vision at all [2].

We added a graphics LCD, in order to allow the programmer to see the camera data and adjust parameters without the need for additional equipment or a link to a host system. Although the camera provides color images at medium 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 should be done on a workstation using our interactive image processing tool *Improv* [4].

A set of four buttons allows the programmer to set program parameters and load/store programs from ROM and RAM.

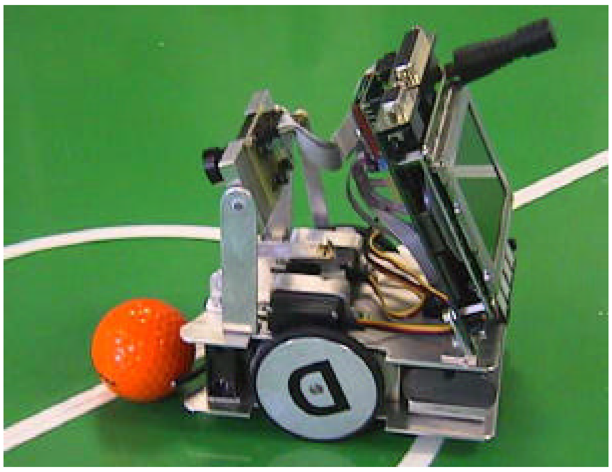


Fig 1. EyeBot robot with local intelligence

2.2 Performance Criteria

The controller has to perform a number of tasks in time-shared parallel mode. This includes handling timer interrupts, sensor input and actuator output (DC motors and servos). All image processing routines run concurrently to these background tasks.

A low resolution image is sufficient for most mobile robot applications. We chose a resolution of 60×80 pixels. Running at 35 MHz, the controller can handle about 16 frames per second (fps) for grayscale and about 4 fps for color images. This number will be reduced depending on the complexity of the application program’s image operations. E.g. image acquisition for grayscale, Sobel edge detection and display on the LCD can be performed at a rate of more than 10 frames per second.

2.3 Programming Environment

A version of the *gnu* C-compiler and library [5] 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 infra-red 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) [6]. This system allows a "click-and-connect" construction of robot control structures. In its data flow model, sensors are sources and actuators are sinks, both representing system-defined module boxes. User-defined control boxes can be added, together with interconnection links between all modules, representing data flow.

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. 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 C I/O library *clib* 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.

2.4 Mobile Robot Applications

EyeBot has been successfully used in the construction of two-wheel-driven vehicles, 4-wheel-driven omni-directional vehicles, 6-legged walking machines, and biped walkers. It is currently being considered for the project of a flying robot.

In order to accommodate this variation in robot mechanics and sensor/motor configurations, special care has been taken to keep the RoBIOS operating system flexible. A **h**ardware **d**escription **t**able (HDT) has been included in the system design, which is used by the operating system to detect which hardware components are currently connected to the system and to provide device drivers. 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 these components.

3 Sensing

All wheel-driven robots of the *EyeBot* family use the following sensors:

- shaft encoders
- infra-red distance measurement sensors
- compass module
- digital camera

Plus communication between the robots.

The legged robots of the *EyeBot* family use additional sensors:

- infra-red proximity switches in the feet
- acceleration sensors in torso

3.1 Shaft Encoders

The most basic feedback is generated by the motors' encapsulated shaft encoders. This data is used for three purposes:

- PI controller for individual wheel to maintain constant wheel speed
- PI controller to maintain desired path curvature (i.e. straight line)
- Dead reckoning to update vehicle position and orientation

The controller's dedicated timing processor unit (TPU) is used to deal with the shaft encoder feedback as a background process.

3.2 Infra-red Distance Measurement

Each robot is equipped with three infra-red sensors to measure the distance to the front, to the left and to the right. This data can be used to:

- avoid collision with an obstacle
- navigate and map an unknown environment
- update internal position in a known environment

Since we are using low cost devices, the sensors have to be calibrated for each robot and, due to a number of reasons, also generate false readings from time to time. Application programs have to take care of this, so a level of software fault tolerance is required.

3.3 Compass Module

The biggest problem in using dead reckoning for position and orientation estimation in a mobile robot is that it deteriorates over time, unless the data can be updated at certain reference points. A wall in combination with a distance sensor can be a reference point for the robot position, but updating robot orientation is very difficult without additional sensors.

A similar problem exists in legged robots, since they do not have direct feedback sensors like shaft encoders and their turning speed may vary greatly with the surface structure (e.g. concrete or carpet).

In these cases, a compass module which senses the earth's magnetic field is a big help. However, these sensors are usually only correct to a few degrees and may have severe disturbances in the vicinity of metal. So the exact sensor placement has to be chosen carefully.

3.4 Digital Camera

We developed a camera module based on a CMOS image chip. This gives a resolution of 60×80 pixels in 24bit color. Since all image acquisition, image processing, and image display is done onboard the *EyeBot*, there is no need to transmit image data.

3.5 Infra-red Proximity Switch

In the past, we used infra-red proximity switches also for wheeled robots, but later left them out in favor of the more powerful distance measurement sensors.

In the biped walking robots, we use two infra-red proximity switches in each foot as a feedback whether the robot touches the ground.

3.6 Acceleration Sensor

We currently use acceleration sensors only in the biped walking robots. These sensors are mounted perpendicular for two axes and return the current robot body posture. A PID controller uses this as a feedback to make the robot balance.

Since we use rather cheap servos in the construction of the biped walker, there is considerable jitter noise in the standing robot, which is of course directly reflected in the raw sensor readings. Since this noise should not effect the robot control, we use digital filters to remove this noise from the sensor signal before it is used for feedback.

3.7 Inter-Robot Communication

While the wireless communication network between the robots is not exactly a sensor, it is nevertheless a source of input data to the robot from its environment. It may contain sensor data from other robots, parts of a shared plan, intentions from other robots, or commands from other robots or a human operator.

4 Behaviour Modules

In the following we discuss a number of behaviour modules while concentrating on the perception aspects. The sample application is playing robot soccer. The modules can be re-used for other individual or group robot tasks. In particular the position tracking and obstacle avoidance modules are being used for exploration and mapping tasks of other robot application projects.

4.1 Detection of Colored Objects

We analyze the visual input from the on-board digital camera in order to detect important objects on the field. The ball or the opponent's goal are detected using their color. These colors are taught to the robot before the game is started. The robot has to be placed in front of an object, then the mean HSV-values of the center region on the screen are calculated and saved for the object's future identification.

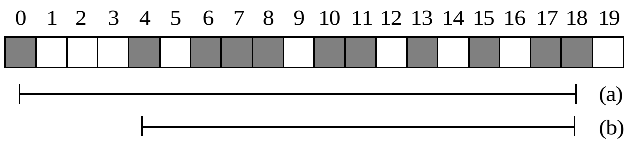


Fig 2. Analysing color image line

The pixel lines of the input picture are continuously searched for objects whose mean color value compared to the trained color value of the ball is below a certain threshold and which have the desired size (trying to distinguish the ball and the yellow goal). In Figure 2 a simplified line of pixels is shown, object pixels are displayed in grey, others in white. The implemented algorithm initially searches for ball-colored pixels at each end of a line (see region (a): first = 0, last = 18), then the mean color value is calculated. If it is below the threshold, the ball has been found, otherwise the region will be narrowed, attempting to find a better match (see region (b): first=4, last=18). The algorithm stops as soon as the size of the analyzed region becomes smaller than the desired size of the object. In the line displayed in Figure 2 a ball with a size of 15 pixels is found after two iterations.

4.2 Distance Estimation from Camera Data

Once the object has been identified in an image, the next step is to translate image coordinates into global coordinates in the robot's environment. This is done in two steps:

- First, the position of the ball as seen from the robot is calculated from the given pixel values, assuming a fixed camera position and orientation. This calculation depends on the *height* of the object in the image. The higher the position in the image, the further an object's distance from the robot.
- Second, given the robot's current position and orientation, the local coordinates are transferred into global position and orientation on the field.

Since the camera is looking down at an angle, it is possible to determine the object distance from the image coordinates. In an experiment (see Figure 3), the relation between pixel coordinates and object distance in meters has been determined. Instead of using approximation functions, we decided to use the faster and also more accurate method of lookup tables. This allows to calculate the exact ball position in meters from the screen coordinates in pixels and the current camera position/orientation.

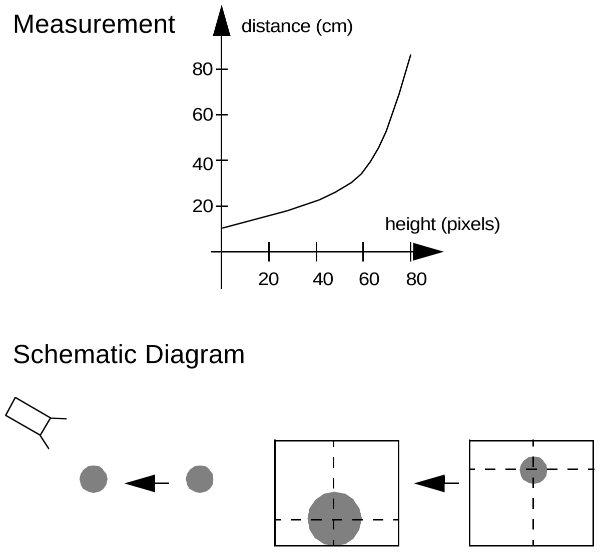


Fig 3. Relation between object height and distance

The distance values were found through a series of measurements, for each camera position and for each image line. In order to reduce this effort, we only used three different camera positions (up, middle, down), which resulted in three different lookup tables.

Depending on the robot's current status, the appropriate table is used for distance translation. The resulting relative distances are then translated into global coordinates using polar coordinates.

An example output picture on the robot LCD can be seen in Figure 4. The lines indicate the position of the detected ball in the picture while its global position on the field is displayed in centimeters on the right hand side.

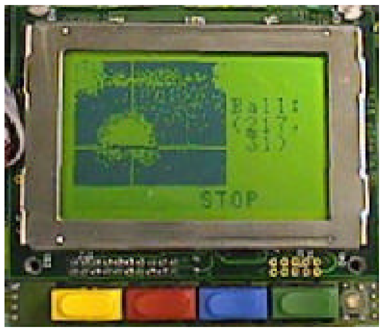
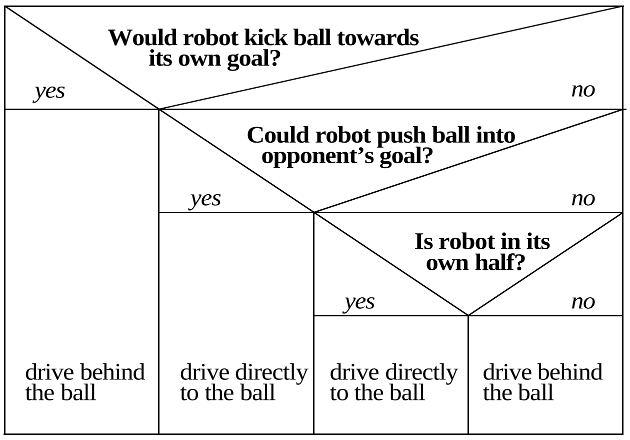


Fig 4. LCD output after ball detection

4.3 Trajectory Planning

After detecting the global position of the ball on the field, the robot starts driving towards it. To make the decision of where the selected robot drives, we had to develop several driving algorithms. The appropriate driving action is chosen depending on the position of the ball and the robot's own position (Figure 5).



Would robot kick ball towards its own goal?
yes
no
Could robot push ball into opponent's goal?
yes
no
Is robot in its own half?
yes
no
drive behind the ball
drive directly to the ball
drive directly to the ball
drive behind the ball

Fig 5. Ball approach strategy

The necessary decision factors are calculated as follows:

Would robot kick ball towards its own team's half?

With φ being robot orientation and β being the angle between robot orientation and ball position the absolute value of the robot orientation after driving a curve to the ball is $\varphi' = \varphi - 2\beta$. The absolute value of this angle must not be more than 90 degrees.

Could robot push ball into opponent's goal?

With *ballx* and *bally* being the coordinates of the ball on the field, *length* being the length of the field and φ' being the above described heading of the robot after driving directly a curve to the ball, $ydist = (length - ballx) \cdot \tan(\varphi') + bally$ is calculated as the distance where the robot would hit the goal line if it drove a curve to the ball and then towards the goal in a straight line. If the absolute value of *ydist* is smaller than half the width of the goal, the robot is able to push the ball directly into the goal.

Is robot in its own half?

If the *x* value of the robot position is less than half the length of the field, the robot is still in its own team's half.

4.4 Driving Actions

After completing the trajectory planning, the robot selects the appropriate driving action for execution. Here, the robot can either drive directly towards the ball (if the ball is roughly in line with the opponent's goal) or drive around the ball (in order to kick it in the opposite direction, e.g. avoiding an own goal). Figure 6 shows all different cases.

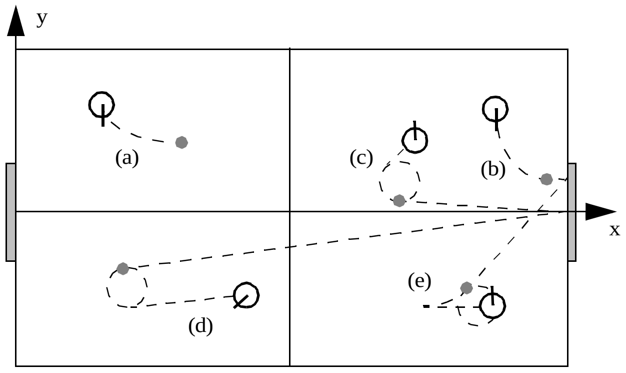


Fig 6. Ball approach cases

Drive directly to the ball (Figure 6 a,b):

With *localx* and *localy* being the local coordinates of the ball seen from the robot, the angle to reach the ball can be set directly as $\alpha = -\operatorname{atan}\left(\frac{localy}{localx}\right)$. With l being the distance between the robot and the ball, the distance to drive in a curve is given by $d = l\alpha\sin(\alpha)$

Drive around the ball (Figure 6 c,d,e):

If a robot is looking towards the ball but at the same time facing its own goal, it can drive to a circle with a fixed radius that goes through the ball. The radius of this circle is chosen arbitrarily and was defined to be 5 cm. The circle is placed in a way that the tangent at the position of the ball also goes through the opponents' goal. The robot turns on the spot until it faces this circle, drives to it in a straight line and behind the ball on the circular path.

Driving along a spline curve:

An interesting alternative driving routine are splines, since they generate a smooth path and avoid turning on the spot, so they can be used to generate a faster path.

Given the robot position P_k and its heading DP_k as well as the ball position P_{k+1} and the robot's destination heading DP_{k+1} (facing towards the opponent's goal from the current ball position), it is possible to calculate a spline which for every fraction u of the way from the current robot position to the ball position describes the desired location of the robot.

The Hermite blending functions $H_0 \ldots H_3$ with parameter u are defined as follows:

$$H_0 = 2u^3 - 3u^2 + 1$$

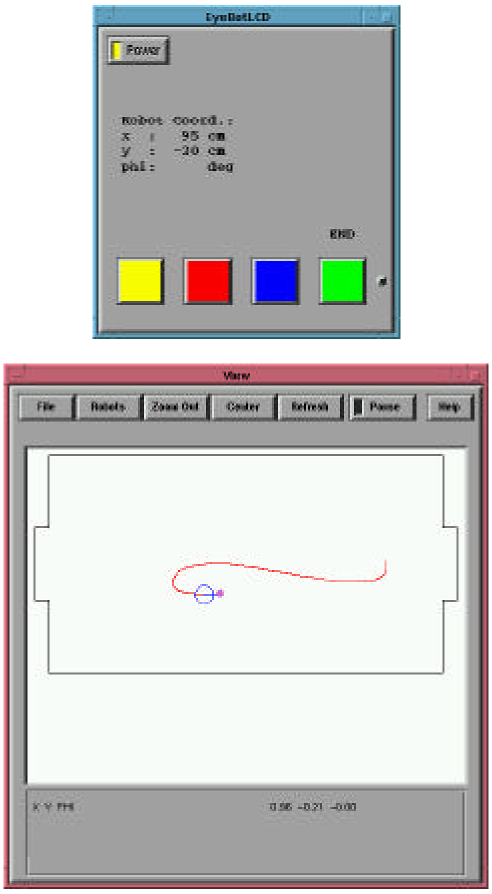
$$H_1 = (-2)u^3 + 3u^2$$

$$H_2 = u^3 - 3u^2 + u$$

$$H_3 = u^3 - u^2$$

The current robot position is then defined by:

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



```
Robot coord.:
x  :   95 cm
y  :  -20 cm
phi:      deg
```

```
X Y PHI                    0.96  -0.21  -0.00
```

Fig 7. EyeSim screen display

A PID 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 driving function `DriveSpline(start, end)` has been added to the RoBIOS library. The robot's speed is constantly updated by a background process that is invoked 100 times per second. If the ball cannot be detected any more (e.g. the robot had to drive around it and lost it out of sight), the robot keeps driving to the end of the original curve. The new driving command is being issued as soon as the (moving) ball is recognised at a different position.

This strategy has first been designed and tested on the *EyeBot* simulator *EyeSim* (Figure 7), before running on the actual robot.

5 Summary and Future Research

EyeBot robot developments include not only the wheeled robots with differential steering, but also several different 6-legged and biped walking machines (Figure 8).

All these robots are using the same controller type and the identical operating system, individually adapted to each robot's sensor/actuator configuration by the concept of a hardware description table. A number of sensors used for different types of mobile robots has been discussed.

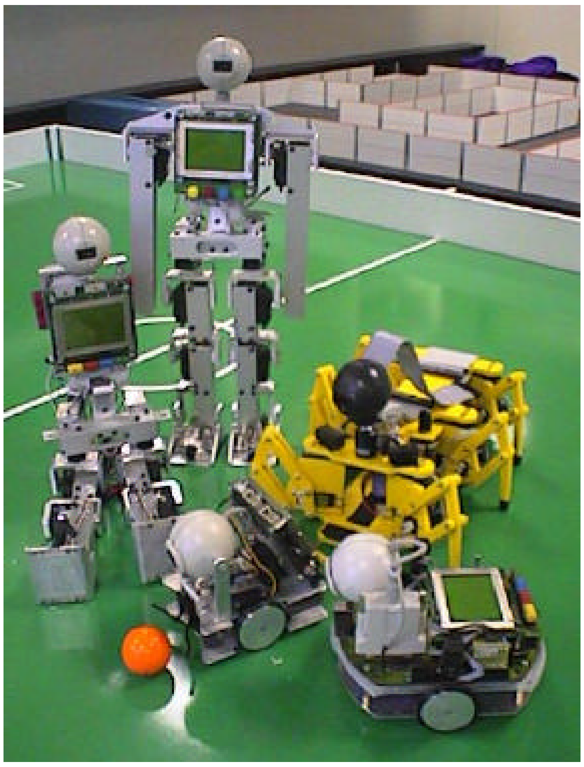


Fig 8. EyeBot robot family

We have presented *EyeBot*, a platform used for autonomous mobile robots with local intelligence, *EyeSim*, a simulator for *EyeBot*, and perception/planning/action strategies for playing robot soccer. *EyeBots* are currently being used for a number of intelligent agent projects, including behaviour-based approaches for implementing complex group behaviour.

Acknowledgments

The authors acknowledge the work of all students who participated in this project, especially Elliot Nicholls, Nicholas Tay, Thomas Lampart, and Klaus Schmitt. *EyeBot* is an ongoing joint project between Univ. Stuttgart, Rochester Institute of Technology, Univ. Kaiserslautern, and UWA Perth. The authors would like to express special thanks to Richard Meager and Ivan Neubronner from the E&E Eng. Workshops of UWA for re-designing and building the robot mechanics and controller hardware.

References

1. M. Asada, *RoboCup-98: Robot Soccer World Cup II*, Proc. of the second RoboCup Workshop, RoboCup Federation, Paris, July 1998
2. 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)
3. T. Bräunl, M. Kalbacher, P. Levi, G. Mamier, *CoMRoS: Cooperative Mobile Robots* Stuttgart, Proc. 13. Nat. Conf. on Artificial Intelligence, AAAI Press, Portland OR, August 1996
4. T. Bräunl, *Improv and EyeBot – Real-Time Vision on-board Mobile Robots*, 4th Annual Conference on Mechatronics and Machine Vision in Practice (M2VIP), Toowoomba QLD, Australia, Sep. 1997, pp.131–135 (5)
5. The GNU Project, *GNU Documentation*, online, Delorie Software,
   www.delorie.com/gnu/docs/
6. 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)