Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture

Douglas Blank & Deepak Kumar
Computer Science
Bryn Mawr College
Bryn Mawr, PA 19010
{dblank,dkumar}@brynmawr.edu

Lisa Meeden
Computer Science
Swarthmore College
Swarthmore, PA 19081
meeden@cs.swarthmore.edu

James B. Marshall
Computer Science
Pomona College
Claremont, CA 91711
marshall@cs.pomona.edu

Abstract

In this paper we propose an intrinsic developmental algorithm that is designed to allow a mobile robot to incrementally progress through levels of increasingly sophisticated behavior. We believe that the core ingredients for such a developmental algorithm are abstractions, anticipations, and self-motivations. We describe a multilevel, cascaded discovery and control architecture that includes these core ingredients. As a first step toward implementing the proposed architecture, we explore two novel mechanisms: a governor for automatically regulating the training of a neural network, and a path-planning neural network driven by patterns of "mental states" that represent protogoals.

1. Introduction

Most intelligent robot control systems begin with the goal of creating a robot to carry out human-specified tasks. While these tasks vary in difficulty, they must, by their very nature, involve abstract concepts. For example, typical tasks might be: go to a specific location, identify an object, or pick up an object. Attempting to directly achieve the goal of carrying out human commands creates basic assumptions about the architectural design of a robot. We call this philosophy *task-oriented design*.

Within the task-oriented design paradigm, there are two competing methodologies: top-down, and bottom-up. Top-down designers apply computational algorithms that can be carried out on the robots so as to accomplish a given task. A wide range of top-down techniques are employed in robotics, such as dead reckoning (*e.g.*, using internal measures of space), sensor fusion, behavior fusion, and symbolic logic.

Likewise, bottom-up designers usually take the task to be performed by the robot as a pre-specified assumption. However, the control architecture of the robot is designed in a bottom-up fashion, with the overall behavior of the system emerging from the interactions of multiple components. Examples include subsumption architectures, supervised learning schemes, and evolutionary computation.

We believe that a significant pitfall exists in both the top-down and bottom-up task-oriented robot design methodologies: *inherent anthropomorphic bias*. This bias refers to the design of prespecified robot tasks: traditional research in the design of intelligent robots has attempted to get robots to do the tasks a human can, and to do them in a humanlike manner. Historically,

this methodology started out by having robots imitate the physical actions of a child playing with blocks. A task was conceived of as a symbolic *planning* problem, and then, using a robot equipped with an arm and a gripper, the robot was asked to manipulate the blocks in specific ways. The inherent anthropomorphic bias existed by design, since the issue was to explore models of intelligent human behavior. The pitfall in this approach, however, is that the symbolic modeling of behavior is based on assumptions about the capabilities of human bodies and human concepts, assumptions which may be inappropriate for the physical body and experiences of a robot.

Furthermore, even if we could build a robot with a humanlike body and senses, it is not clear that we can jump straight to the abstract task at hand. Many control issues need to be solved in order to have a robotic system carry out even the simplest of tasks. After a half-century of continuous research, the artificial intelligence and robotics communities are still far from developing a truly general-purpose intelligent system.

Recently, a new approach called *developmental robotics* has been applied to the design of robot behaviors (Lungarella, Metta, Pfeifer, & Sandini, 2003; Weng, McClelland, Pentland, Sporns, Stockman, Sur, & Thelen, 2001). In this approach, a robot or artifact under the control of an *intrinsic developmental algorithm* discovers capabilities through autonomous real-time interactions with its environment using its own sensors and effectors. That is, given a physical robot, behaviors (as well as mental capabilities) are "grown" using a developmental algorithm. The kinds of behaviors and mental capabilities that result from this process are not explicitly specified in advance. The focus is mainly on the intrinsic developmental algorithm and the computational mechanisms that allow the robot to grow.

A developmental approach to robotics is partly an attempt to eliminate anthropomorphic bias. By exploring the nature of development, the robot is essentially freed from the task of achieving a prespecified goal. As long as the intrinsic developmental algorithm demonstrates growing behavior there is no need to prespecify any particular task for the robot to perform. Indeed, it is the goal of developmental robotics to explore the range of tasks that can be learned (or grown) by a robot, given a specific developmental algorithm and a control architecture. This paper outlines our approach to a developmental robotics program and two experimental steps toward an implementation.

2. Overview of a Developmental Robotics Paradigm

The ultimate goal of our developmental robotics program is to design a control architecture that could be installed within a robot so that when that robot is turned on for the first time, it initiates an ongoing, autonomous developmental process. This process should be unsupervised, unscheduled, and task-independent, and the architecture should work equally well on any robot platform—a fixed robot arm, a wheeled robot, or a legged robot.

The intrinsic developmental process we are currently exploring contains three essential mechanisms: *abstraction*, *anticipation*, and *self-motivation*. In a realistic, dynamic environment, a robot is flooded with a constant stream of perceptual information. In order to use this information effectively for determining actions, a robot must have the ability to make abstractions so as to focus its attention on the most relevant features of the environment. Based on these abstractions, a robot must be able to anticipate how the environment will change over time, so as to go beyond simple reflexive behavior to purposeful behavior. Most importantly, the entire process should be driven

by internally-generated motivations which push the system toward further abstractions and more complex anticipations.

We believe that the developmental process should be employed in a hierarchical, bootstrapping manner, so as to result in the discovery of a range of increasingly sophisticated behaviors. For example, starting with a basic, built-in innate behavior, the robot might exercise its sensors and motors, use its mechanisms for abstraction and anticipation, and discover simple reflex behavior. A self-motivated control scheme would then take advantage of these discoveries to supersede the robot's innate behavior. This constitutes the first stage of the bootstrapping process.

The same intrinsic developmental algorithm can be applied recursively in subsequent stages, using the knowledge discovered in previous stages. For example, a secondary stage may abstract *sequences* of behaviors and corresponding perceptual views from the environment. These behavior sequences, termed *protoplans* (Meeden, 1994), might be used to guide the robot through a series of views or situations in the environment that would lead it to visit "interesting" places, as determined by its internal model of motivation. We will call these places *protogoals*. Here, the *proto* prefix is used to distinguish standard notions of plans and goals from the developmental ones described here. The same developmental process may be cascaded beyond this stage, enabling the discovery of ever more sophisticated plans and goals.

The control scheme that is responsible for driving the robot at each stage uses previously-discovered abstractions and anticipations while being pushed by internal motivations. At the lowest level, the motivational model indicates to the system how "comfortable" it is with its current situation. If it is too comfortable, it may become bored and take measures to move the robot into a more interesting region of the environment. Conversely, if the situation is chaotic, the system may become overexcited and attempt to return to a more stable and well-known region. These anthropomorphic notions will be described in more technical terms below.

Ultimately, the robot will grow enough to start exhibiting purposeful behaviors. For example, a robot might form the goal of getting to some particular place, and then be able to plan the behaviors necessary to go there. By its very nature, goal-directed behavior is decomposed using regression mechanisms, as is traditionally done in most research on AI planning. However, the "planning" performed in a developmental system more closely resembles a model of stimulus-response than it does a search process.

To summarize, we are proposing a multilevel, cascaded discovery and control architecture to explore developmental robotics. Each level of the architecture uses an instantiation of the intrinsic developmental algorithm and the control scheme. The key components of the developmental algorithm are the processes of abstraction and anticipation in the context of a model of motivation. In the rest of this paper, we elaborate on the details of this proposal.

3. The Intrinsic Developmental Algorithm

As in nature, the control architecture is not a completely blank slate, but contains a simple reflexive model for producing behavior, as well as the infrastructure necessary for adaptation. Over time, through self-motivated interactions with the environment, the robot acquires the knowledge necessary to exist in the environment in a purposeful way. The robot learns, not only about its environment, but also about its own perceptual and motor capabilities via this process. This approach to development proceeds hierarchically.

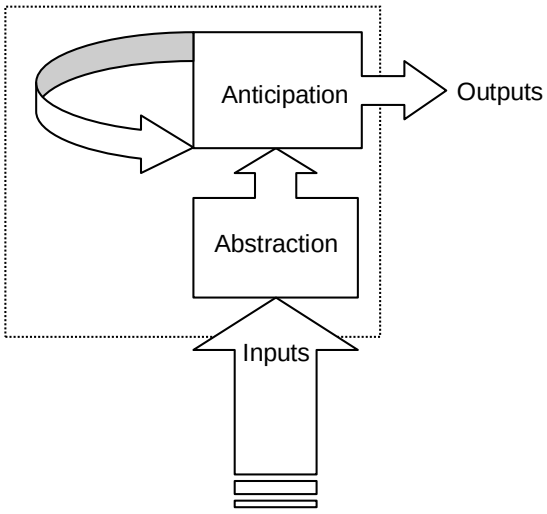

Figure 1: Key components of the intrinsic developmental algorithm.

Each level of the hierarchy combines two fundamental mechanisms—abstraction and anticipation—with motivation playing an overarching role (see Figure 1). We are currently exploring two abstraction mechanisms: self-organizing maps and resource allocating vector quantizers. For the anticipation mechanism we are focusing on simple recurrent networks.

3.1 Discovering Abstractions

Every robot is endowed with a suite of sensors and effectors. Data obtained from these sensors and effectors while the robot explores its environment represent the robot's experiences in the environment. As an essential first step in the growth process, the robot must derive abstractions from this data. Even for simple robots, however, this sensory-motor data can be very high-dimensional. Non-parametric clustering algorithms, such as self-organizing maps and resource allocating vector quantizers, work well for abstracting such high-dimensional data.

Self-organizing maps (SOMs) were pioneered by Kohonen (Kohonen, 2001). Briefly, a SOM maps typically high-dimensional input vectors to particular cells in a low-dimensional matrix. The matrix is topologically arranged in a unsupervised manner such that very similar input vectors map to the same cell, less similar inputs map to nearby cells, and very different inputs map to distant cells.

In a SOM, similarity is computed by comparing an input vector to a model vector associated with each cell. The model vector that is closest (as determined by the Euclidean distance between the model vector and the input) is designated as the winner. The model vectors of the winning cell and the cells in its surrounding neighborhood are updated to more closely match the given input vector.

The SOM idea is simple and effective. Any information that can be turned into a vector of numeric values can be self-organized by such a map. The idea has been applied in a wide variety of

problems ranging from creating maps for classifying the World Wide Web to analyzing financial data (Kohonen, 2001). Resulting SOMs are useful for two related reasons: their ability to automatically find abstractions, and their ability to help visualize complex data.

A resource allocating vector quantizer (RAVQ) (Linaker & Niklasson, 2000) is closely related to a SOM. The primary difference is that a SOM has a fixed number of model vectors arranged in a particular topology, while an RAVQ has no topology but can dynamically create new model vectors. Constructive systems of this kind can have problems distinguishing a new category of input from spurious noisy input. An RAVQ prevents occasional noisy inputs from generating new categories by keeping a buffer of the most recent inputs and calculating a moving average based on them. Only a stable sequence of new patterns will lead to the creation of a new model vector. Like a SOM, an RAVQ is a good tool for automatically discovering useful abstractions in high-dimensional data.

3.2 Anticipating the Future

Once a robot has discovered some abstractions, it can use them to try to anticipate what will happen next. This can help the robot to predict its own future, and enables it to begin to take over control from a lower level. Anticipation is a temporal activity and thus requires a time-sensitive computational mechanism.

The simple recurrent network (SRN) architecture was created by Elman in the late 1980's (Elman, 1990). There are two main classes of artificial neural networks: feedforward networks in which all activation flows in one direction, and recurrent networks in which activation is allowed to flow both forward and backward. In order to deal with time-dependent tasks, a feedforward network usually requires a fixed-length window to hold the past several inputs, while a recurrent network can take the current input alone and build up a contextual memory of the past inputs. The SRN combines the simplicity of a feedforward network with the power of a recurrent network. Like SOMs, SRNs are also simple and effective. They have been used to process many types of sequential data, including language and music.

3.3 Self-Motivation

In a sense, our goal is to find a model of self-motivation that acts like a co-evolutionary competitive arms race (Juillé & Pollack, 1996). The basic idea of such an arms race is that two populations of systems are pitted against each other, with each one continually "one-upping" the other in a spiraling increase of fitness. This works if the two populations begin with nearly equal fitness levels and remain relatively even throughout the race.

We imagine that a developmental motivational mechanism could play the same role, driving the system in a similar manner to ever-increasing performance. One possibility that we are exploring is an architecture that pushes the robot into mental and environmental states that are neither too easy nor too hard to correctly anticipate. In this manner, the robot would get "bored" in environments in which it can easily predict the outcome of its actions, but would retreat from environments that seem chaotic. We suspect that the region between predictability and randomness is a prime area for learning—a region one might compare to the so-called "edge of chaos" (Langton, 1991). The exact nature of such a mechanism, however, has yet to be fully developed.

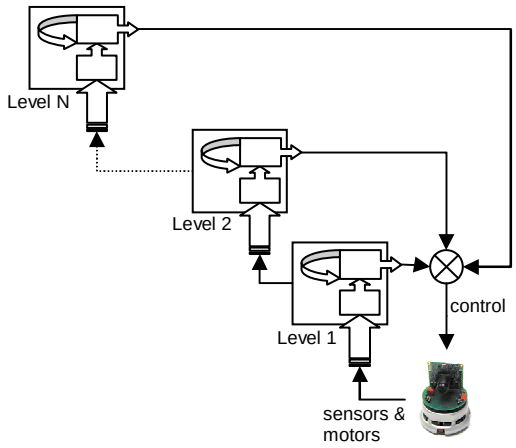

Figure 2: A general developmental robotics architecture.

3.4 The Control Scheme

Figure 2 depicts the hierarchical nature of the proposed control architecture. There is a Level 0 (not shown in the figure), which is built-in and consists of a set of simple control reflexes that enable the robot to wander and avoid obstacles. Each subsequent level combines an abstraction mechanism and an anticipation mechanism to allow the robot to adapt to its environment based on experience. The anticipation mechanism has a feedback loop to illustrate its time-dependent nature. Each level generates a robot control output signal, and abstractions from each level are used as inputs to the next higher level. The control outputs from all levels are integrated in a subsumption fashion, with higher levels having priority over lower levels.

Input to the first level of the hierarchy comes directly from the robot's sensors and motors. The abstraction mechanism at this level begins to extract basic perceptual and motor features observed during the robot's initial reflexive movements. The anticipation mechanism observes the abstractions being made and begins to recognize repeated multi-step sequences of features through time, chunking them into new, more compact representations. See Prem, Hortnagl, and Dorffner (2002) for a thorough overview of methods of storing sequences for use in autonomous robots.

The next level of the hierarchy takes these newly created chunked representations as input. Using the same abstraction mechanism, this level begins to make abstractions based on these chunked sequences. Using the same anticipation mechanism, this level begins to recognize sequences of sub-sequences from the previous level, chunking them further and sending them on to the next level. In this way, each level of the hierarchy processes the input at a longer time scale than the previous level.

This hierarchical development is driven by internal motivations rather than external goals. As mentioned, one possible motivation is to avoid boredom while not straying into chaos—in other words, to maintain a balance between exploitation and exploration. The anticipation mechanism provides a good measure of where the developing system falls along this continuum. When the anticipation mechanism of one level is able to accurately predict the behavior of the previous level,

it is time for the higher level to subsume control of the robot and for exploration to begin at the next level.

4. Implementing the Intrinsic Developmental Algorithm

Having described the fundamental elements for a developmental robotics program, we now explore the possibility of implementing these ideas. Implementing the architecture described above will have to be carried out in stages. The development of the architecture will require extensive empirical experiments in order to help establish the viability of the components of the proposed intrinsic developmental algorithm and how various alternatives might compare. For example, what is a good choice for the abstraction component: the hidden layer of a neural network, a SOM, an RAVQ, or something else? Along the way, there will no doubt be numerous challenges that will have to be addressed and several constraints that may have to be relaxed or modified.

We believe that focusing on task-independent problems is imperative. Likewise, we believe that the system should generate its own motivations and learn while under its own control. However, until we have all of the pieces of the architecture in place, enforcing all of these constraints will be difficult. Therefore, we have begun by exploring task-oriented, teacher-driven problems in our initial experiments. In the remainder of this paper, we describe two such experiments that demonstrate the viability of our developmental algorithm, and along the way address a few of the immediate problems that arise. In both cases, the abstractions generated by the developmental algorithm are very different from those that a human might design, but at the same time are appropriate for the robot's behavior in its environment.

- In the first experiment, we demonstrate how abstractions can be used to govern the learning of a neural-network robot controller.
- In the second experiment, we show how abstractions can be used to create purposeful behaviors for a robot.

All of our experiments have been implemented in Pyro (Blank, Meeden, & Kumar, 2003), a Python-based open source project with the aim of creating an abstract interface for robot programming. Pyro's interface makes it easy to run the same control program on different hardware platforms, such as a Khepera (Mondada, Franzi, & Ienne, 1993) or a Pioneer robot, as well as on different simulated platforms such as Player/Stage (Gerkey, Vaughan, & Howard, 2003) or Aria. Results reported in this paper are from experiments carried out on a simulated Pioneer robot operating in a Player/Stage world. Our long-term goal is to demonstrate the viability of our developmental architecture by obtaining consistent results across several different robot platforms.

4.1 Experiment 1: Using Abstractions to Govern Learning

It is well-known that neural networks are very sensitive to the order of presentation of their training patterns during learning. One serious problem that can arise has been termed *catastrophic forgetting* (French, 1999), and occurs when new training information completely destroys previously learned information. Given that our goal is to create an autonomous developmental learner based on neural networks, this sensitivity to training order is of great concern. We cannot hand-pick the training

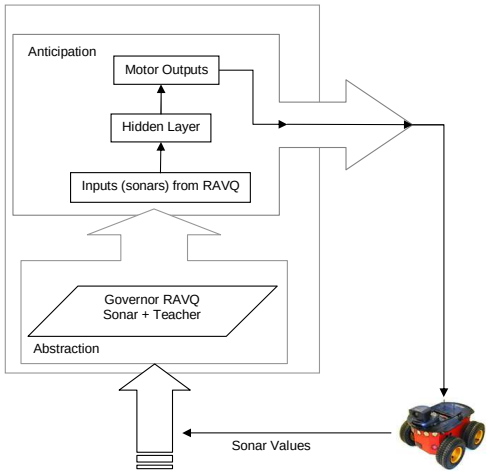

Figure 3: A network governor for training the wall-following task. Model vectors in the RAVQ are 18-dimensional (16 sonars plus 2 motor commands). The network has 16 input units, 8 hidden units, and 2 output units.

patterns for our developing robot. Instead, the robot chooses its own actions, initially according to its innate reflexes, and eventually based on its internal motivations. It is quite likely that the robot will experience long sequences of relatively similar sensory inputs, for instance while it is moving across a large open space. Due to this prolonged repetition of inputs, a neural network controller could easily be overtrained on the appropriate behavior for this state, and forget how to respond appropriately to other important states. We need an autonomous mechanism to oversee the training process so as to avoid catastrophic forgetting.

A *governor* is a mechanical device for automatically controlling the speed of a steam engine by regulating its intake of fuel. We have adopted this terminology to describe a new method of determining appropriate training patterns for neural networks. A *network governor* is an algorithmic device for automatically regulating the flow of training patterns into the network so as to avoid problems such as catastrophic forgetting.

One known solution to the catastrophic forgetting problem is to rehearse prior training patterns as new training patterns are presented (French, 1999). This allows the network to remember old information while it is incorporating new information. Our network governor uses this approach, but enhances it with the ability to decide when an input pattern should be considered new. The network governor is implemented as an RAVQ. As discussed earlier, one of the advantages of an RAVQ as an abstraction mechanism is that it can dynamically generate as many categories as needed for the given domain.

As shown in Figure 3, the governor sits between the network being trained and the robot. Its job is to categorize the current situation, discarding repeated instances of the same category and saving instances where the category changes. Model vectors in the RAVQ governor are made up of two components: the current sensor values and the desired motor command. On each time

Control method	Successful Trials	Mean Score
Teacher	10 of 10	711.10
Governor-Trained Network	8 of 10	685.91
Standard Network	2 of 10	1241.27

Table 1: Performance on the wall-following task. The score is based on the total sum of the robot's distance from the wall at each time step.

step, the RAVQ determines whether the current situation can be described by the previous model vector or if a different model vector is required. If the current situation is categorized as a change, then it is saved into a training buffer. When the training buffer is full, the oldest entry in the buffer is dropped to make room for the new entry. After the categorization step is complete, a backpropagation training step on the network is performed. The RAVQ governor maintains a pointer into the training buffer marking the position of the most recently trained pattern. This pointer is advanced, and the pattern stored in the next position of the buffer is trained. Over time, the governor's training buffer builds up a sequence of the unique situations that the robot has experienced in the recent past. In this way, the governor can rehearse the key moments of change while ignoring long stretches where no change occurs and thus avoid catastrophic forgetting.

Note that the network only sees raw sensory values and not the governor's categorizations. However, which sensory values it sees and the order in which it sees them is determined by the governor. The governor thus acts as a training supervisor. Once training is complete, the network can stand alone to perform the tasks on which it was trained.

To illustrate the importance of a governor to the developmental approach, we present the results from one experiment. The question to be addressed is: does a governor-trained network outperform a standard network in typical robot control tasks? For this experiment, the task is wall-following. The environment is a 5-meter by 5-meter world consisting of an L-shaped room with a narrow opening into a smaller square-shaped room. We created a teacher program to generate the desired translation and rotation motor commands to wall-follow on the left around the perimeter of this environment. The teacher tried to keep the robot no closer than 0.7 robot units and no farther than 1.0 robot units from the wall (a robot unit is equal to the diameter of the robot). We collected ten data sets using the teacher program to control a simulated Pioneer robot for 20,000 steps. Due to added noise in the simulation, each data set was different. Off-line, we then trained ten governed networks and ten standard networks on the saved data sets. The RAVQ governor parameter settings were: delta = 0.6, epsilon = 0.2, moving average buffer size = 2, training buffer size = 100. The neural network parameter settings were: learning rate = 0.2, momentum = 0.9.

After training was completed, we tested the networks in the same environment for 1,000 steps. The performance score for this task was the sum of the distances recorded by the left sonar sensor (in robot units). For this task, lower scores reflect better performance, as long as the robot does not hit the wall. If the resulting behavior caused the robot to get stuck against a wall, it was deemed unsuccessful and was not used to calculate mean performance. Table 1 summarizes the results. Not surprisingly, all ten tests of the teacher were successful. Eight of the governor-trained networks were successful and actually obtained a lower mean score (although the difference is not statistically significant). Only two of the standard networks were successful; however, they did not

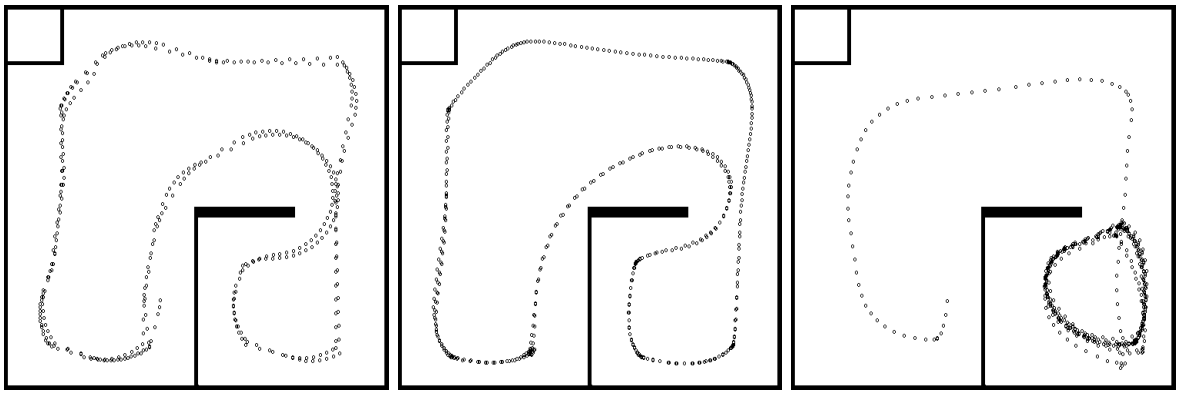

Figure 4: Example paths of the teacher (left), a governed neural network (center), and a standard neural network (right) in the wall-following task.

display good wall-following behavior, and got stuck cycling around one of the two rooms.

Representative paths of the teacher, a governor-trained network, and a standard ungoverned network are shown in Figure 4. Only the center point of the robot is depicted at each position along the path. It is interesting to note that the path of the governor-trained network is smoother than the path of the teacher. This is partially due to the fact that the governor-trained networks slow down in the corners. This can be seen from the density of the points at the corners in the center image of Figure 4. Also the governor-trained network is better at returning to the wall after exiting the smaller room. In contrast, although the standard network seems to have learned to follow walls, it remains relatively far away from them, which creates a problem when it tries to exit the smaller room. Clearly for this task the governor-trained networks outperformed the standard networks.

On average, the governor-trained networks created 87 distinct model vectors to categorize the situations in this small two-room world. Figure 5 shows the series of model vectors that were activated by one governor-trained network as the robot moved in a clockwise direction around the environment. The numbers in the figure indicate the model vectors associated with each position along the path of the robot. The path starts with the points categorized by model vector 17 and moves clockwise, ending with the points categorized by model vector 15. The number of model vectors that are created is highly dependent on the parameter settings of the RAVQ. However, in related experiments (not discussed in this paper), we found that the settings used here produced the most robust results for this task. It seems somewhat surprising that so many categories would be useful in such a simple world. This illustrates why our human intuitions about how to describe the world are probably not the right level of description for a robot and why we should avoid such anthropomorphic biases. Recently, there has been an increased research interest in how to equip robots with the ability to extract information from their own sensory stream. The techniques applied have included winner-take-all networks (Nolfi & Tani, 1999), clustering algorithms (Cohen, Oates, & Schmill, 2000), and RAVQs (Linaker & Jacobsson, 2001).

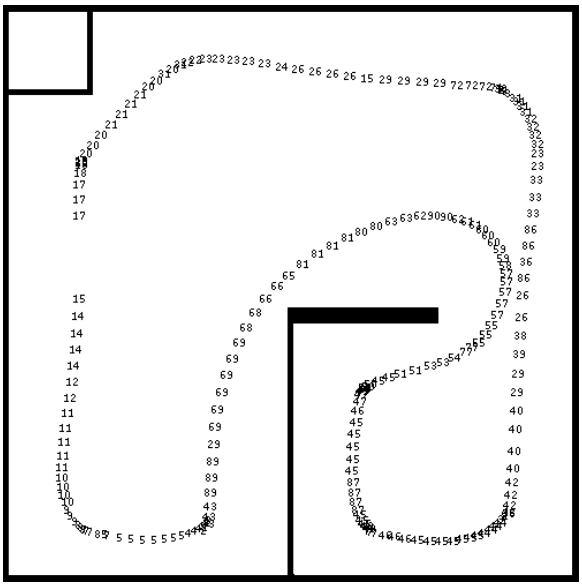

Figure 5: The path of the robot under control of the governor-trained network, showing the perceptual categories associated with each position as the robot moves clockwise around the environment.

Although such a governed neural controller is just a small step toward a fully developmental system, it does exhibit several of the key traits of the intrinsic developmental architecture described earlier. The targets of the network are generated by the system so as to *anticipate* what movement follows from a set of sensory inputs. In order to generalize, the hidden layer of the network must make appropriate *abstractions*. It may appear that the system completely lacks any type of *self-motivation*. However, it is interesting to consider the ability of the governor to "pay attention" to the inputs only when they are perceived to have changed. A possible next step would be to connect a "desire" to maintain a certain level of perceived change to a movement generator. For example, if the abstract states were changing too quickly (as determined by the governor), the controller could signal the robot to slow down. The governor could then play a critical role in the implementation of a model of motivation.

4.2 Experiment 2: Using Abstractions to Create Purposeful Behaviors

In the experiment described above, the governor mechanism acts as an intermediary between the environment and the robot controller network during training by deciding which sensory patterns to use as training patterns for the network. These selected patterns are held in the training buffer, while redundant patterns (as determined by the governor) are ignored. The controller network is thus temporally decoupled from the raw sensory input during training, since the training pattern used to update the network's weights on a given cycle may not correspond to the actual sonar pattern currently perceived by the robot. Insulating the controller from the robot's direct

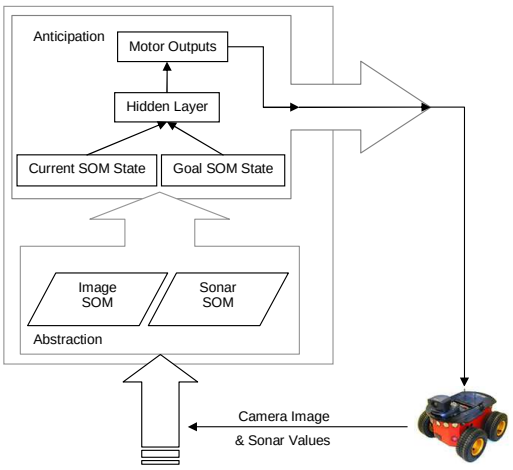

Figure 6: Using SOM abstractions to specify and achieve goals. The feedforward network consists of three layers, fully connected: 140 input units, 45 hidden units, and 2 output units.

environmental experience is a necessary first step toward achieving abstraction.

As was seen in Figure 4, using the governor can improve the overall behavior of the network. However, it is still a relatively weak abstraction mechanism, for at least two reasons. First, the governor's abstractions are used only during training. Once training is complete, the network operates without the governor. Second, even during the training phase, the category information generated by the RAVQ is never used by the controller network itself. The network sees only raw, unfiltered sensory information from the environment.

An alternative approach would be for the network to operate on higher-level representations *derived* from the sensory data, rather than on the raw sensory data itself. This seems necessary if the robot is to develop increasingly sophisticated types of behaviors that rely on a more conceptual view of the environment. These higher-level representational patterns might be regarded as "mental states" of the robot, as opposed to direct sensory states. They would be created by an appropriate abstraction mechanism interposed between the environment and the controller network, which would insulate the network from the robot's direct experience.

To test out such an approach, we implemented an abstraction mechanism as a pair of self-organizing maps, one for camera images and one for sonar readings (see Figure 6). As the robot moves through its environment under the control of some innate behavior, its camera images and sonar readings are recorded and fed into the SOMs, which transform the high-dimensional sensory data into a single compact, low-dimensional representation of the robot's perceptual state (*i.e.*, its "mental state"). These more abstract representations are then used by the controller network to determine the robot's next action. For this experiment, the network's job on each time step was to take two such patterns as input—one representing the robot's current situation and one representing a desired goal situation—and to generate an appropriate translation and rotation signal for the motors. We were interested in seeing, first of all, whether the system could discover

such representations on its own; second, to what extent these abstract representations capture relevant aspects of the environment; and third, whether the controller network could learn to use these representations to guide the robot toward a particular goal.

Our experiment was carried out using a simulated Pioneer robot with a blob-vision camera interface. A camera image consisted of a 20×15 array of pixels, where each pixel was a set of three color values (red, green, blue). Thus the raw image data consisted of a 900-dimensional vector. The sonar data consisted of a 16-dimensional vector of sonar values. Both SOMs used a 20×15 hexagonal topology. The robot's simulated 5-meter by 5-meter world contained a central obstacle surrounded by walls with regions of different colors. The purpose of the colored walls was to make it easier for the robot to visually distinguish different parts of the environment from one another.

The first step was to train the SOMs to produce useful abstractions of the robot's sensory inputs. To generate the training data, we let the robot wander around its environment for 5,000 time steps under the control of an innate obstacle-avoiding behavior, recording at each step the robot's raw camera and sonar sensor readings, and its motor responses (determined by the innate behavior). We then trained the SOMs, off-line, on the camera and sonar data, using 200 complete passes through each of the data sets. Both SOMs were trained with alpha = 0.2 and an initial neighborhood size of 5. Once trained, the SOMs could be used to transform the robot's raw sensory readings into a corresponding abstract representation of the robot's perceptual state. The next step was to determine whether the controller network could use these representations in place of actual sensor readings to accomplish some task. That is, in contrast to Experiment 1 described earlier, could the network learn to effectively control the robot solely in terms of these abstracted patterns, without having direct access to the environment through sensory states?

Ideally, the task learned by the network should arise from the robot's own internal motivations. However, in order to establish the basic soundness of our approach, we chose to train the network to simply follow a path through the world from one location to another, using training data generated by manually driving the robot from a starting position to two different goal positions. The network's task was to take the robot's current abstract "mental state" as input together with the desired final state (both generated by using the SOMs to transform the sensory readings associated with each position), and to output an appropriate motor response that brings the robot closer to achieving its goal. Figure 6 shows the network architecture. The network was trained with momentum = 0.9, and learning rate = 0.1.

Ten different training paths were manually created for each of the two goal positions, always beginning with the robot at the same position in the northeast corner of the world but varying its orientation. Goal A consisted of the robot facing the west wall of the world directly in front of a yellow region, while Goal B was at the southwest corner of the world with the robot facing southwest, looking at a red region on one wall and a blue region on the other. The controller network was trained without noise on these paths using ordinary backpropagation, with the motor values recorded for each path serving as training targets.

After training, we tested the robot to verify that it could behave appropriately when given different goals to achieve starting from the same initial position. The left image in Figure 7 shows the different paths taken by the robot when given the abstract representations of goals A and B. This shows that the robot can navigate successfully through its environment to reach a desired goal

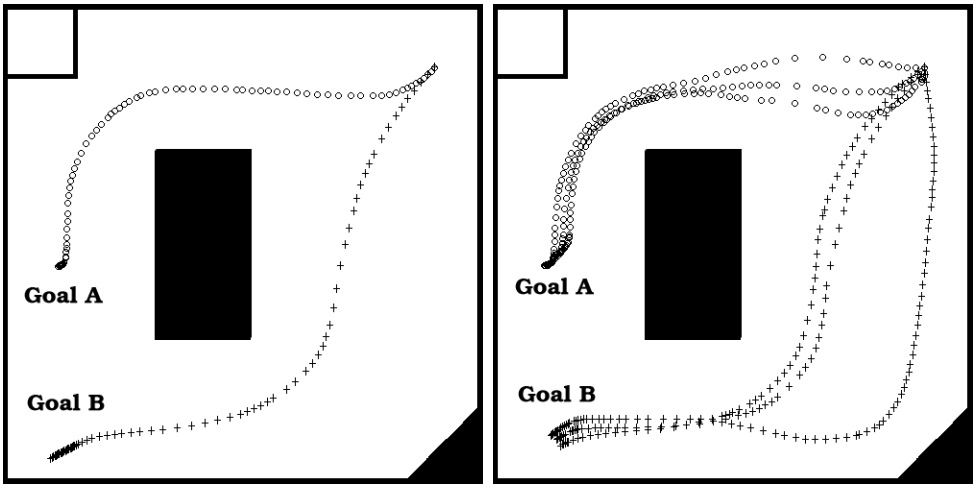

Figure 7: Protoplans and protogoals. The figure on the left shows the paths of the trained network from the start state given each of the goals. The figure on the right shows the paths when noise is added to the sensors. The network is able to generalize and direct the robot to the appropriate goal.

using only the abstractions created by the SOMs to guide it from where it currently is to where it needs to go.

To test generalization, we varied the starting orientation of the robot and added 10% noise to the sensors, using the same goals as before. The results are shown in the right image of Figure 7. The robot was able to find its way to the appropriate goal in each case. Notice that some of the paths overlap near the beginning of the runs, indicating that the robot was indeed paying attention to the information contained in the abstract goal state in deciding which way to go.

In order to get a better sense of the layout of the robot's "mental space", we plotted the trajectory of the camera SOM states encountered as the robot moved through the world from the starting position to a goal, under control of the network. Figure 8 shows the sequence of camera SOM model vectors that were activated for each of the paths shown in the left image of Figure 7. Each path seems to activate a different region of the SOM. In the case of the path to goal B, the SOM trajectory is largely confined to two small regions of the map, transitioning from one region to the other as the robot curves around the bottom right corner of the obstacle, approximately two-thirds of the way along the path. These regions are centered around model vectors (6, 2) and (4, 11) on the SOM diagram. The fact that the path clusters in these two areas in abstraction space is not too surprising, given that for each leg of the journey, the robot is looking at roughly the same region of the environment, and therefore sees similar camera images. Somewhat more surprising is the abrupt jump to the upper right region of the map as the robot reaches goal B. A similar jump occurs at the end of the trajectory for goal A. Implications of this observation will require further experiments and analyses of the SOM abstraction topologies.

Again, as in the first experiment, it is surprising that the self-organization of the system is

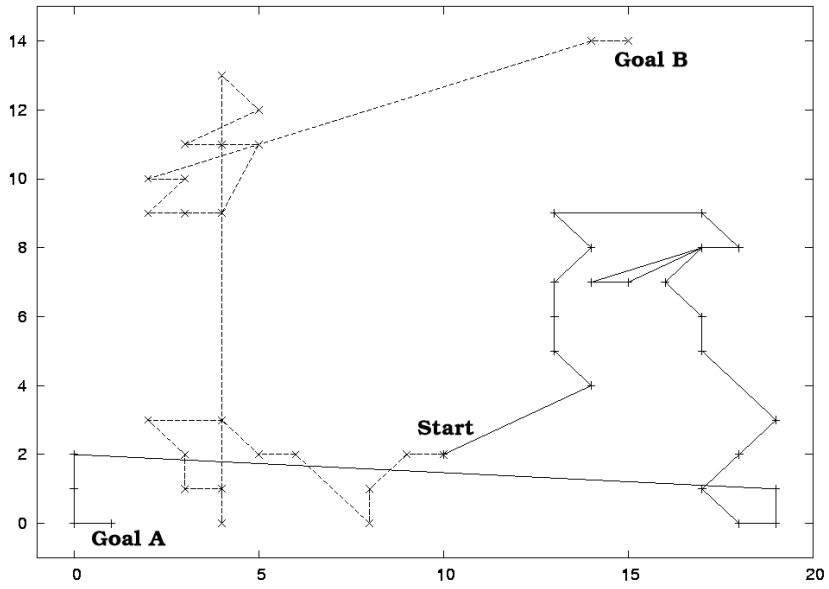

Figure 8: The trajectories through the SOM abstraction space for the two paths in the left image of Figure 7.

very different from that which a human might design, but appears to be exactly appropriate for the robot's sensors and view of the environment. Likewise, this experiment contains elements of our intrinsic developmental algorithm: abstractions are generated by the SOMs and used by the feedforward neural network; and the network is trained on anticipated movements. In addition, this experiment showed that a network could use self-organized abstractions as protogoals. However, where might these goals come from? We believe that goals and motivations can be added by building in a desire to reach self-selected protogoals. At first, such protogoals might be randomly-selected perceptual states. If the robot can wander from an initial state to a randomly-select goal state, it could perhaps begin to build protoplans.

5. Conclusions

In this paper, we have argued that there is a significant pitfall in traditional task-oriented robot design—an inherent anthropomorphic bias. Because robots are equipped with very different kinds of sensory and motor capabilities, they cannot easily share our human conceptualizations. Furthermore, we have argued in favor of eliminating the anthropomorphic bias by adopting a developmental approach to robot design.

We have described an intrinsic developmental algorithm designed to allow a mobile robot to incrementally progress through levels of increasingly sophisticated behavior. We believe that the core ingredients for such a developmental algorithm are abstractions, anticipations, and self-motivations. Our initial explorations in this paradigm involved two experiments. The first introduced the idea

of a neural network governor. The governor-trained network was shown to produce better behavior than that of a standard feedforward backpropagation network. In addition, the governor-trained network produced smoother wall-following paths than the teacher. The second experiment demonstrated the viability of using self-organized abstractions as representations of current states and goal states in the context of a goal-seeking network.

6. Acknowledgments

We would like to thank all of the students who contributed to this project. From Bryn Mawr College these include Ioana Butoi, Ananya Misra and Darby Thompson. From Swarthmore College these include Matt Fiedler, Evan Moses, Daniel Sproul, Jeremy Stober, and Yee Lin Tan. Several high school students have also contributed, including Cassandra Telenko and Andrew John.

References

Blank, D., Meeden, L., & Kumar, D. (2003). Python robotics: An environment for exploring robotics beyond LEGOs. In *Proceedings of the SIGCSE Conference*, Reno, Nevada.

Cohen, P., Oates, T., & Schmill, M. (2000). A method for clustering the experience of a mobile robot that accords with human judgments. In *Proceedings of the Seventeenth AAAI Conference.*

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

French, R. (1999). Catastrophic forgetting in connectionist networks: Causes, consequences and solutions. *Trends in Cognitive Sciences*, *3*(4), 128–135.

Gerkey, B., Vaughan, R., & Howard, A. (2003). The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pp. 317–323.

Juillé, H., & Pollack, J. B. (1996). Dynamics of co-evolutionary learning. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 526–534. MIT Press.

Kohonen, T. (2001). *Self-Organizing Maps* (Third edition). Springer.

Langton, C. (1991). Computation at the edge of chaos: Phase transitions and emergent computation. In Forest, S. (Ed.), *Emergent Computation*, pp. 12–37. MIT Press.

Linaker, F., & Jacobsson, H. (2001). Learning delayed response tasks through unsupervised event extraction. *International Journal of Computational Intelligence and Applications*, *1*(4), 413–426.

Linaker, F., & Niklasson, L. (2000). Time series segmentation using an adaptive resource allocating vector quantization network based on change detection. In *Proceedings of the International Joint Conference on Nerual Networks*. IEEE Computer Society.

Lungarella, M., Metta, G., Pfeifer, R., & Sandini, G. (2003). Developmental robotics: A survey. *Connection Science*, *15*(4), 151–190.

Meeden, L. (1994). *Towards planning: Incremental investigations into adaptive robot control*. Ph.D. thesis, Indiana University, Bloomington.

Mondada, R., Franzi, E., & Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In Yoshikawa, T. Y., & Miyazaki, F. (Eds.), *Proceedings of the Third International Symposium on Experimental Robots*, Berlin. Springer-Verlag.

Nolfi, S., & Tani, J. (1999). Extracting regularities in space and time through a cascade of prediction networks: The case of a mobile robot navigating in a structured environment. *Connection Science*, *11*(2), 1131–1141.

Prem, E., Hortnagl, E., & Dorffner, G. (2002). Growing event memories for autonomous robots. In Goerke, N., Dorffner, G., Prem, E., Morgavi, G., & Ross, P. (Eds.), *Proceedings of the Workshop On Growing Up Artifacts that Live: Basic Principles and Future Trends*, Edinburgh, Scotland. The Seventh International Conference on Simulation of Adaptive Behavior.

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., & Thelen, E. (2001). Autonomous mental development by robots and animals. *Science*, *291*, 599–600.