An Emergent Framework for Self-Motivation in Developmental Robotics

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

Douglas Blank
Computer Science Program
Bryn Mawr College
Bryn Mawr, PA 19010
dblank@cs.brynmawr.edu

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

Abstract

This paper explores a philosophy and connectionist algorithm for creating a long-term, self-motivated developmental robot control system. Self-motivation is viewed as an emergent property arising from two competing pressures: the need to accurately predict the environment while simultaneously wanting to seek out novelty in the environment. These competing internal pressures are designed to drive the system in a manner reminiscent of a co-evolutionary arms race.

Introduction

The quest for creating robot control systems that undergo an autonomous and extended developmental learning process was initiated by Weng and his colleagues (Weng *et al.* 2001). In their report, they differentiate the field of developmental robotics from traditional robotics by focusing on *task-independent* learning. Rather than building control systems to perform specific, predefined tasks, developmental robotics seeks to create open-ended learning systems that continually adapt to new problems. A number of robot control architectures have been created using this paradigm (Almassy, Edelman, & Sporns 1998; Weng & Zhang 2002; Lungarella *et al.* 2003), many of which involve some form of reinforcement learning. Reinforcement learning is an appealing approach because it provides a method for giving feedback to a developing system without having to specify how to succeed. Instead, the system is simply rewarded or punished, and must determine on its own how to behave so as to maximize its reward.

However, there is no consensus yet about the most appropriate source for the reinforcement signal in a developmental robotics system. The reinforcement could come from an external teacher, from an internal mechanism such as emotion, or from a combination of external and internal sources. For example, the SAIL robot, an early prototype of a developmental learning system, depended on external reinforcement. SAIL could learn to navigate the corridors of a building by being manually pushed by a human teacher, or by having the teacher press the robot's "good" button or "bad" button in response to its behavior (Weng *et al.* 2001). A more recent version of SAIL employs a reinforcement signal that is the weighted sum of both external reinforcement and an internal measure of novelty (Huang & Weng 2002). The system compares the predicted next state to the actual next state, and if the prediction is incorrect, novelty is considered to be high. The intent of introducing novelty is to model habituation, as when human babies get bored by constant stimulation and are attracted to novel stimuli. In the SAIL system, the external reinforcement is weighted much more strongly than the internal novelty detection. Therefore the external teacher can easily override the internal drive to perceive new things.

Another fruitful area of inspiration for creating general-purpose internal reinforcement signals is the use of artificial emotions (Gadanho & Hallam 1998). In Gadanho and Hallam's work, a simulated Khepera robot is endowed with a set of homeostatic variables related to energy, pain, and restlessness. The environment contains a set of obstacles and a set of food sources. The robot's energy decreases on every time step, and increases when it visits a food source. The robot's pain increases when it bumps into obstacles and its restlessness increases when it is not moving. These homeostatic variables can serve to positively reinforce behavior that increases energy and negatively reinforce behavior that increases pain or restlessness. Currently, these reinforcement signals are only used to determine when to switch between a set of pre-programmed behaviors. Thus the robot does not develop new behaviors, but simply determines the best way to sequence its innate behaviors.

We believe that a key step in exploring developmental architectures is to focus on internal sources of reinforcement. The learning process should be driven by *self-motivation*, that is, by the system's own internally-generated representations and goals, instead of relying on those provided by a teacher or designer outside the system according to some specific task to be learned. We are interested in creating a general learning architecture with self-motivation at its core, along with the other key processes of *abstraction* and *anticipation* (Blank *et al.* 2005). Abstraction and anticipation are active research areas (Kuipers & Beeson 2001; Butz, Sigaud, & Gerard 2002), but self-motivation has not yet received as much attention from the research community. We envision a control system in which abstraction, anticipation, and self-motivation are closely intertwined and develop together from the start within a single unified framework, using both internal and external sources of reinforcement. Such a system would build up abstractions of its experiences over time, guided by its internal motives, while learning to anticipate the effects of its sensorimotor interactions with the environment. Furthermore, a robot capable of learning about its own sensors and effectors as well as its surrounding environment would avoid the problem of anthropomorphic bias, since the robot's knowledge of its inherent capabilities and limitations, having been acquired through firsthand experience, would be directly grounded in sensorimotor perceptions (Blank *et al.* 2005).

There is another, perhaps even more important advantage of self-motivated systems. They can exhibit a degree of open-endedness not possible for systems that are designed to learn specific tasks. For example, the human capacity for learning is not only general-purpose and task-independent, but typically continues over a lifetime, becoming progressively more complex and sophisticated in the types of abstractions and behaviors that can be acquired. The learning tasks themselves may change over time, as different circumstances and goals arise, but the impetus to adapt is ever present.

How does this self-driven pressure to learn arise? In our view, it emerges from the interactions of other competing pressures within the system, in a manner reminiscent of a co-evolutionary arms race, in which two co-evolving species continually push each other toward ever greater complexity. For example, such a self-driven system might attempt to predict future states as accurately as possible, while also attempting to seek out unanticipated, novel states. In effect, these two pressures compete directly against one another, since a system able to perfectly predict future states would never encounter any novelty, and a system that regarded everything it saw as new and unexpected would be incapable of predicting anything. However, if these pressures are balanced appropriately, the system might be able to "bootstrap" its way to increasingly sophisticated behaviors and organization. In other words, by seeking out situations with enough novelty to be interesting without being overwhelmingly unpredictable, the system might achieve a kind of temporary "homeostasis" balanced between surprise and predictability. Gradually, the system would gain the upper hand as it learned to anticipate unexpected things better, and its level of "boredom" would increase, in turn pushing it to explore its environment in search of richer, more interesting experiences. On the other hand, too much surprise would cause it to seek out more predictable regions of the environment. The result would be a type of punctuated learning in which the system remains at a given level long enough to master the tasks at hand, before moving on to the next level. Clearly, such a capability would depend on having a robust, general-purpose learning system that could deal with the multitude of different learning tasks that would arise as the system's experiences and behaviors increased in complexity.

This paper takes an incremental approach to the problem of creating a self-motivated developmental system driven by predictability and novelty. As a first step, we propose a connectionist architecture and learning algorithm for implementing self-motivated robot control. A set of experiments is performed on a simulated robot to demonstrate the feasibility of this approach. Next, a detailed examination of the training process for one run on the robot is presented. Finally, the implications of this approach are discussed. It is important to note that the relatively simple environment used in the experiments described here is not rich enough to allow the full realization of higher levels of behavior that such a system should ultimately be capable of developing in principle. However, having shown the viability of this approach under basic conditions, we envision extending it to more realistic and complex environments in future work.

Architecture and Algorithm

In this section we propose a neural-network based learning architecture to address these issues, in which discrepancies between the predicted outcomes and the actual outcomes of the robot's actions in its environment serve as the fundamental source of self-motivation, thereby determining what the robot will learn to do. Although this represents an innate bias built into the architecture, it is not task-specific. The hope is that given the right developmental learning algorithm "hard wired" into the system (whether by evolution or engineering), the robot will be able to learn appropriate task-specific behaviors through its own experiences, guided by internally-generated feedback.

Under control of the neural network, the robot generates motor actions to perform, along with predictions of the effects of these actions on its current situation. In our model, situations and predictions consist of simple two-dimensional visual scenes, but other types of sensory representations could be used. After performing an action and observing the results, the robot's prediction is compared with the actual outcome, and a representation of the prediction error is created. This representation forms the basis of a reinforcement training signal for the network, using a version of Complementary Reinforcement Backpropagation (CRBP) (Ackley & Littman 1990).

In CRBP, continuous-valued output activations from a network are transformed into binary values stochastically, typically by flipping a biased coin using the output activations as biases. Depending on the particular binary output pattern generated, the network may receive reward or punishment as feedback. In the case of reward, the network's weights are changed using backpropagation with the binary pattern itself as the training target. In the case of punishment, however, the *complement* of the pattern is used. The stochastic nature of CRBP allows the network to learn using only positive or negative feedback signals instead of a fully-supervised training regimen, which is ideal from the point of view of a robot exploring its environment in real time.

In our version of CRBP, the amount of stochastic noise involved in transforming continuous output values into binary can be varied dynamically, under control of the robot itself. We introduce a *computational temperature* parameter τ, ranging from 0 to 100, that controls the amount of noise used in generating motor action vectors and their complements (Mitchell & Hofstadter 1989). At low temperature

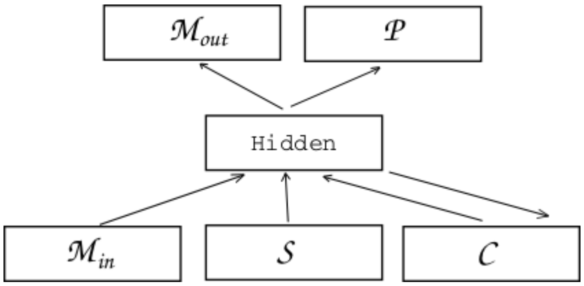

Figure 1: The network architecture

levels, activation values are translated to 0 or 1 nearly deterministically, while at high temperature the translation is nearly random, with 0 or 1 chosen essentially independently of the activation value. At intermediate temperatures, the translation function is a sigmoid curve of the general form $1/(1 + e^{-\alpha(x-0.5)})$, with the steepness parameter α of the sigmoid depending on τ. Thus temperature acts as a knob that determines the amount of influence the activation values exert on the translation process, ranging from no influence when $\tau = 100$ to complete determinism when $\tau = 0$.

Given the inherently temporal nature of prediction, we chose to use a Simple Recurrent Network (SRN) architecture (Elman 1990), shown in Figure 1. There are separate banks of units for representing the robot's motor actions ($\mathcal{M}_{in}$ and $\mathcal{M}_{out}$), sensory state ($\mathcal{S}$), sensory prediction ($\mathcal{P}$), and temporal context ($\mathcal{C}$), with each bank fully connected to the hidden layer. The purpose of the network is twofold: to generate motor actions for controlling the robot, and to generate predictions that in turn guide the training of the network itself. Prediction and control are interleaved during the training process, with different banks of input and output units active at different times. Since the choice of motor action depends on the robot's current sensory state and temporal context, banks $\mathcal{M}_{out}$, $\mathcal{S}$, and $\mathcal{C}$ are active when deciding what to do next, with $\mathcal{M}_{in}$ and $\mathcal{P}$ disabled. Predicting the next state depends on which motor action is performed given the current state and context, so banks $\mathcal{M}_{in}$, $\mathcal{S}$, $\mathcal{C}$, and $\mathcal{P}$ are active during prediction, with $\mathcal{M}_{out}$ disabled. Some weights of the network (namely, those from the state and context banks to the hidden layer) participate in learning both the control and prediction tasks, reflecting their closely intertwined relationship, while others are specific to one task or the other.

The training algorithm can be understood in terms of three general phases. In the first phase, internal feedback signals are generated from the robot's prediction error. A representation of the prediction error is created based on the discrepancy between the robot's actual observed state and its prediction made on the previous time step, and from this a reinforcement signal is computed, along with a temperature value.

Learning occurs during the second phase. First, the network weights responsible for *motor control* are updated using CRBP, based on the reinforcement signal from phase one. This corresponds to *behavioral learning*, which is driven by discrepancies in the robot's own internally-generated anticipations, rather than by feedback coming directly from the environment or an external teacher. Next, the network weights responsible for *prediction* are updated, using ordinary backpropagation with the robot's actual observed state as the feedback signal. This corresponds to *anticipatory learning*, which is driven by the robot's direct experience in the environment.

In the final control phase, the network generates the next action for the robot to take, as well as a prediction of the outcome of taking that action, and then executes the action.

A more detailed description of the algorithm is given below, outlining the steps performed at time t. At the beginning of Step 1, the following information is known: $\mathcal{M}_{t-1}$ is the motor action performed by the robot on the previous time step; $\mathcal{S}_{t-1}$ is the robot's previous sensory state; $\mathcal{C}_{t-1}$ is its previous temporal context; $\mathcal{P}_{t-1}$ is the prediction, generated at time $t-1$, of the robot's sensory state at time t; and $\mathcal{E}_{t-1}$ is a representation of the prediction error at time $t-1$, based on the discrepancy between $\mathcal{S}_{t-1}$ and $\mathcal{P}_{t-2}$.

- *Generation of internal feedback*

1. Observe the current sensory state $\mathcal{S}_t$.
2. Compare $\mathcal{S}_t$ to $\mathcal{P}_{t-1}$ and create a representation of the prediction error $\mathcal{E}_t$.
3. Compare $\mathcal{E}_t$ to $\mathcal{E}_{t-1}$ and compute a reinforcement signal r of $+1$, -1, or 0, and a temperature τ between 0 and 100.

- *Learning phase*

4. If r is positive, set the motor target $\mathcal{M}_{target}$ to $\mathcal{M}_{t-1}$. If r is negative, set $\mathcal{M}_{target}$ to the complement of $\mathcal{M}_{t-1}$. If r is zero, skip to Step 7.
5. With banks $\mathcal{M}_{in}$ and $\mathcal{P}$ disabled, perform one backpropagation pass with inputs $\mathcal{S}_{t-1}$ and $\mathcal{C}_{t-1}$ on the state and context banks, and $\mathcal{M}_{target}$ on the motor output bank. In the case of positive reinforcement, this makes the network more likely to produce $\mathcal{M}_{t-1}$ given the state and context $\mathcal{S}_{t-1}$ and $\mathcal{C}_{t-1}$. For negative reinforcement, however, the opposite action will be more likely.
6. With bank $\mathcal{M}_{out}$ disabled, perform one backpropagation pass with inputs $\mathcal{M}_{t-1}$, $\mathcal{S}_{t-1}$, and $\mathcal{C}_{t-1}$, and target $\mathcal{S}_t$ on the prediction bank. This makes the network more likely to correctly predict state $\mathcal{S}_t$ when performing motor action $\mathcal{M}_{t-1}$ in state $\mathcal{S}_{t-1}$ with context $\mathcal{C}_{t-1}$. Set $\mathcal{C}_t$ to the hidden layer activation pattern resulting from this step.

- *Control phase*

7. With banks $\mathcal{M}_{in}$ and $\mathcal{P}$ disabled, compute the activation of the output bank $\mathcal{M}_{out}$ using $\mathcal{S}_t$ and $\mathcal{C}_t$ as inputs to the network. Stochastically transform the continuous-valued activations of $\mathcal{M}_{out}$ into a binary motor representation $\mathcal{M}_t$, with the amount of noise determined by τ. This step generates the next motor action for the robot to perform, given its current state and context.

8. With bank $\mathcal{M}_{out}$ disabled, compute the prediction $\mathcal{P}_t$ using $\mathcal{M}_t$, $\mathcal{S}_t$, and $\mathcal{C}_t$ as inputs to the network. This step generates the robot's prediction of the next state given the motor action to perform and its current state and context.
9. Perform action $\mathcal{M}_t$.
10. Set t equal to $t+1$ and go to Step 1.

When training with CRBP, it is often helpful to use a higher learning rate for positive reinforcement than that used for negative reinforcement (Ackley & Littman 1990). A positive reinforcement signal provides evidence that the motor action just performed was a good response to the current situation, so a relatively large weight change helps to increase the likelihood that the robot will take the same action the next time it finds itself in a similar situation. Negative reinforcement, however, suggests only that the motor action was *not* a good thing to do, and offers no guarantee that the opposite action would actually have been better. In this case, using a lower learning rate helps to steer the network away from producing the same response in the future, while remaining somewhat noncommittal about what response the network should actually produce. Thus the learning rate to use in Step 5 above can be set dynamically in Step 4 according to the value of r. In addition, a separate learning rate for prediction may be used in Step 6 if desired.

State Representation

The above algorithm does not specify exactly how representations of the prediction error $\mathcal{E}_t$ are created in Step 2, or how reinforcement signals are computed from them in Step 3. In fact, the algorithm is fairly general, and does not depend on the particular representation chosen for robot states or motor actions. Furthermore, there is no requirement that robot states must contain purely *sensory* information from the external environment. States could contain additional proprioceptor information, as well as explicit representations of more abstract information generated internally by the robot, such as the prediction error itself.

In our current model, a state $\mathcal{S}_t$ is represented as a 40×10 grayscale image of intensity values normalized to the range 0–1, generated from a simulated blob vision camera. Prediction error $\mathcal{E}_t$ is represented as a 40×10 map of the error values obtained in Step 2 by subtracting the corresponding image values of $\mathcal{S}_t$ and $\mathcal{P}_{t-1}$, and normalizing to 0–1.

Internal Reinforcement Signal

To compute the reinforcement signal in Step 3, we first compute the "center of mass" coordinate, called the *error centroid*, for each two-dimensional error map $\mathcal{E}_{t-1}$ and $\mathcal{E}_t$. This coordinate is simply the weighted average of the two-dimensional coordinates of all 40×10 error values, weighted by the size of the error. In our experiments, we have used a binary weighting function in which the weight of the error is 1 if the observed value is significantly greater than the predicted value at that point in the map, or 0 otherwise. Other mapping functions are of course possible, such as weighting a value by the magnitude of the error. To compute the reinforcement, the error centroids of $\mathcal{E}_{t-1}$ and $\mathcal{E}_t$ are compared. If the centroid has moved *closer* to the center of the error map from time step $t-1$ to t, the reinforcement is positive; if the centroid has moved *away* from the center, the reinforcement is negative; otherwise it is zero.

The temperature is also updated on the basis of the prediction error. Recall that the temperature ranges from 0 (deterministic) to 100 (random). Currently there are only two cases when the temperature is not set to 0. The first is when there is no error centroid, which corresponds to perfect prediction. In this case, the temperature is set to 100 to induce exploration. The second is when the error centroid has remained stable between two successive steps, but is still not centered. In this case, the temperature is set to 50.

This method of computing the reinforcement signal represents a built-in bias of the system. This can be thought of as an innate tendency of the robot to want to attend to regions of unanticipated activity in the visual field by moving them to the center of view. It is important to note, however, that the reinforcement signal is not based directly on visual input from the environment; rather, it is based on the robot's own *expectations* of what it will see as a result of responding to its current situation. The training of the network is driven by this internally-generated error information rather than by externally-generated visual information.

Motor Representation

A binary representation for motor actions is necessary in order to allow CRBP to be used for the training of the network's motor responses. In Step 7 above, the continuous-valued activations of the $\mathcal{M}_{out}$ units are transformed into a binary vector $\mathcal{M}_t$. By injecting stochastic noise into this process, the network gains the ability to nondeterministically explore its weight space. This is especially important in the case of negative reinforcement, in which the optimal training target is unknown.

In the experiments described below, we used a simulated robot with only one degree of freedom of movement. The position of the robot was fixed at the center of its environment, with only its angular orientation allowed to change. We chose an 8-bit representation for the motor actions, where the number of ones in a pattern specified the robot's rotation speed and direction, allowing 9 distinct actions to be encoded. The order of the bits was irrelevant. For example, all-zeros represented turning left quickly, all-ones represented turning right quickly, and an equal number of ones and zeros caused the robot to stop. Many different patterns, therefore, were potentially available for the network to use in representing a particular motor action, which gave the robot more flexibility in learning to generate its motor responses. Accordingly, the $\mathcal{M}_{out}$ bank in Figure 1 contained eight units. However, when a motor action is presented to the network as input, it is first translated back into a continuous-valued scalar in the range 0–1, in order to make learning easier for the network. The $\mathcal{M}_{in}$ bank thus consisted of only a single unit.

Experiments

To test the architecture and the training algorithm, we created a simple environment in which the developing robot is

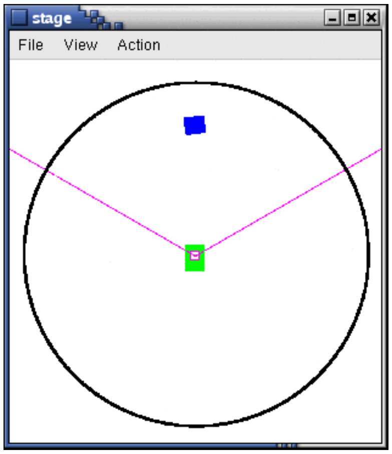

Figure 2: View of the training arena

fixed at the center of a circular arena and can rotate in order to observe its world. Also in the environment is a moving "target" robot controlled by an innate obstacle-avoidance behavior (see Figure 2). In some experiments, an additional stationary "decoy" robot was also present, in order to create a slightly more complicated environment.

The goal of these experiments is to induce the developing robot to attend to the target robot by tracking its motion. Clearly it should be possible to learn tracking by providing an external reinforcement signal that is based on whether the target robot is centered in the developing robot's visual field. However, the more interesting issue is whether the developing robot can learn to track given only an internal reinforcement signal based on the error of its own predictions. In this case the external reinforcement signal is directly related to the task of tracking, while the internal reinforcement signal is more indirect. In the following experiments we compare the performance of a developing robot when using external and internal reinforcement signals. The performance measure is based on the average offset of the moving target robot from the center of the developing robot's visual field.

The experiments were conducted using the Stage mobile robot simulator (Gerkey, Vaughan, & Howard 2003), where the developing robot was a simulated ActivMedia Pioneer 2 with a camera. The simulated camera had a 120-degree viewing angle centered on the front of the robot (indicated by the straight lines in Figure 2). Although the Stage simulator does not have simulated pixel-based camera output, we transformed Stage's "blob" data into a 40×10 grayscale image. When the target robot was in view, approximately 16 pixels (4% of the total image) were affected. The robot could turn to the left or right using one of 9 possible rotation speeds, as described earlier in section .

Using the robotics programming environment Pyro (Blank, Meeden, & Kumar 2003), we constructed the neural network shown in Figure 1, where the input layer had 1 motor-in unit, 400 state units, and 30 context units, the hidden layer had 30 units, and the output layer had 8 motor-out units and 400 prediction units. Using Pyro, the network was trained with the three-phase procedure described in section .

The target robot roamed around the inside circumference of the circular wall. At the beginning of each training trial, the target was positioned on the north side of the circle facing west. It then traveled to the left for several hundred time steps, following the circular wall as it went. When it reached the south side of the arena, it was repositioned at the starting point, but this time facing east. The target robot then traveled along the wall to the right, until again it reached a point approximately due south of the starting point. The purpose of this two-legged journey was to ensure that leftward and rightward motion was represented equally during training. The combined westward and eastward journey of the target robot constituted one training trial for the developing robot. Furthermore, whenever the target robot was repositioned at the north side of the arena, the activations of all of the network's context units $\mathcal{C}$ were reinitialized to 0.5. This occurred at the beginning and the middle of each training trial.

In our first experiment, which served as a basic benchmark, the external reinforcement signal was based on the *visual* centroid of the camera image. The robot received positive reinforcement if the visual centroid moved toward the center of the visual field, and negative feedback if it moved away. If the target robot was not in view, no learning was performed. We ran this experiment with computational temperature turned off (*i.e.*, set to 0) in order to see how well the robot could learn in the absence of noise. All of the runs attained a high level of performance within 10 training trials. The network architecture and training procedure enabled the robot to learn to track the target easily.

Of course, our real interest was in seeing if the robot could learn this task indirectly, by using its internally-generated prediction error in place of the actual visual input (as described earlier in section). As it turned out, using the internal reinforcement signal required that computational temperature be turned on in order for learning to be successful. Although the learning process was slower, the robot was still able to learn to track the moving target robot, even with a stationary decoy robot present in the environment. The next section examines in detail one successful run of this second experiment.

Analysis of a Training Run

This run is representative of those that learned to track the moving target robot using only the internally-generated reinforcement signal based on movement of the error centroid. As can be seen in Figure 3 (left), initial performance was about 0.50, but quickly rose to above 0.80 within the first 40 trials. On trial 44 the performance of the network reached its peak, around 0.87. For comparison, we hand-coded a robot to perform the visual tracking task as well as possible, and it scored 0.92. A perfect score of 1.0 is unattainable due to the system's inability to maintain the centroid in the exact center of view at all times.

Recall that our system is designed to perform two conflicting tasks: to accurately predict the next state $\mathcal{P}_{t+1}$, but also to track the areas of its visual field where it cannot predict. Not surprisingly, the better the system is able to predict, the less it is able to track, resulting in a lower perfor-

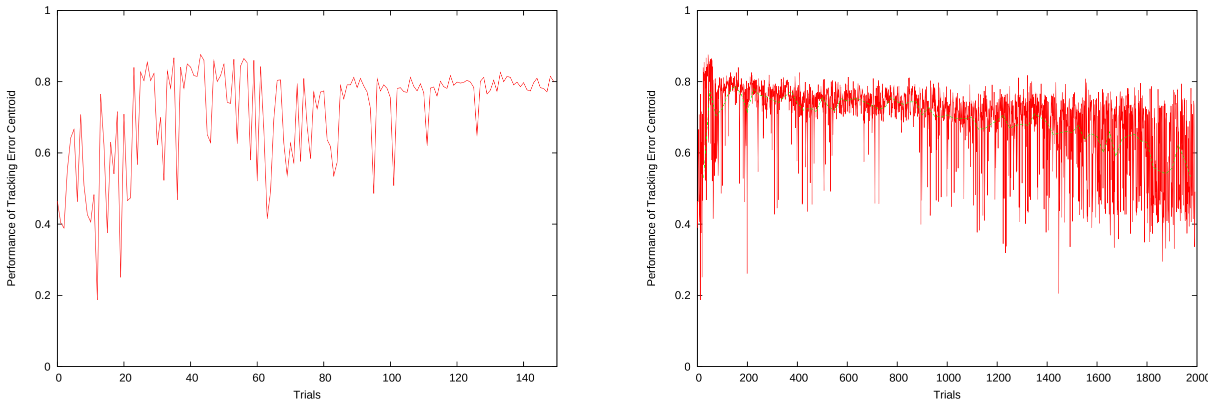

Figure 3: Performance of error centroid tracking: first 150 training trials (left); all trials (right)

mance measure. From these competing goals, three recognizable phases emerge: an early phase (around trials 0 to 35) where the performance on tracking the moving target robot increases; a middle phase where the peak performance is attained (around trials 35 to 60); and a late phase in which tracking performance slowly declines (trials 60 and greater).

Figure 4 shows representative camera images and prediction error data from the middle and late phases of this run. Each column labeled *Camera* shows a sequence of four camera images, with time running from top to bottom. The target robot can be seen as a square of gray pixels near the center of the visual field. The prediction error associated with each camera image is shown to its right. The black pixels indicate where the errors occurred on the prediction bank $\mathcal{P}$ at that step during training. Notice that some of the prediction error regions are smaller than the associated regions from the camera image. This indicates that the system has begun to make some accurate predictions. The system received negative feedback between the first and the second rows and again between the third and fourth rows (since the error centroids have moved slightly farther away from the center). Between the second and third rows, the network was rewarded, since the centroid moved toward the center of the field.

For the camera images and prediction error in the late phase of training, the most noticeable feature is that in the first and fourth rows, there is no error in prediction. This resulted in reward between the first and second rows, and also between the second and third rows (as the centroid gets closer to the center). However, the system was again punished between the third and fourth rows as it "lost" the error centroid.

Further examination of the tracking performance during the late phase shows that it continues to fall until the end of the run at trial 2000. Figure 3 (right) shows the steady decline in performance and an increasing range of performance variability. To understand this behavior better, let us look more closely at how the prediction error evolves over time.

Figure 5 shows that prediction accuracy climbs over the span of 2000 trials, albeit very slowly and also with increasing variability. Indeed, as performance continues to

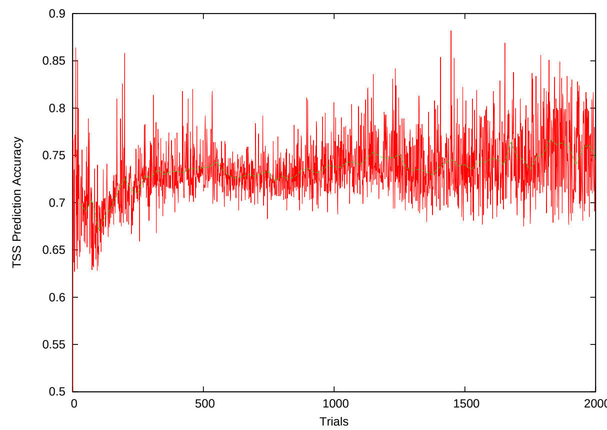

Figure 5: Prediction accuracy over all trials

increase in the late stage, the robot encounters fewer views containing any error at all, for which it is then punished. It is in this stage that the competing pressures discussed earlier are most apparent. If the experimental environment had been richer and more varied, after the developing robot had learned tracking, it would likely have been driven by its prediction error to focus on a new aspect of its world.

Discussion and Conclusion

The defining characteristic of a developmental robotics architecture is task-independence. A developmental system must be open-ended and capable of finding interesting phenomena to focus on and learn about. The previous experiment suggests that a very general internal mechanism, such as an error centroid created from the robot's own predictions, can serve as a successful reinforcement signal for a developmental connectionist architecture. This initial experiment provides a benchmark for what a self-motivated learner can achieve with limited sensory capabilities in a simple environment. However, the idea of using prediction error as a reinforcer is so general that this same mechanism should be capable of providing a useful reinforcement signal for other sensory modalities and more complex environments.

This paper has outlined a philosophy for designing sys-

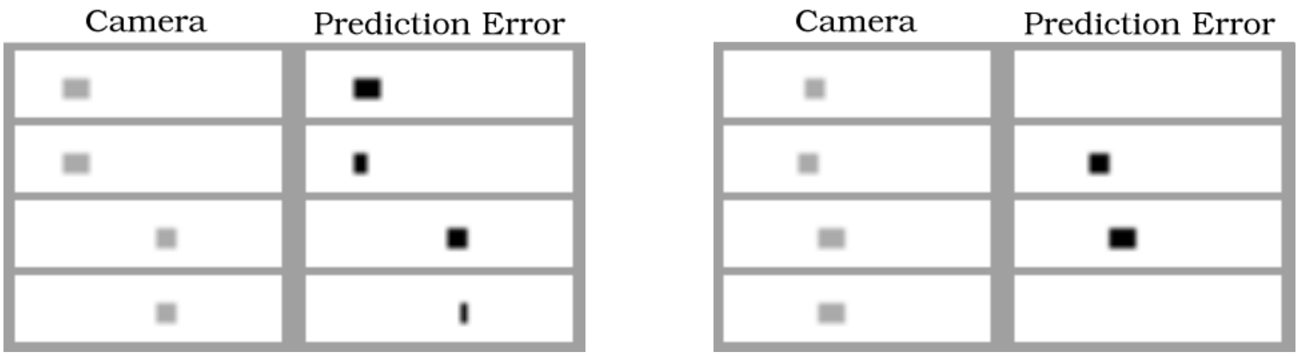

Figure 4: Sample camera images and prediction error data from the middle phase of learning (left) and the late phase of learning (right)

tems with self-motivation. We believe that self-motivation is an emergent property generated by the competing pressures that arise in attempting to balance predictability and novelty. In the current work, we have proposed a simple recurrent network architecture and algorithm in which systems for learning prediction and control are closely intertwined. The prediction and control pathways within the network share connection weights from the state and context units to the hidden units, and are trained in an interleaved fashion. One system attempts to make predictions of future sensory experiences while the other uses a reinforcement signal based on error provided by the first to drive control. Previous research has shown that prediction learning is facilitated within a system when control and prediction share pathways and when the control signals are internally generated, but not when the control signals come from an outside teacher (Parisi, Cecconi, & Nolfi 1990).

In our model, as the predictive system becomes better at anticipating the consequences of the control system's actions, novelty decreases, and the behavior of the predictive system becomes more tightly coupled to the behavior of the control system. As novelty decreases, the error map generated by the predictive system becomes smaller and more fragmented, which may cause the error centroid to jump around at random or disappear entirely. The control system thus has a harder time attending to novel parts of the sensory input. As the control system's performance declines, the robot appears to "lose interest" in those aspects of the sensory input that had previously captured its attention. The coupling between the predictive and control systems therefore begins to weaken, since the control system is no longer reliably paying attention to what it had before. As the predictive system loses its ability to reliably predict the responses of the control system, novelty once again begins to increase. At this point, the novelty of some other stimulus may begin to attract the system's attention (although in our experiment the developing robot never found another focus of attention). We believe that this scenario could potentially serve as a model of habituation. More generally, the interplay between predictability and novelty in our view provides a rich framework for exploring open-ended learning and skill acquisition in developmental robotics.

Acknowledgements

We would like to thank Deepak Kumar, Paul Grobstein, Chris Prince, and the members of the Emergent Phenomena Research Group at Bryn Mawr for engaging discussions on this topic. We would also like to thank the anonymous reviewers whose suggestions helped to improve the paper.

References

Ackley, D. H., and Littman, M. L. 1990. Generalization and scaling in reinforcement learning. In Touretzky, D. S., ed., *Advances in Neural Information Processing Systems 2*. San Mateo, CA: Morgan Kaufmann. 550–557.

Almassy, N.; Edelman, G. M.; and Sporns, O. 1998. Behavioral constraints in the development of neuronal properties: A cortical model embedded in a real-world device. *Cerebral Cortex* 8:346.

Blank, D.; Kumar, D.; Meeden, L.; and Marshall, J. 2005. Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems* 36(2).

Blank, D.; Meeden, L.; and Kumar, D. 2003. Python robotics: An environment for exploring robotics beyond LEGOs. In *ACM Special Interest Group on Computer Science Education Conference*.

Butz, M. V.; Sigaud, O.; and Gerard, P. 2002. Internal models and anticipations in adaptive learning systems. In *Proceedings of the Workshop on Adaptive Behavior in Anticipatory Learning Systems*, 1–23.

Elman, J. L. 1990. Finding structure in time. *Cognitive Science* 14(2):179–211.

Gadanho, S. C., and Hallam, J. 1998. Exploring the role of emotions in autonomous robot learning. In *Proceedings of the AAAI Fall Symposium on emotional intelligence*, 84–89. AAAI Press.

Gerkey, B.; Vaughan, R.; and Howard, A. 2003. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, 317–323.

Huang, X., and Weng, J. 2002. Novelty and reinforcement learning in the value system of developmental robots. In

Proceedings of the 2nd International Workshop on Epigenetic Robotics, volume 94, 47–55. Lund, Sweden: Lund University Cognitive Studies.

Kuipers, B., and Beeson, P. 2001. Toward bootstrap learning for place recognition. In Coradeschi, S., and Saffiotti, A., eds., *Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems, 2001 AAAI Fall Symposium*, number 01-01 in FS, 25–30. AAAI Press.

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

Mitchell, M., and Hofstadter, D. R. 1989. The role of computational temperature in a computer model of concepts and analogy-making. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates.

Parisi, D.; Cecconi, F.; and Nolfi, S. 1990. Econets: Neural networks that learn in an environment. *Network* 1:149–168.

Weng, J., and Zhang, Y. 2002. Developmental robotics – a new paradigm. In *Proceedings of the 2nd International Workshop on Epigenetic Robotics*, volume 94, 163–174. Lund, Sweden: Lund University Cognitive Studies.

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