**Published at:**
Transactions on Control, Automation, and Systems Engineering, ICASE: The Institute of Control, Automation and Systems Engineers, KOREA, Vol. 2, No. 1, June, 2003, June 2003, pp. 1-8 (8)

# 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 sensor 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 intelligent group behavior. Not being limited to the game of soccer and being completely autonomous, we are looking at a number of interesting scenarios. Robots can communicate with each other, e.g. for exchanging positions, findings of objects or just the local states they are currently in (e.g. sharing their current goals with other robots in the group). We are 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

## I. 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 the same way as 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, we present a different, behavior-based approach for autonomous robots, which can be used to implement arbitrary robot application scenarios.

## II. Robot Hardware

Each of our robots comprises a microcontroller system (*EyeBot*) 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 interested in research on autonomous mobile systems, so we took this clearly disadvantaged robot soccer approach instead of using 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 power. Frame grabbing can be done at 30 frames per second (fps). However, depending on the detection algorithm used this will be reduced significantly. Self-localization is an important task for our robots, since we do not use a global positioning system like an overhead camera. We rely on dead reckoning from a specified starting position and orientation. However, a robot will soon lose its exact position and orientation due to wheel slippage or - much worse - collision with another robot. Therefore, we integrated a digital compass to 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. Its local infrared sensors can update the robot position whenever it gets close to one of the sidewalls or a corner.

The camera mechanics was changed from tilting to panning in order to improve ball tracking. The camera can be moved sideways at a higher speed than the whole robot can turn, so this will allow us tracking of balls moving faster across a robot's field of view than it would 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 "master" and new incoming agents or leaving agents are handled automatically by this virtual token ring approach [3].

## III. Distributed Behavior Model

A behavior-based architecture for mobile agents has been suggested by Bräunl at Univ. Stuttgart in 1995 in [4]. This system by the name of *"Rock&Roll"* was successfully implemented in 1997 at UWA by Lampert/Bräunl and allowed **robot-independent** programming by selecting behaviors from a library list (Figure 1). Module libraries have been written for different robots (here: EyeBot, Soccerbot and Pioneer I robot), so all robot-specific implementation details are hidden from the application "programmer". Actually, no programming in a stronger sense takes place at this user level, since only modules are selected from a list and linked on a canvas. Automatic code is then generated by the Rock&Roll system, depending on the selection of the actual robot model.

Modules belong to the groups: sensor (yellow), compute (green), or actuator (red) and were all executed in parallel. Drawing links between the modules sets the actual data flow between modules. This will implicitly determine not only the data exchange, but also the synchronization of modules, since modules can "sense" new data coming in and use this for their internal calculations.

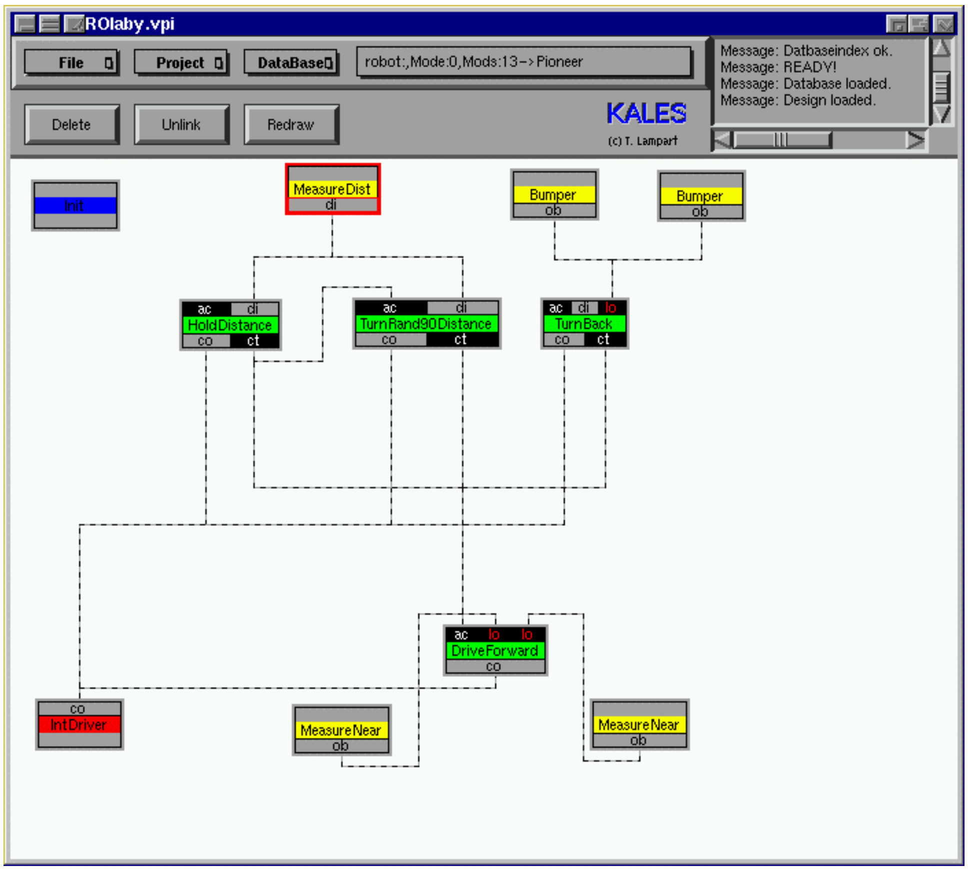
What the Rock&Roll system does not have is a clear distinction between behaviors and arbitration. This we intend to solve with our new general "behavior toolkit" as shown in Figure 2. This new toolkit provides the basic framework for implementing any behavior-based system, whether following a centralized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

## IV. 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 the application to our robot soccer team "CIIPS Glory" will now be described.

Manuscript received: 20 September 2002

*Thomas Bräunl*: Center for Intelligent Information Processing Systems, The University of Western Australia (braunl@ee.uwa.edu.au) http://robotics.ee.uwa.edu.au
*Joshua Petitt*: School of Mechanical Engineering, The University of Western Australia (petitj01@mech.uwa.edu.au)



Message: Datbaseindex ok.
Message: READY!
Message: Database loaded.
Message: Design loaded.

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

Many robots have been built around the "sense-think-act" paradigm, with varying degrees of success. This follows

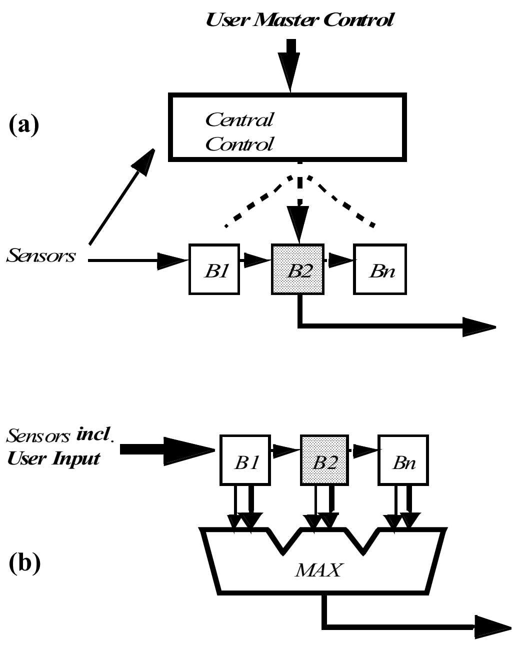


Fig. 2. Behavior Toolkit with (a) centralized or (b) local control

the path of classical artificial intelligence since a plan is created from a set of inputs. Other researchers have discarded this idea and have created robots with essentially a "sense-act" paradigm. These robots are termed "behavior-based" and have proven to be successful in dynamic environments. Other hybrid approaches have been implemented also with varying degrees of success. The commonalities between the two conventions are that in both models, the agent has *intentionality* [5] and the agent is *embodied* [6]. Intentionality refers to the desire of the agent to manipulate its surroundings and embodiment means that the agent only has limited knowledge about its environment and that the agent can act on, or change, its surroundings. In the previous sentence, the word "knowledge" was 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 represent a semantic net, for the classical group, a set of neuron weights, for the connectionists, and/or direct sensor readings (analog or digital in representation), for the behaviorists.

The drawback to the 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 carefully plan each task.
- Know nothing explicit about its tasks, just learn the correct way to accomplish it.
- Know nothing at all.

The idea of a zero-sum physical game, like soccer, is a perfect way to expose the deficiencies of each approach. The classical approach is often too slow to keep track of the progress of the game and a "good" plan now can easily turn into a "poor" plan later. From this point, the classical approach to 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 with complicated mappings of inputs to outputs, can be successful in negotiating dynamic environments (if there is no plan, then there is no plan to change), 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 this point, but I think many would agree that the mental abilities of ants, fish, or even birds are too coarse for the ability to have success in a game). The rest of the discussion will be intent on resolving the differences between the two camps and to show that both are needed for the success of an intelligent agent and to present a model which incorporates both approaches.

The real underlying difference in the reasoning versus reacting models 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 [7], 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 processes, rather that they communicate and affect each other.

Other researchers [8] have used three layer architectures in robot designs. They have assigned names to the layers surch as *skill layer*, *sequencing layer*, *planning layer* and the *controller*, *sequencer*, *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

One of the more powerful features of the MC68332 is the availability of the timer processer unit (TPU). The TPU has available 16 channels, which execute independant of the CPU. The timers can be set to interrupt the CPU on regular intervals (1/100 msec.) and divert the CPU to execute a list of functions. This means that the CPU time can be divided 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 functions must execute quickly otherwise the interrupting function will be interrupted. Thus, the functions are reserved for motor control which requires regular intervals for execution.

### Nodes, Lists, and Heaps

Before continuing further with the explination of the unconscious, a few basic concepts must be addressed. The fundamental data structure used in our architecture is a doubley-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 it can only be accessed by another object which possesses the correct key. This allows for multiple threads of control to access shared resources.

### Sensor Input

Full autonomy was the primary goal of this project. Each robot must be able to work independent from the group. Therefore, each robot must be equipped with all the facilities necessary for 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 around the robot to detect obstacles. Sometimes the number can reach as high as twelve or twenty-four. The Glory have taken the opposite approach, instead of 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 track objects to its side, like walls or other robots.

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

### Motor Output

The EyeMind architecture defines an abstract 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 to eliminate conflicting signals.

### Communications

The Glory 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 presents lends itself to fast communication between robots.

### I/O Management

The Id class provides the functionality for registering and managing all the sensors and actuators used by the agent. The Id contains lists of references for both actuator and sensor objects, from this point collectively referred to as I/O objects. Any object (later described as a behavior) that wishes to access an I/O object must first check to see that the object is registered with the Id. If the sensor or actuator is registered, the Id will return the I/O object pointer to the behavior. The behavior can then lock the sensor or actuator, so no other behaviors can access it. Note that the sensors or actuators are not required to be locked 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 sensors or actuators, for example a DC motor, it is important that only one behavior be able to have access. 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.

### Behaviors

A behavior is a mapping of sensor input to motor output through an arbitrary function. By combining simple behaviors, a robot can be engineered to assume a more complex, intelligent-appearing behavior. Robots that do simple tasks such as following light beams, to more complex problems like solving mazes and navigating dynamic enviornments, can be all be constructed, sometimes with no processing unit at all [9]. What they all have in common is a feedback loop between sensors and actuators and the ability for behaviors to 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, a excitation value is added to the current excitation value. If the 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 outputting directly to 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 Figure 3 for a diagram. The essential difference between the two objects is the delay between input and output through the function. For a typical ANN, the input signal is propagated through the layers in a synchronous fashion and the function 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 sensed.

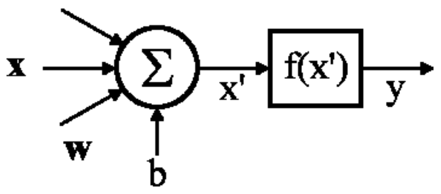


Fig. 3. Simple Summing Model of a Neuron

This model may be arranged as a feed-back controller or as a subsumption unit. Note 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 exists when a behavior is both excited and suppressed by an "infinite" signal, for this case the behavior is suppressed.

The Id class also registers and manages all behaviors that are currently executing. The Id class retains a list of up to sixteen 'root' behaviors. Each of these behaviors are excited by the timer processor unit (TPU) on 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.

### 4.2 The Conscious Ego

**States**

Another 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 will return an integer value corresponding to if the state given *is* the current object. Note that this is defining an arbitrary measure, not actually comparing if the two objects are 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 in. The other function which must be defined, the `Satisfy()` function, is what gives the EyeMind architecture flexibility.

The `Ego` can be thought of as a conflict resolution module. 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 Satified(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 are the desired state. It also sorts the desired states by their desire to be satisfied so that `desired_states` with a higher desire value get satisfied sooner. Notice 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 to determine when desired state is not being satisfied. This is the function of the `Criticise()` procedure. In the simplest case, the `Criticise()` function will check the timestamp of the desired state. This represents the time by which the desired state should be achieved. If the current system time is smaller than the desired state timestamp, then the `Criticise()` function will return true and the desired states `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.

When the `Satisfy()` function is called by the `Ego`, a set of behaviors are evoked to try and satisfy the state. For example, the `PositionState` will check to see if the `Drive` behavior is active, if it is then it 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.

The final line of the `Ego` algorithm calls the `CreateStrategy()` function of the `SuperEgo` object, which will now be discussed.

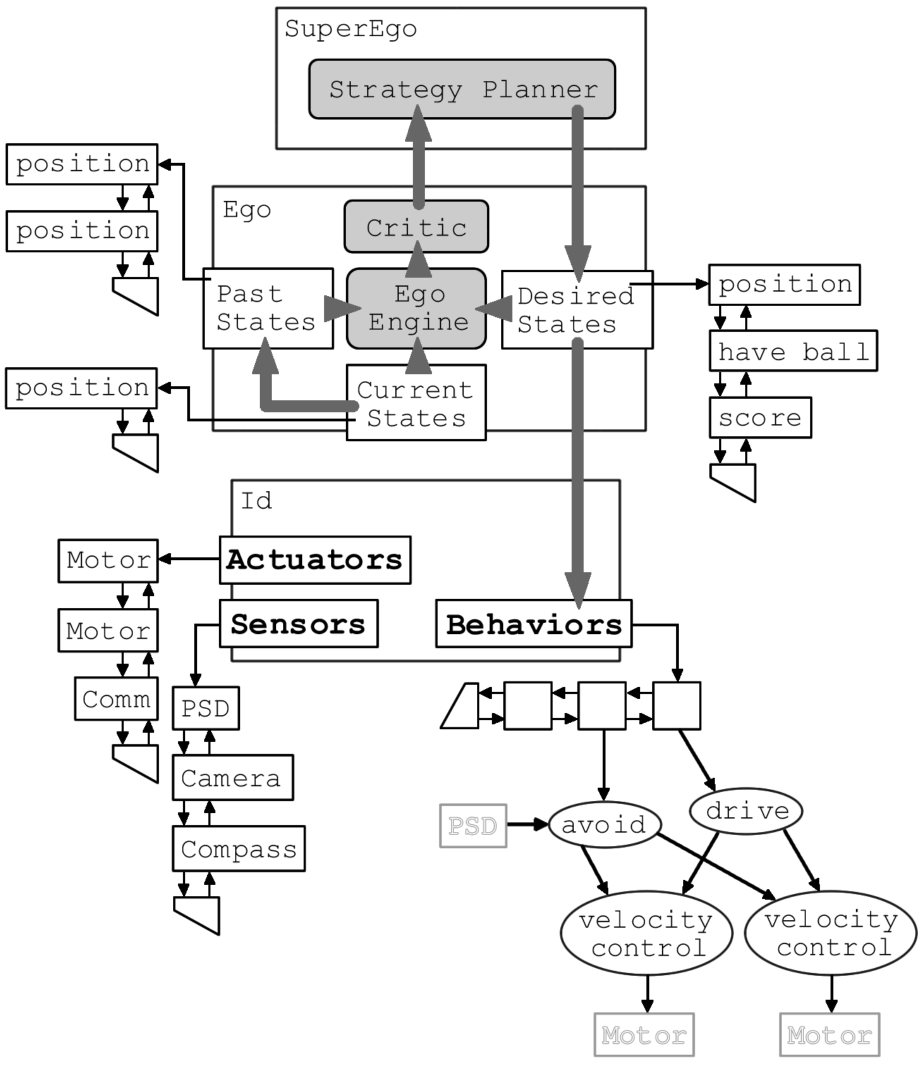


Fig. 4. Diagram of EyeMind Architecture

### 4.3 The Super Ego

An important asset for an intelligent agent to have is the ability to plan ahead. For an exciting game of soccer, the robots must be able to coordinate offensive and defensive plans, or plays. This is easily incorporated into the EyeMind framework. The `SuperEgo` class houses no real information, but provides the interface for higher-level algorithms, such as an expert system and adaptive critic.

When the `CreateStrategy()` function is called, the list of desired states is concatenated with a list of states which correspond to the strategy, or the specific soccer play. This method of control has an advantage over optimal control because the path is only roughly defined as a series of desired positions. These points can be easily and quickly adapted, thereby adapting the overall path of the robot. 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 Ego, Super-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. This does allow for the robot to be planning while acting. This is apparent in our actions as well, 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 about some future event or remembering past events and caring little about the current task of walking and avoiding obstacles on your mission.

**Strategy**

The coordination of movements between robots gives your team an obvious advantage over a “every man for himself” style of play. However, a strategy can quickly change therefore this change should be quickly recognized between robots. The strategy should not dictate an individual’s every movement. This task is left to the individual. What the strategy should determine is the overall movement of the team, as well as individual roles, such as offensive or defensive, forward, midfield, or fullback.

The strategy module is located in the super-ego of the robot. When the strategy function is called, the robot chooses a play out of a set playbook. The playbook contains a list of four lists.

**Behavior Experiments**

Forming a feedback loop between a PSD and a servomotor creates an interesting behavior. This relationship presented itself as a novel way to position a camera. Various approaches have been tried for camera control. Such approaches often deal with acquiring images from the camera and processing these images. However, these approaches are often computationally intensive not feasible for real-time control. Instead, a `Behavior` object which ties sensor data directly to camera movement without expensive overhead processing. The result is a fast, robust control strategy that can be used for positioning a camera.

The system that controls human eye movement is a very interesting subject. While not fully understood, there are a few behaviors that the eyes perform instinctively that we have come to understand through observation. The first is called saccade, which causes the eyes to constantly dart about, refreshing the mental image stored in our visual cortex. The second behavior causes our eyes to steadily follow moving objects. The third behavior causes our eyes to compensate for the movement of our head by moving in the opposite direction of our neck.

The set-up is relatively simple. A CCD camera is affixed to a common hobby servo. Attached directly below acquire images, control the servo and read data from the PSD. The control law for this is then relatively simple. Notice that the algorithm accomplishes two things. First, the camera moves towards objects that are close to it. Second, the decrement value causes all the values in the array to constantly go to zero, which simulates a forgetfulness mechanism in the array. This approach for controlling camera movement can successfully track slow moving objects, and causes the camera to favor toward objects that are close to the robot.

## VI. Conclusion

We have presented a behavior-based approach for controlling mobile robot agents. What has been presented here is "EyeMind", a *"programmed"* behavior-based system. However, as future work, we intend to implement the "Behavior Toolkit", which allows *"generation"* of behavior-based systems similar to our existing "Rock&Roll" system.

The only drawback to the system shown here, for abstract agents, is the problem of assigning goals. The game of soccer has an obvious main objective, to win the game. This is advantageous to the programmer of the system. However, if we abstract the agent away from the game of soccer, what goals should it have? An even more important question is how does the agent acquire goals? 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. But 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 cognitive.

## References

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

[2] Thomas Bräunl, Peter Reinholdtsen, Stephen 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, 2001, pp. 523-526(4)

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

[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, März 1995, pp. 310–317 (8)

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

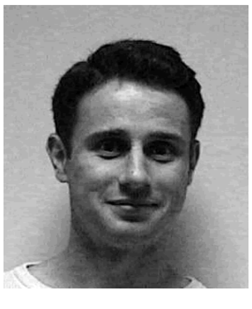
[6] Brooks, R. A. (1991). *Intelligence without Representation*. Mind Design II. J. Haugeland. Cambridge, The MIT Press**:** 395-420.

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

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

[9] 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 PhD 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 a B.S. in Mechanical Engineering from the University of Missouri-Rolla. His interests are robotics, kinematics, control theory and artificial intelligence.