A Framework for Cognitive Agents

Joshua D. Petitt and Thomas Bräunl

Abstract: We designed a family of completely autonomous mobile robots with local intelligence. Each robot has a number of on-board sensors, including vision, and does not rely on global positioning systems The on-board embedded controller is sufficient to analyze several low-resolution color images per second. This enables our robots to perform several complex tasks such as navigation, map generation, or providing intelligent group behavior. Not being limited to playing the game of soccer and being completely autonomous, we are also looking at a number of other interesting scenarios. The robots can communicate with each other, e.g. for exchanging positions, information about objects or just the local states they are currently in (e.g. sharing their current objectives with other robots in the group). We are particularly interested in the differences between a behavior-based approach versus a traditional control algorithm at this still very low level of action.

Keywords: Adaptive control robot group, autonomous agent, behavior-based systems.

1. INTRODUCTION

We have worked with small autonomous mobile robots for a number of years [1]. Their actuator and sensor hardware has evolved over the years in along with the operating system and the control software. Although our robots can be applied to many different applications, we have used them for playing robot soccer without global sensors [2] in a number of competitions over the years. Each competition "campaign" was programmed by a number of students and, in the past, followed traditional hierarchical software architecture.

In this paper, however, we present a different, behavior-based approach for autonomous robots, which can be used to implement arbitrary robot application scenarios.

2. ROBOT HARDWARE

Each of our robots comprises a micro-controller system (*EyeBot*) [3] interfaced to a digital color camera, distance sensors, shaft encoders, compass, DC motors, servos, wireless module, and a graphics display. All image processing is done on-board. We are especially interested in research on autonomous mobile systems, so we took this clearly disadvantaged robot soccer approach instead of the traditional approach, involving the use of a global overhead camera.

We are using low-resolution images (160x120 in 24bit color) and have to restrict image processing complexity due to limited processor computation rate. Frame grabbing can be done at 30 frames per second (fps). However, depending on the detection algorithm used, this is normally reduced quite significantly. Self-localization is an important task for our robots, since we do not use a global positioning system, such as one based on the use of an overhead camera. Instead, we rely on dead reckoning from a specified starting position and orientation. However, a robot will soon lose track of its exact position and orientation due to wheel slippage or - much worse - collision with another robot. Therefore, we integrated a digital compass into our robots.

In the robot soccer application, orientation is more important than position, because it guarantees that a robot is heading for the right goal. The robot's local infrared sensors can update its position, whenever it gets close to one of the sidewalls or to a corner.

The camera mechanics were changed from a tilting to panning operation in order to improve ball tracking. The camera can be moved sideways at a higher speed than the whole robot can turn, so this allows the tracking of balls which are moving faster across the robot's field of view, than would otherwise be possible with a static camera mount.

An innovative wireless protocol allows a group of autonomous agents to communicate with each other. Each agent can become the "master" and new incoming agents or leaving agents are handled automatically by this virtual token ring approach [4].

Manuscript received Febuary 28, 2002; accepted June 24, 2002.

Thomas Bräunl is with the Center for Intelligent Information Processing Systems, University of Western Australia. (email: braunl@ee.uwa.edu.au).

Joshua Petitt is with the School of Mechanical Engineering, University of Western Australia, 35 Stirling Highway, CRAWLEY, WA 6009. (email: petitj01@mech.uwa.edu.au).

3. DISTRIBUTED BEHAVIOR MODEL

Bräunl at Univ. Stuttgart proposed a behavior-based architecture for mobile agents in 1995 in [5]. Lampert/Bräunl successfully implemented this system, by the name of "Rock&Roll", in 1997 at UWA. It allowed **robot-independent** programming by selecting behaviors from a library list (Fig. 1).

Module libraries were written for different robots (In this case: EyeBot, Soccerbot and the Pioneer I robot), so all robot-specific implementation details are hidden from the application "programmer". Actually, no programming, in the real sense of the term, takes place at this user level, since modules are simply selected from a list and linked onto a canvas. Automatic code generation is then performed by the Rock&Roll system, depending on the configuration of the actual robot model.

The modules employed belong to the following groups: sensor (yellow), compute (green), or actuator (red) and are all executed in parallel. Drawing links between the modules defines the actual data flow between the modules. This implicitly determines not only the data exchange, but also the synchronization of the modules, since these modules can "sense" new data coming in and use this information for their internal calculations.

What the Rock&Roll system does not have is a clear distinction between behaviors and arbitration. However, we intend to solve this problem with our new general "behavior toolkit", as shown in Fig. 2. This new toolkit provides the basic framework required for implementing any behavior-based system, whether it follows a centralized or a distributed control model. One application of this framework is the "EyeMind" framework, proposed by Petitt, which is discussed in more detail in the following sections.

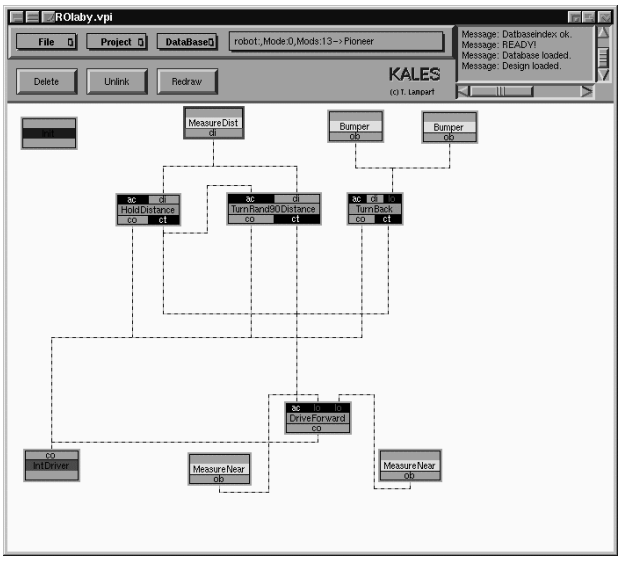


Fig. 1. Rock&Roll, Lampert/ Bräunl 1997.

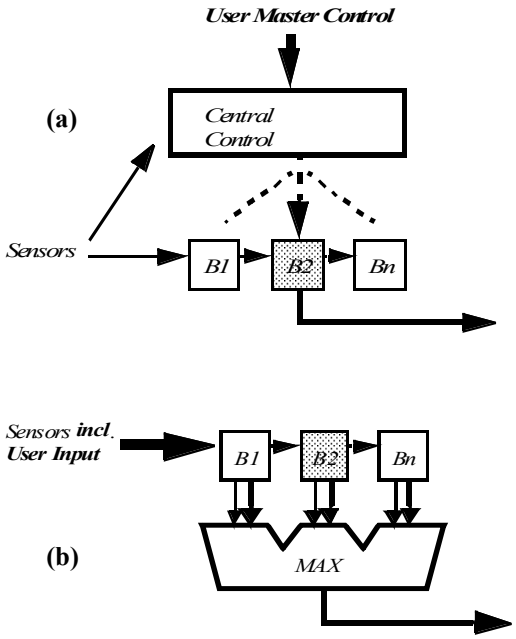


Fig. 2. Behavior Toolkit with
(a) centralized or (b) local control.

4. THE EYEMIND FRAMEWORK

The EyeMind framework is a set of objects, written in C++, which can be extended to create complex behavior networks. The objects, their functions and their application to our robot soccer team "CIIPS Glory" will now be described.

Many robots have been built around the "sense-think-act" paradigm, with varying degrees of success. This follows the path of classical artificial intelligence, since a plan is created from a set of inputs or data stored in memory. Other researchers have discarded this idea and have created robots with what is essentially a "sense-act" paradigm. These robots are termed "behavior-based" and have proven to be successful in dynamic environments. Other hybrid approaches have also been implemented, again with varying degrees of success. The commonalities between the two conventions are that in both models, the agent has *intentionality* [6] and the agent is *embodied* [7]. Intentionality refers to the desire of the agent to manipulate its surroundings, while embodiment means that the agent only has limited knowledge about its environment, and that it can act on, or change, its surroundings. In the previous sentence, the word "knowledge" is used rather loosely. There are still many questions about the true nature of knowledge and therefore it lacks a firm definition. In this instance, let's assume that knowledge is any information that the agent uses to control its actions. This means that knowledge could of neuron weights for the connectionists, and/or direct sensor readings

(analog or digital in representation) for the behaviorists.

The drawback to these three approaches, classical, connectionist, and behaviorist, is that the agent must either

- Know explicitly how to accomplish every task it is given, as well as being able to carefully plan each task.
- Know nothing explicit about its tasks, and just rely on learning the correct way to accomplish them from an expert or teacher.
- Know nothing; just react to the environmental stimulus. Only tasks for which the robots are "engineered" to perform can be accomplished (i.e. wander and avoid obstacles).

A zero-sum physical game, like soccer, provides a perfect way of exposing the deficiencies of each approach. The classical approach is often too slow to keep track of the progress of the game, and so a plan, which is valuable at one moment in time, can easily become less effective or useless later on. From this point further, classical AI will be referred to as the "reasoning" approach and the behavior-based approach will be termed the "reacting" approach. The connectionist approach, which in many ways is a reactive approach combined with complicated mappings of inputs to outputs, can be successful in negotiating dynamic environments (i.e. if there is no plan, then there is no plan to be changed), but cognition and reasoning are completely left out. The problem is that neither approach truly captures what most humans would consider *intelligent* behavior (the behaviorists might argue about this point, but I think that many would agree that the mental abilities of ants, fish, or even those of birds are too crude for them to have the ability to succeed in a team game). The remainder of the discussion will be devoted to resolving the differences between the two approaches and to show that both of these approaches are needed for the success of an intelligent agent. We will also present a model that incorporates both approaches.

The real underlying difference, in the reasoning versus the reacting model, is that the reasoning approach is roughly modeled after the cognition that takes place in our forebrain, or our conscious mental activities. Conversely, the reacting approach is modeled after the lower brain and brainstem or our unconscious mental activities, which primarily include motor control. Freud [8], in an attempt to understand human thinking, developed a model of the human psyche, which encompasses both the unconscious and conscious mental activates. He asserted that the mind could conceptually be divided into three parts, the id, ego and super-ego. The point must be made that he did not consider each of these to be distinct

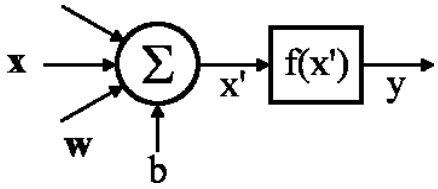


Fig. 3. Simple summing model of a neuron.

processes, rather that they communicate and affect each other.

Other researchers [9] have used three layer architectures in robot designs. They have assigned names to these layers, such as the *skill*, *sequencing* and *planning layers* or the *controller*, *sequencer* and *deliberator*. The three-layered architecture described here uses the descriptors "id", "ego" and "super-ego", following the nomenclature of Freud's model of the mind.

4.1 The unconscious id

The Id class provides the functionality required for instantiating and managing all the sensors and actuators used by the agent. The Id stores lists of references for active actuator and sensor objects, which from this point on will be collectively referred to as input/output (I/O) objects. The Id also manages the current *behaviors* of the robot. Before continuing, further explanation will be given about behaviors, sensors, and actuators and how they are stored. The fundamental data structure used in our architecture is a doubly linked List class composed of Node elements. Derived from the Node class is the LockNode class. This provides the functionality to 'lock' a node, so that it can only be accessed by the object that possesses the correct key. This enables multiple threads of control to access shared resources.

Behaviors - A behavior can be defined as a mapping of sensor input to motor output. By combining simple behaviors, a robot can be engineered to assume a more complex appearing behavior, which could be perceived by an observer as "intelligent". Robots that do simple tasks such as following light beams, to ones that can perform more complex problems like solving mazes and navigating through dynamic environments, can all be constructed, sometimes containing no processing unit at all [10]. What they all have in common is a feedback loop between their sensors and actuators and the ability for their behaviors to be suppressed or excited. The Glory architecture defines an abstract class called a Behavior, from which all other behaviors are derived. A generic behavior has an excitation input and an internal threshold value. The generic behavior also has an output, which can be a single value or a vector of values. When the behavior is excited, the excitation value generated is added to the current excitation value, which is itself the result of previous excitation

events. If the new excitation value is greater than a threshold value, then the behavior is activated and it 'fires'. The firing of the behavior can take many forms, from creating an output signal for an actuator, requesting sensor input, to triggering another behavior.

The Behavior object is almost identical to the model of a neuron used in many artificial neural networks (ANN) (see Fig. 3 for a diagram). The essential difference between these two types of objects is the time delay between the arrival of the input signal and the generated output. For a typical ANN, the input signal is propagated through the layers in a synchronous fashion and the function almost immediately produces output that reflects the state of the inputs at that moment. With a Behavior there is a time delay between the input to the Behavior and the change in the output being sensed. In other words, *a behavior is a function of both the inputs and of time*, or a response of a dynamic system to a time varying input. The similarity between a behavior and a neuron allows for much of the functionality to be contained in a common base class and facilitates information flow from a neural network directly to a behavior or vice-versus. Therefore, many of the learning algorithms that have been developed for ANNs can be applied to behavior networks.

This model may be arranged as a feed-back controller or as a subsumption unit. Note that by allowing representations for positive and negative infinity (defined as the largest and smallest floating point numbers allowed by the machine), then the behavior may be suppressed or excited by only one input. An ambiguity would exist if a behavior were to be both excited and suppressed by an "infinite" signal, thus we have chosen for the behavior to be suppressed.

The Id class retains a list of up to sixteen 'root' behaviors. Each of these behaviors are excited by the timer processor unit (TPU) at set intervals. The TPU interrupts the CPU and causes the CPU to execute the list of root behaviors. The root behaviors then either do nothing, or execute their specific Fire function and propagate the signal through the network.

One of the more powerful features of the MC68332 is the availability of the timer processor unit (TPU). The TPU has 16 available channels, which execute independently of the CPU. The timers can be set to interrupt the CPU at regular intervals (1/100 sec.) and divert the CPU to execute a list of functions. This means that the CPU time can be shared between the 'conscious' processes normally being executed and the 'unconscious' processes that are constantly interrupting the main processes in order to execute. Because the TPU interrupts the CPU at regular intervals, the interrupting functions must execute quickly otherwise they themselves will be interrupted. Thus, these functions are reserved for motor control, which requires regular intervals for execution.

Sensor Input - Full autonomy was the primary goal of this project. Each robot must be able to work independently from the rest of the group. Therefore, each robot must be equipped with all the facilities necessary to assure its independence. The most important sensor for each robot is the CCD camera. The 160x120 pixel, 16-bit image provides each robot with rich information about its environment.

Each robot is also equipped with five distance sensors. Many robot designers use large numbers of distance sensors (sonar, infrared), mounted at different points around the robot, in order to detect obstacles. Sometimes the total number of distance sensors can be as high as twelve or twenty-four. For the CIIPS Glory robots, we utilized the opposite approach. Instead of using a large number of fixed sensors, our robots use only five position sensing devices (PSDs). One is attached under the camera and two are mounted on the front of the robot, facing in the forward direction. The last two PSDs are mounted on the front of the robot, facing outward. This set-up is useful, because the robot can even track objects on either side of it, like walls or other robots.

The robots also have a compass module and encoders to measure the rotation of the each wheel.

Sensor data are stored in objects derived from the Sensor class, which is itself derived from the LockNode class. This allows for the sensors be arranged in a list structure.

Motor Output - The EyeMind architecture defines an abstract output class, Actuator, from which the DCMotor and Servo objects are derived. Like the Sensor class, the Actuator class is also derived from the LockNode class. The locking feature of the LockNode is especially important for the output, in order to be able to eliminate conflicting signals.

Communications - The Glory robots communicate through a 38,400 baud wireless communication module. The communication is set-up in a virtual token ring network. The highly dynamic, multi-agent environment, that robot soccer provides, lends itself to fast communication between robots.

I/O Management - Any other software object that wishes to access a specific I/O object, this is most often an instantiated Behavior object, must first check to see that this I/O object is registered with the Id. If the I/O object is registered, the Id will provide the object with a pointer to the I/O object. The object that receives the I/O pointer can then lock the sensor or actuator, so that no other behaviors can access it. Note that locking of I/O objects is not required after they are accessed. For the case of the PSD object, it is perfectly acceptable for more than one behavior to access this object and request its current value.

However, for other I/O objects, specifically output objects such as a DC motor, it is important that only one behavior be able to have access at any one time. In either case, if the I/O object is not registered with the Id when the behavior requests a pointer to the I/O object, then the Id instantiates this object, initializes it, places it on the list of registered I/O objects and then returns a pointer to the I/O object.

4.2 The conscious ego

The Ego class can be thought of as a conflict resolution module. This is analogous to Freud's ego in the human psyche model. We, as intelligent beings, make many immediate decisions which are soon forgotten and require little thought, but do pass briefly through our conscious processes. An example of this would be whether to turn or push a handle on a door. In most cases, when encountered with the need to pass through a closed door, one would attempt the action (behavior) that is most apparent or has worked in the past; let us assume this is to push the handle. If the door does not open, the behavior did not produce satisfying results, a new behavior is chosen, for instance to push the handle. The problem of the closed door probably passed briefly through our mind before the desired result (to open the door) dictated the action chosen (to push the handle). In the case that the door did not open, the problem at hand occupies more of our conscious thought as we determine an alternative action.

A fundamental abstract class in the Glory architecture is the State class. This class holds one piece of information, the desire to be satisfied. In order to derive a specific State, two functions must be defined. These are the Is(State* state) function and the Satisfy function. The Is(State* state) function returns an integer value corresponding to whether or not the state given *is* the current object. Note that this involves the use of an arbitrary measure, not actually comparing the two objects to determine if they are quantitatively the same, or refer to the same object. For example, a basic derived state is the PositionState, which returns true if the PositionState object is within a certain radius of the PositionState which was passed to the function. The other function which must be defined, the Satisfy function, is an important one, because it is this that gives the EyeMind architecture its flexibility. When the Satisfy function is called by the Ego, a set of behaviors are evoked, in an attempt to satisfy the State object. For example, the PositionState will check to see if the Drive behavior is active. If it is, then the PositionState will modify the desired position of the Drive behavior so that when the Drive behavior is executed, it will cause the robot to be closer to satisfying its states.

There are pointers to three lists of states in the ego, past states, desired states and current states. It should be pointed out that this model does not have a single all encompassing state. Therefore the current status of the robot is determined by a list of sub-states, such as position, battery power, possession of the ball, etc. The basic algorithm for the ego is:

```
while (desired_states)
{
   for each state
   {
          if Criticise(past_states,
current_states, desired_states)
      {
         LearnBad(superego);
         RemoveState(state);
      }
      else if Satisfied(id)
      {
            LearnGood(superego);
         RemoveState(state);
      }
      else state->Satisfy( );
   }
}
superego->CreateStrategy( );
```

The actual algorithm is slightly more advanced because it checks if any of the current states correspond to the desired state. It also sorts the desired states by their desire to be satisfied so that desired_states with a higher desire value can be satisfied sooner. Notice that the fourth line of the algorithm calls a function called Criticise( ). For the agent to be fully autonomous and intelligent appearing, it is imperative that the agent has a mechanism that enables it to determine when a desired state is not being satisfied. This is the role of the Criticise function. In the simplest case, the Criticise function will check the timestamp of the desired state. This timestamp represents the time by which the desired state should be attained. If the current system time is smaller than the desired state's timestamp, then the Criticise function will return true and the desired state's Satisfy function will execute. If the current system time is larger than the desired timestamp, then the Criticise function will return false and a new strategy will be created.

The final line of the Ego algorithm calls the Create Strategy function of the SuperEgo object, which will now be discussed.

4.3. The super ego

An important asset for an intelligent agent to have is the ability to plan ahead. For the game of soccer to be exciting, the robots must be able to coordinateoffensive and defensive plans, or plays.

This feature can easily be incorporated into the EyeMind framework. The SuperEgo class houses no real information, but provides the interface with higher-level algorithms, such as expert systems and adaptive critics.

When the CreateStrategy( ) function is called, a list of states which correspond to the strategy is appended to the list of desired states. This method of control has an advantage over the optimal control method, in that the path only needs to be roughly defined in terms of a series of desired positions. These positions can be modified easily and quickly, thereby enabling the overall path of the robot to be adapted in response to a changing situation. The movement from one point to the next, however, is accomplished with a reactive control method, allowing the robot to quickly respond to external events, like moving objects.

Because the SuperEgo, Ego, and Id run on different threads of execution, each can be effectively performed "at the same time". Of course, each thread shares processing time with each of the other processes. However, this does nevertheless allow the robot to plan while it is acting. This is a reflection of our own, human, actions. How often have you been doing a task, like walking to the bank, and thinking about something completely different? During that time you were probably planning some future event or remembering past events and caring little about the current task of walking and avoiding obstacles along your path.

The ability to coordinate movements between robots provides the team with an obvious advantage, as compared with an "every man for himself" style of play. However, the strategy being followed at any one time may have to be changed quite quickly, and such a change needs to be quickly communicated between all of the robots. However, the overall strategy should not be allowed to dictate an individual robot's every movement. Individual robots should retain a reasonable degree of autonomy. What the strategy should determine is the overall movement of the team, as well as the individual roles or positions within the team, such as offensive or defensive, forward, midfield or fullback.

The only unresolved issue to the system shown here, for abstract agents, is the problem of assigning objectives, or desired states. The game of soccer has an obvious main objective, to win the game by scoring more points than your opponent. Knowing this in advance is advantageous to the programmer of the system. However, if we consider the agent outside of the context of the game of soccer, what objectives should it have? An even more important question is how does the agent acquire its objectives? One could easily imagine asking the agent, "What do you want to do today?" The 'answer' to this question can easily be pre-programmed or dynamically changed depending on the master's instructions. However, the self-acquisition of goals is not such a straightforward task and, in the authors' opinion, an agent that is able to do this could be considered to be cognitive. For the CIIPS Glory, when the strategy function is called, the robot chooses a particular play from a predefined playbook. The playbook contains a set of four lists, one list for each of the robot players.

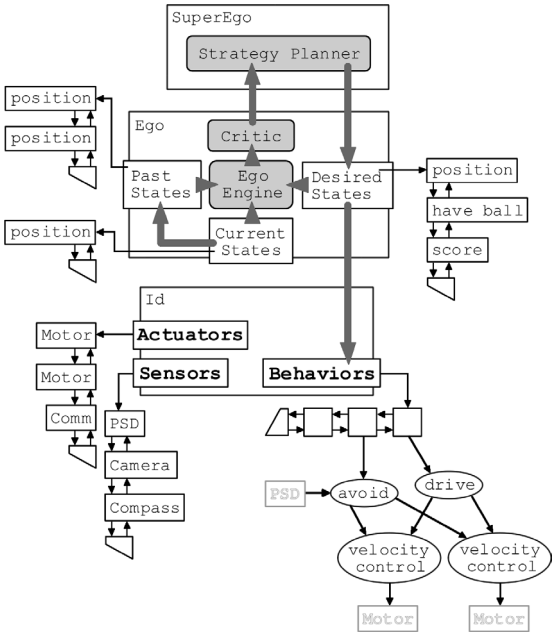


Fig. 4. Diagram of eyemind architecture.

Each list contains positions for a robot to navigate and is tailored to specific offensive or defensive roles. This approach works for the application of robot soccer but is of little use for generic agents. One suggested method of resolving the "what to do now" dilemma is generating random lists of behaviors to execute then storing lists which produce "good" results. However, because most robots and intelligent agents are not designed with survival and reproductive instincts, the evaluation of what constitutes general good behavior is unclear.

5. CONCLUSION

In this paper, we presented a behavior-based approach to controlling mobile robot agents. The method presented in this study is called "EyeMind", a "programmed" behavior-based system based on Freud's model of the human mind. This framework is an attempt to incorporate the useful qualities of each of the three intelligent architectures. It captures the fast, reactive control of behavior-based implementations while allowing high-level logic to dictate the overall result of the robots actions. The behaviors can be arranged in net structures and be seamlessly integrated with artificial neural networks thus allowing

for the system to learn. In future studies, we intend to implement the "Behavior Toolkit", which allows the "generation" of behavior-based systems similar to the existing "Rock&Roll" system.

REFERENCES

[1] T. Bräunl, B. Graf, *Small Robot Agents with On-Board Vision and Local Intelligence*, Advanced Robotics, vol. 14, no. 1, pp. 51-64 (14). 2000.

[2] T. Bräunl, P. Reinholdtsen, S. Humble, *CIIPS Glory - Small Soccer Robots with Local Image Processing*, P. Stone, T. Bulch, G. Kraetzschmar (Eds.), Robocup 2000: Robot Soccer World Cup IV, Springer-Verlag Berlin Heidelberg, LNAI 2019, pp. 523-526(4), 2001.

[3] T. Bräunl, *Embedded Robotics – Mobile Robot Design and Applications with Embedded Systems*, Springer-Verlag, Heidelberg, 2003.

[4] T. Bräunl, P. Wilke, *Flexible Wireless Communication Network for Mobile Robot Agents*, Industrial Robot International Journal, vol. 28, no. 3, pp. 220-232 (13). 2001.

[5] P. Levi, M. Muscholl, T. Bräunl, *Cooperative Mobile Robots Stuttgart: Architecture and Tasks*, Proc. of the 4th International Conference on Intelligent Autonomous Systems, IAS–4, Karlsruhe, März, pp. 310– 317 (8). 1995.

[6] D. C. Dennett. *True Believers: The Intentional Strategy and Why It Works*. Mind Design II. J. Haugeland. Cambridge, The MIT Press**:** 57-79. 1981.

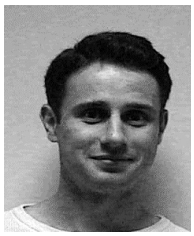
[7] R. A. Brooks, *Intelligence without Representation*. Mind Design II. J. Haugeland. Cambridge, The MIT Press**:** 395-420. 1991.

[8] S. Freud, *The Ego and the Id*. London, Hogarth Press. 1947.

[9] E. Gat, *Three-Layer Architectures*. Artificial Intelligence and Mobile Robots. D. Kortenkamp, R. P. Bonasso and R. Murphy. Cambridge, The MIT Press**:** 195-210. 1998.

[10] R. Arkin, *Behavior-Based Robotics*, MIT-Press, Cambridge MA, 1998.

Thomas Braunl is Associate Professor at the University of Western Australia, Perth, where he founded and directs the Mobile Robot Lab. He received a Diploma in informatics in 1986 from Univ. Kaiserslautern, an MS in computer science in 1987 from the University of Southern California, Los Angeles, and a Ph. D. and Habilitation in Informatics in 1989 and 1994, respectively, from Univ. Stuttgart. Professor Bräunl's research interests are robotics, vision, graphics, and concurrency. He is author of several research books and textbooks and developed the EyeBot mobile robot family.



Joshua Petitt received the B.S. in mechanical engineering from the University of Missouri-Rolla. His interests are robotics, kinematics, control theory and artificial intelligence.