

Surprise-Based Learning for Autonomous Systems

Nadeesha Ranasinghe and Wei-Min Shen

ABSTRACT

Dealing with unexpected situations is a key challenge faced by autonomous robots. This paper describes a promising solution for this challenge called "Surprise-Based Learning" (SBL), in which a learning robot engages in a life-long cyclic learning process consisting of "prediction, action, observation, analysis (of surprise) and adaptation". In particular, the robot always predicts the consequences of its actions, detects the surprises whenever there is a significant discrepancy between the prediction and the observed reality, analyzes the surprises for its causes and uses the critical knowledge extracted from analysis to adapt itself to the unexpected situations. SBL was successfully demonstrated on a physical modular robot which learned to navigate to desired goal-scenes with no prior knowledge about the environment, its sensors or the consequences of its actions. The scalability of SBL in the number of sensors and actions was evaluated to be reasonable for a physical robot, while adaptation to unexpected situations such as hardware failure and goal transfer was very successful.

1. Introduction

One of the key challenging issues for autonomous robots is dealing with unexpected situations. Such situations rise when there are unexpected changes in robots themselves (such as sensors, actions, and body configurations), or in the tasks and environment that the robot is facing. This challenge is not avoidable at the outset because no matter how careful a robot is engineered, the initial knowledge of the robot is bound to be incomplete or incorrect with respect to the richness of the real world. To address this challenge, autonomous robots must learn and adapt. They must detect surprises, analyze surprises, and adapt their knowledge whenever they can. This requirement demands an integrated solution for perception, planning, memory, reasoning, learning, focusing of attention, and discovery. It must cope with the continuous (non-discrete), uncertain, and vast information/action space in the real world.

This paper describes a promising solution for the above challenge based on a set of techniques centered on the idea of Surprise-Based Learning (SBL). In a life-long process, a learning robot engages in a cycle of "prediction, action, observation, analysis (of surprise), and adaptation." In particular, the robot always predicts the consequences of its actions, detects surprises whenever there is a significant discrepancy between a prediction and the observed reality, analyzes the surprise for its causes, uses the critical knowledge extracted from the analysis to adapt itself to the unexpected situations. All these capabilities are integrated in a unified architecture.

Three advantages of this approach are observed experimentally. It is scalable to the number of percepts and actions, fault-tolerant for failures and malfunctions in sensors and actions, and adaptable to unexpected changes and uncertainties in tasks and environments. These advantages are enabled by three key underlying technologies: an algorithm for complementary discrimination learning, a focus-of-attention technique for automatically selecting and associating relevant sensors with actions, and a memory/forgetting mechanism for model and goal management. This paper will describe these techniques in detail and demonstrate the mentioned advantages through a sequence of experiments, derived from the literature of developmental psychology and perception adaptation, as well as cognitive science and memory experiments with animals. In particular, for scalability, we increase the number of sensors of the robot as well as the number of actions and show that the robot's learned model is only increases moderately. For fault-tolerance, we deliberately sabotage the camera by twisting it by 180°, and change the underlying meanings of the actions without telling the robot. For transfer learning, we let the robot learn how to reach a hidden goal via exploration and learning, and then move the hidden goal to various different locations without informing the robot. We observe that the robot will first be surprised and then gradually forget the old goal and explore the space for the new goal. Once the new goal is found, it transfers all of its spatial knowledge learned in the previous trial and can quickly reach the new goal from any initial location in the environment.

The rest of the paper is organized as follows. Section 2 outlines the background and related work for surprise-based learning. Section 3 provides a description of the hardware setup. Section 4 describes the unified architecture, representation, the SBL process and the underlying techniques mentioned above. Section 5 presents the experimental results for scalability, fault-tolerance, and knowledge transfer between different goals. Section 6 concludes the paper with discussions and future research directions.

2. Related Work

The basic concept of surprise-based learning was first proposed by Shen and Simon in 1989 [1] and later formalized as Complementary Discrimination Learning [2-4]. This learning paradigm stems from Piaget's theory of Developmental Psychology [5], Herbert Simon's theory on dual-space search for knowledge and problem solving [6], and C.S. Peirce's method for science that "our idea of anything is our idea of its sensible effects [7]." Over the years, researchers have attempted to formalize this intuitively simple but powerful idea into an effective and general learning technique. A number of experiments in discrete or symbolic environments have been carried out with successes [8], including the developmental psychology experiments for children to learn how to use novel tools [8], scientific discovery of hidden features (genes) [1,4,9], game playing [10], and learning from large knowledge bases [11]. This paper generalizes these previous results for real robots to learn autonomously from continuous and uncertain environments.

As detailed in the next subsection, SBL has previously been compared to many different learning techniques [12] and its ability to adapt to failures has been exhaustively tested [13]. In computer science, SBL is related to several solutions for the inverse problem, such as Gold's algorithm for system identification in a limit, Angluin's L* algorithm [14] for learning finite state machines with hidden states using queries and resets, the L*-extended algorithm by Rivest and Schapire [15] using homing sequences, and the D* algorithm based on local distinguishing experiments [3]. For learning from stochastic environments, SBL is related to learning hidden Markov models [16], partially observable Markov decision processes [17], and most recently, predictive state representations [18] and temporal difference algorithms [19]. Some systems also incorporate *novelty* [20], with a flavor of surprise, into the value function of states, although they are not used to modify the learned models. The notion of *prediction* is common in both developmental psychology and AI. Piaget's *constructivism* [5] and Gibson's *affordance* [21] are two famous examples. In AI systems, the concepts of *schemas* [22] and *fluent* [23] all resemble the *prediction rules* we use here, although their primary use is not for detecting and analyzing surprises as we do here. Different from these results, this paper presents new technique and results for scalability, fault-tolerance, and transferable learning for different goals. Compared to the paradigm of reinforcement learning, SBL is model-based and offers quicker adaptation for large scaled and continuous problems.

Related working in Learning (Differences between SBL and other learners):

At present most learning algorithms can be classified as supervised, unsupervised or reinforcement learning [24]. Supervised learning (SL) requires the use of an external supervisor that may not present here. In contrast, unsupervised learning (UL) opts to learn without any feedback from the environment by attempting to remap its inputs to outputs, using techniques such as clustering. Hence, it may overlook the fact that feedback from the environment may provide critical information for learning.

Reinforcement learning (RL) receives feedback from the environment. Some of the more successful RL algorithms used in related robotic problems include Evolutionary Robotics [25] and Intrinsically Motivated Reinforcement Learning [26]. However, most RL algorithms focus on learning a policy from a given discrete state model (or a world model) and the reward is typically associated with a single goal. This makes transferring the learned knowledge to other problems more difficult.

Complementary Discrimination Learning (CDL) [8] attempts to learn a world model from a continuous state space and is capable of predicting future states based on the current states and actions. This facilitates knowledge transfer between goals and discovering new terms [27]. However, CDL is more logical-based learning and has not been applied to physical robots to learn directly from the physical world. In addition, CDL only performs generalization and specialization on a complementary rule, while other activities, such as abstraction, surprise analysis with noisy sensors, dynamic goal assignment, prediction creation and continuous maintenance, are necessary for robotic learning.

Evolutionary Robotics (ER) is a powerful learning framework which facilitates the generation of robot controllers automatically using neural networks and genetic programming. The emphasis in ER is to learn a controller given some model or detailed information about the environment, which may not be readily available. However, advances in ER such as the Exploration-Estimation Algorithm [28] proved that an internal model such as an action or sensor model of a robot can be learned without prior knowledge given that the environment and the robot can be reset to its initial configuration prior to each experiment, which is not supported in this problem.

Another promising approach is Intrinsically Motivated Reinforcement Learning (IMRL) where the robot uses a

self-generated reward to learn a useful set of skills. IMRL has successfully demonstrated learning useful behaviors [29], and a merger with ER as in [30] was able to demonstrate navigation in a simulated environment. The authors have mentioned that IMRL can cope with situated learning, but the high dimensional continuous space problem that embodiment produces, is beyond its current capability.

There has been a large amount of research in model learning as in [31] and [32], yet the majority focus on action, sensor or internal models of the robot and not the external world. In the autonomous robotic learning problem we are interested in, the learner must accommodate limited processing, noisy actuation and limited or noisy sensing available on a physical robot. Furthermore, the ability to predict and be "surprised" by any ill effects is also critical. This powerful learning paradigm has been analyzed, theorized, explored and used in a few other learning applications such as traffic control [33] and computer vision [34].

Alternatively, learning a world model could be accomplished to some extent by simply recording the environment using conventional map building such as Simultaneous Localization and Mapping (SLAM) [35]. However, accurate-enough models for the robot's actions and sensors must be given to facilitate this type of learning. These models are not available for developmental learning, which is motivated by previous researches as highlighted earlier.

Related work in Fault-Tolerant Algorithms:

Researchers have identified two main strategies to deal with failure. One strategy involves using several redundant sensors and actuators with failure detection, such that in the event of a failure the robot can disable the faulty hardware and switch to operational hardware. This strategy has been effectively implemented on several robots as discussed in [36], but the additional redundancy comes at a cost in terms of resources and requires great foresight at design time in order to maximize the robot's robustness. The second strategy involves failure detection followed by adaptation. In [37] adaptation has been demonstrated on a legged robot. This impressive result was obtained with several sensor and actuator models combined with some carefully designed control strategies, making it difficult to scale to any robot with an arbitrary number of sensors and actuators. Adaptation or resilience through self-modeling [38] has demonstrated that with little prior knowledge it is possible to learn appropriate control strategies autonomously and overcome actuator failures.

SBL capitalizes on both fault tolerance strategies. When redundant hardware such as sensors fails, SBL is able to detect the failure and ignore such features. The benefits arise from the fact that neither human intervention for changes to the control algorithm, nor amazing foresight, nor special hardware for failure detection is required. The algorithm provides adaptation by ignoring faulty features, subsequently listening to repaired features, as well as dynamically altering its world model. To our knowledge, surprise-based learning is the only fault tolerant robotics system researched to-date that is capable of dealing with sensing and actuation failures without a priori models of the sensors, actuators or the environment, while learning to accomplish its tasks.

Extensions of RL algorithms tested in the Morris water maze:

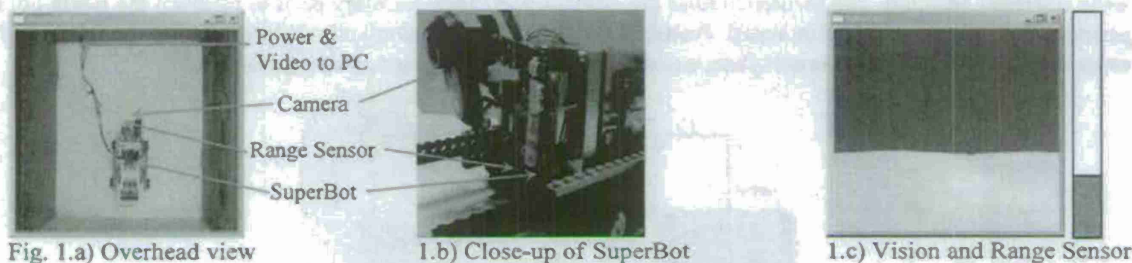
Krichmar et al. [39] tested a brain-based device called "Darwin X" on a dry version of the water maze. This robot utilized visual cues and odometry as input and its behavior was guided by a simulated nervous system modeled on the anatomy and physiology of the vertebrate nervous system. Busch et al. [40] built on this idea by simulating a water maze environment to compare an attributed probabilistic graph search approach and a temporal difference reinforcement learning approach based solely on visual cues encoded via a self-organizing map which discretized the perceptual space. Stone et al. [41] tested the simulated algorithm in a physical robot and extended the algorithm to facilitate adaptation of the reinforcement learning approach to the relocation of the hidden platform.

SBL does not discretize the perceptual space. Instead, it learns a model of the environment as a set of prediction rules. Reinforcement Learning (RL) requires a large amount of training episodes to learn a path to a single goal. SBL learns via surprises with each action it takes, so it does not need separate training episodes. Most RL algorithms focus on learning a policy from a given discrete state model and the reward is typically associated with a single goal, making it difficult to transfer learned knowledge to other goals or to other problems which are similar. This fact was noted by Stone et al. prompting an extension of the original algorithm, but even with the short term and long term rewards the trajectories of the robot were still slightly biased when the goal was relocated. SBL accommodates multiple goals and can be transferred to other problems with minimal changes as the algorithm does not require any prior information regarding the environment, the sensors or the actuators. In addition, SBL's ability to accommodate

sensor and actuator failure during runtime without any external intervention makes it an ideal candidate for a physical robot operating in a real environment such as the dry version of the water maze.

3. Hardware Description

SBL is implemented and tested on a single SuperBot module [42]. The robot is equipped with bi-directional WiFi communications, an onboard camera, a short distance range sensor, an external PC capable of wirelessly interfacing with the robot and an overhead camera overlooking the entire environment for human observations and experiment recording. The environment is enclosed with four uniquely colored walls and a discernable floor as seen in Figure 1. The onboard camera is mounted in a way that the robot loses sight of the ground plane when it is approximately 6" from the wall it's facing. The range sensor is an IR proximity detector, which typically reports a value that increases as the robot gets closer to a wall. The range sensor has been attached such that it faces the same direction as the camera, and its maximum range response is roughly 10", meaning that it returns a constant value when it is further than 10" away from the wall in sight. These sensors are generally noisy. For example, the range sensor works differently for different walls as different colors reflect IR slightly differently. The sensor data is relayed to the PC where the learning system can radio action commands back to the robot.



The robot is preloaded with four actions corresponding to the motions of forward, backward, left and right. Each action is defined as an execution pattern of the motors for a pre-specified period of time (duration), and the robot has no priori knowledge about the expected results of these actions. In the current implementation, the duration for all four actions is set to be 5 seconds, and that results in an estimated distance for the forward action to be 3.5~4.2cm, backward 2.5~4.5cm, left 13~15 degrees, and right 13~15 degrees. As expected, these numbers are noisy and uncertain, partially due to the slippage of the wheels on the uneven and non-uniform surface. Furthermore, the robot has no priori knowledge about the relationships among the actions. For example, it has no knowledge about the reversibility of the actions, such as forward (backward) may be equal to the reverse of backward (forward), and left (right) may be equal to the reverse of right (left). In fact, due to the noise in the actions, these actions are not reversible at all. To the robot, the name of an action has no pre-defined meaning and it must be learned through experience.

The onboard camera and range sensor allow the robot to make an *observation* (or *scene*) before and after each action. An observation consists of a set of identifiable *percepts* (or *sensor features*). In this robot, the percepts include blobs of colors (based on mean shift segmentation [43]), the size and the center-location of the identified blobs, and the readings of the ranger sensor, and they are the basic perceived elements to be labeled, matched, and reasoned in the scene analysis. In the scenes before and after a single action, we assume that the same color represents the continuation of the same object. (This assumption is realizable if the duration for each action is defined sufficiently small.) However, the robot has no pre-knowledge about the number and the types of the percepts/features that would be encountered in the environment, nor does it know how these percepts/features are related to the actions. In addition, all sensors are inherently noisy and uncertain.

During the scene analysis, the percepts are reasoned and compared by a set of mental operators that are given to the robot at the outset. The current operators include the presence (%) or absence (~) of a feature, the change in the size of a feature (<, <=, =, >=, >), the difference in the distance reading, and the center-location or displacement of each visual feature with respect to the frame of reference. The horizontal displacement is indicated by the 'x' prefix and the vertical displacement by 'y'. Note that coordinate system used in this system for image analysis originates from the top left corner of each image (i.e. the vertical displacement increases from top to bottom, while the horizontal displacement increases from left to right).

To summarize the description above, the robot is situated in a learning environment with three assumptions: (1) each action is defined as an execution pattern of the motors for a pre-specified period of time; (2) in the scenes before and after a single action, the same color represents the continuation of the same object; and (3) the meanings

of the comparison operators for scene analysis are given at the outset. However, the robot has no pre-knowledge about the consequences of the actions; the relationships among the actions, the degree of uncertainties in the actions, the number and the types of percepts, the relationships among the percepts, and the relationships between percepts and actions. Furthermore, the environment is a continuous space with no predefined and discretized grids.

4. Surprise-Based Learning

4.1 Integrated Architecture and Unified Representation

The architecture for surprise-based learning is shown in Figure 2. The learning robot engages in a cycle of "prediction, action, observation, detection (of surprise), analysis (of surprise), and adaptation." Actions are planned (or randomly selected) based on the current model that consists of prediction rules. Then, the predictor returns all prediction rules whose conditions and actions match the current state of the environment and the selected action. The action is executed and the resulting state becomes the new current state. If no prediction was made, a new rule is created to reflect the observed conditions and consequences of the action. If all predictions are satisfied by the new observations, then the cycle continues. Otherwise, a surprise is detected and the model will be revised to reflect the newly observed situation. The prediction rules are organized in complementary pairs to maintain the flexibility to be generalized or specialized/discriminated. Probabilities of the rules are implicitly implemented in the process of rule revision. Once rule adaptation is complete, a new cycle starts and the learning/doing continues.

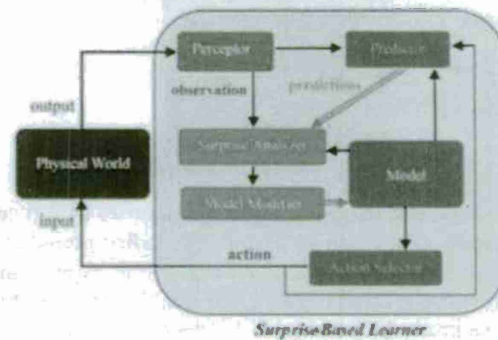


Figure 2: The Architecture of Surprise-Based Learning

The learned knowledge is represented in a model consists of prediction rules in the format of (1) below.

Rule	≡	Conditions → Action → Predictions	(1)
Condition	≡	(Feature → Operator → Value)	(2)
Prediction	≡	(Feature → Operator)	(3)

Each prediction rule is a triplet as in (1) of "condition", "action", and "prediction". Conditions are logical statements describing the state of the perceived features prior to the execution of an action. A condition can be represented as a triple as in (2). Using the comparison operators given to the robot, each condition describes a relationship of a feature and its value. For example, the expression Condition1 ≡ (feature1, >, value1) means that Condition1 is true if feature1 is greater than value1. Several logically related conditions can be grouped together to form a clause using 'And' and 'Not' logical operators. Predictions are clauses that describe the expected change in the state of the perceived features as a result of performing an action. As seen in (3) a prediction can be represented using a tuple, i.e. Prediction1 ≡ (feature1, >) means that if the rule is successful the value of feature1 will increase.

One of the important aspects of prediction rules is that they can be sequenced to form a *prediction sequence* $[s_0, a_1, p_1, \dots, a_n, p_n]$ where s_0 is the current observation at the current state, and $a_i, 1 \leq i \leq n$, are actions, $p_i, 1 \leq i \leq n$, are predictions. As the actions in this sequence are executed, the environmental states are perceived in a sequence $s_1, s_2, \dots, s_n$. A surprise occurs as soon as an environmental state does not match to its correspondent prediction. Notice that a surprise can be either "good", if the unexpected result is desirable, or "bad" otherwise. This notation is similar to the concept of "predictive state representations" recently proposed in [44], but prediction sequence can be used to represent many other concepts such as "plan," "exploration," "experiment," "example," and "advice" in a unified

fashion as follows:

A *plan* is a prediction sequence where the accumulated predictions in the sequence is expected to satisfy a goal;

An *exploration* is a prediction sequence where some predictions are deliberately chosen to generate surprises (so the learner can revise its knowledge);

An *experiment* is a prediction sequence where the last prediction is designed to generate a surprise with a high probability;

An *example* is a prediction sequence provided by an external source that the learner should go through and learn from the surprises generated during the execution of the sequence;

An *advice* is a prediction sequence that should be put into the model as it is. The learner simply translates a piece of advice into a set of prediction rules. For example, "never run over a cliff" can be translated into a prediction rule as (cliff_facing) $\rightarrow$ forward $\rightarrow$ destruction. Depending on the seriousness of the consequence in the rule, the learner can weight the advice ranging from "never" to "sometimes," depending on the tradeoff between the consequence of its action and the desired goal.

4.2 Complementary Discrimination Learning

During learning and adaptation, prediction rules are created, split, and refined according to a set of templates described in equations (4)-(10) below. These form the core of the model modifier in the SBL architecture. We first present the format of these templates and then give detailed examples and explanations thereafter.

Rule Creation Let C_0 and P_0 represent the change of a percept before and after a newly explored action, a new rule is created as follows:

$$\text{Rule}_0 = C_0 \rightarrow \text{Action} \rightarrow P_0 \quad (4)$$

Rule Splitting If a surprise is caused by a single rule (e.g. Rule_0 above), and then for each possible cause C_X identified by the analysis of the surprise, the rule is split into two complementary sibling rules as follows, where P_X is a newly observed consequence of the action with respect to C_X .

$$\text{RuleA}_1 = C_0 \wedge C_X \rightarrow \text{Action} \rightarrow P_0 \vee \neg P_X \quad (5)$$

$$\text{RuleB}_1 = C_0 \wedge \neg C_X \rightarrow \text{Action} \rightarrow \neg P_0 \wedge P_X \quad (6)$$

Rule Refinement If a surprise is caused by a RuleA that has a sibling RuleB , where ϕ_C represents the rule's current condition minus C_0 and η_P represents the prediction of the rule, as follows:

$$\text{RuleA} = C_0 \wedge \phi_C \rightarrow \text{Action} \rightarrow \eta_P \quad (7)$$

$$\text{RuleB} = C_0 \wedge \neg \phi_C \rightarrow \text{Action} \rightarrow \neg \eta_P \quad (8)$$

Then for each possible cause C_X identified by the analysis of the surprise, the rules will be refined as follows:

$$\text{RuleA} = C_0 \wedge \phi_C \wedge C_X \rightarrow \text{Action} \rightarrow \eta_P \quad (9)$$

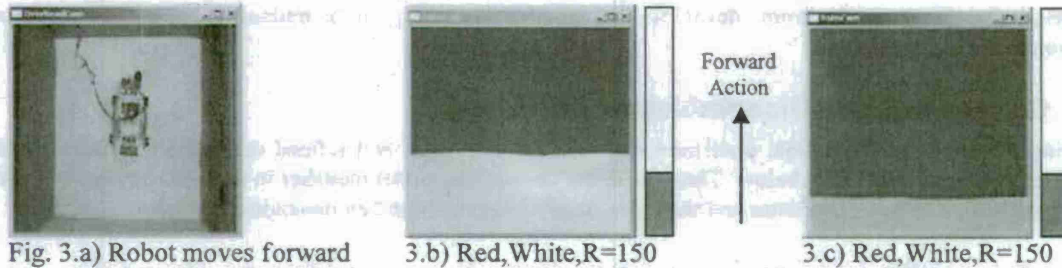
$$\text{RuleB} = C_0 \wedge \neg (\phi_C \wedge C_X) \rightarrow \text{Action} \rightarrow \neg \eta_P \quad (10)$$

Notice that (7)-(10) can be applied to any pair of complementary rules. In general, a pair of complementary rules can be refined multiple times so that as many C_X can be inserted into their conditions according to this procedure. Whenever a rule is discriminated, its complementary rule will be generalized, hence the name complementary discrimination learning.

4.2.1 Rule Creation

If the robot is exploring an action that has never been executed before, then there is a chance that new rules will be created. What it does is to apply the given comparison operators to comparing the percepts in the scenes before and after the action. If no changes can be found by these comparisons, then no further action is taken. Otherwise, for each change detected, a new rule is created according to the template (4) above, with P_O being the changed percept and C_O being the prerequisite of the change.

As an example of rule creation, consider the situation in Figure 3, where the robot first explores the action "forward" and the scenes before and after the action are in Figure 3.b) and 3.c) respectively. There are three percepts in these scenes, red (wall), white (floor), and range sensor. The rule creation mechanism detects four changes in these percepts before and after the action, and creates four new rules F1, F2, F3, and F4, listed below.



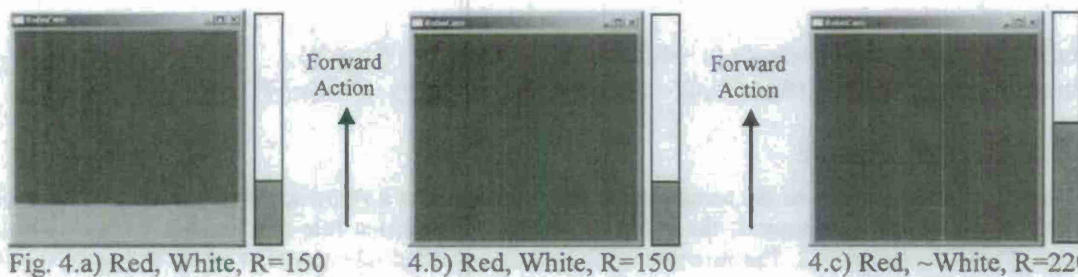
RuleF1: (White, %, 0) $\rightarrow$ FORWARD $\rightarrow$ (White, <) // the size of the white decreased
 RuleF2: (White, %, 0) $\rightarrow$ FORWARD $\rightarrow$ (White, y>) // the y-location of the white increased
 RuleF3: (Red, %, 0) $\rightarrow$ FORWARD $\rightarrow$ (Red, >) // the size of the red increased
 RuleF4: (Red, %, 0) $\rightarrow$ FORWARD $\rightarrow$ (Red, y>) // the y-location of the red increased

For each newly created rule, the robot records the two scenes before and after the action and refers to them in the future as the rule's "base-condition-scene" (e.g. Figure 3.b) and "base-consequence-scene" (e.g. Figure 3.c), respectively.

Beside this case above where there are no prediction rules for the action to be explored, rules can be created in a situation where no existing prediction rules would predict any change in the current situation. When there are predicted rules the system does not perform rule creation, instead it proceeds with rule analysis. However, after the analysis is complete, we can evaluate whether all of the predicted rules were general. If there is even one specialized rule, then proceed as normal, else there is possibility to create better rules, so invoke rule creation. This is a safe practice because it gives the model an opportunity to learn more details in a situation where a single pair of complementary rules by definition covers all possible scenarios (e.g. the robot would never learn that the floor is getting large as it backs up from a wall because the range sensor would have created a pair of rules saying that when the floor appears the range sensor reading remains equal.).

4.2.2 Surprise Detection and Analysis

As prediction rules are incrementally learned and refined, the robot uses them to make predications whenever it can. A surprise is detected if a prediction fails to be realized in the scene after the action. For example, if the robot is about to execute the forward action in a situation shown in Figure 4 where white (floor) is perceived before the action, then by RuleF1, the robot predicts that the size of white (floor) will decrease in the scene after the action. However, the white disappeared after the action and the robot detects a surprise. For each surprise, we refer to the scenes before and after the surprised action as the "surprised-condition-scene" (e.g. Figure 4.a), and "surprised-consequence-scene" (e.g. Figure 4.b), respectively.



Surprise analysis is to apply the comparison operators to compare the percepts in the base-condition-scene with those in the surprised-condition-scene. The comparison prefers differences to be "novel" rather than "ordinary" so it first considers the percepts that are not mentioned in the existing rule before it considers those that do.

If no novel difference can be found, the analysis will return causes that are considered as "ordinary" (i.e. those that are already mentioned in the current condition of the surprised rule). For example consider the very rare situation in which the robot is diagonally located in a corner between two walls (i.e. heading to a wall diagonally while the corner is on one side). As the robot executes the backward action, the range sensor reading decreases while the wall remains equal, but at a certain point its rear hits the wall and the range sensor reading becomes equal. Without any sensors at the rear to tell the differences, the analysis of surprise would return a cause that is already mentioned in the existing condition of its rule (e.g. the change in the range sensor value). If not even an ordinary difference can be detected by these comparisons, then no further action will be taken in this paper. (We considered this case further in a separate paper [4].)

After differences are detected, the analysis will return a set of C_X as the possible "causes" for the current surprise. Each cause is an expression of a percept that was true in the base-condition-scene but false in the surprised-condition-scene. In the current example, comparing Figure 3.b (the base-condition-scene) with Figure 4.a (the surprised-condition-scene) will result in two possible causes: (Red, <, Value1) and (Red, y<, Value2), where Value1 is the size and Value2 is the y-location of the red in Figure 4.a. No other causes will be returned in this case because the value of the range sensor does not change ($R=150$), and White is not as "novel" as Red.

4.2.3 Rule Splitting

Rule splitting is performed when a surprise occurs and is caused by a rule that has never been split before. For each identified possible cause C_X of the surprise, a new pair of complementary prediction rules will be created to reflect both the original consequence and the newly observed and surprised consequence (P_X). The C_X cause will be appended to the existing condition of the original rule so that the rule is specialized (i.e. more careful about making the original prediction in the future). At the same time, the system will create a new complementary rule by adding the negation of C_X in the condition so that it will make the correct prediction in the future. For completeness, these two new rules are ensured to have complementary predictions.

To illustrate the rule splitting procedure, consider the scenario in Figure 4, where the robot's forward action in Figure 4.a caused a surprise for RuleF1 because White does not decrease but disappeared in Figure 4.b. In this case, the analysis of surprise returns two causes as mentioned before, and RuleF1 will be split into two pairs of new complementary rules (Similarly, RuleF2 will be split into two new pairs of complementary rules as well):

- RuleF1.1.1: $(\text{White}, \%, 0) \wedge (\text{Red}, <, \text{Value1}) \rightarrow \text{FORWARD} \rightarrow (\text{White}, <) \vee \neg (\text{Red}, >)$
- RuleF1.1.2: $(\text{White}, \%, 0) \wedge \neg (\text{Red}, <, \text{Value1}) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White}, <) \wedge (\text{Red}, >)$
- RuleF1.2.1: $(\text{White}, \%, 0) \wedge (\text{Red}, y<, \text{Value2}) \rightarrow \text{FORWARD} \rightarrow (\text{White}, <) \vee \neg (\text{Red}, y>)$
- RuleF1.2.2: $(\text{White}, \%, 0) \wedge \neg (\text{Red}, y<, \text{Value2}) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White}, <) \wedge (\text{Red}, y>)$

As another example of rule splitting, imagine that after the splitting of RuleF1 and RuleF2, the robot does another forward action in Figure 4.b, which results in Figure 4.c. This will cause a surprise for RuleF3 and RuleF4 because they predict that the Red will increase but instead it has no change. The analysis of this surprise compares Figure 3.b (the base-condition-scene) with Figure 4.b (the surprised-condition-scene) and will return a single cause (White, %, 0) as the most novel difference. Thus, RuleF3 is split into a new pair of complementary rules as follows (so does RuleF4 in a similar way):

RuleF3.1.1: $(\text{Red},\%,0) \wedge (\text{White},\%,0) \rightarrow \text{FORWARD} \rightarrow (\text{Red},>) \vee (\text{White},\%)$

RuleF3.1.2: $(\text{Red},\%,0) \wedge \neg (\text{White},\%,0) \rightarrow \text{FORWARD} \rightarrow \neg (\text{Red},>) \wedge (\text{White},\sim)$

4.2.4 Rule Refinement

As learning continues if one of the complementary rules encounters a surprise, then for each possible cause C_X identified by the analysis of surprise, the robot will refine the surprised rule and its sibling rule according to equations (7)-(10) in section 4.2. The insertion of C_X into the surprised rule results in a new and more specialized rule as in (9). The complementary rule is generalized as in (10) by negating the entire condition (minus C_0) of the newly specialized rule, so that the two rules are kept as complementary in their conditions. The predictions of the rules are not altered and remain to be complementary.

To illustrate this dual procedure for specialization and generalization, consider the following scenario. Given that RuleF2 was split earlier as mentioned in rule splitting, one possible complementary pair of rules is as follows:

RuleF2.1.1: $(\text{White},\%,0) \wedge (\text{Red},<,\text{Value1}) \rightarrow \text{FORWARD} \rightarrow (\text{White},y>) \vee \neg (\text{Red},>)$

RuleF2.1.2: $(\text{White},\%,0) \wedge \neg (\text{Red},<,\text{Value1}) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White},y>) \wedge (\text{Red},>)$

Now assume that the robot performed a right turn action from Figure 5.a resulting in Figure 5.b. When a forward action is executed from Figure 5.b the predictor selects RuleF2.1.1 because the size of Red, Value3 is slightly less than Value1 (i.e. $\text{White} \text{ and } \text{Value3} < \text{Value1}$). The robot expects the y-location of White to increase, but it is surprised as White disappears as shown in Figure 5.c.

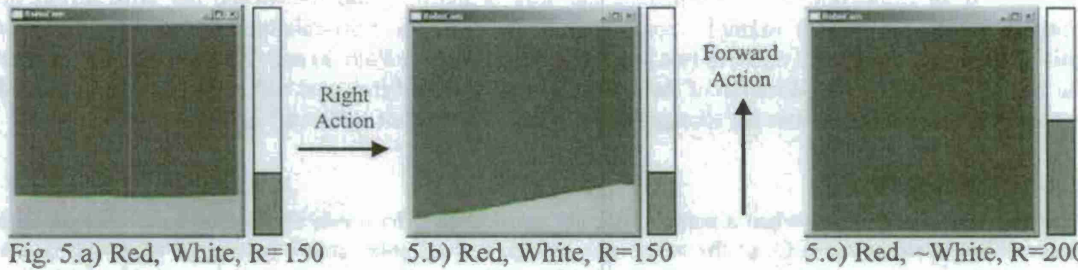


Fig. 5.a) Red, White, R=150

5.b) Red, White, R=150

5.c) Red, ~White, R=200

The analysis of this surprise will compare Figure 5.b. (the surprised-condition-scene) with Figure 3.b. (the base-condition-scene) and concludes that there are exactly six possible clauses: $(\text{Red}, y<,\text{Value4})$, $(\text{Red}, x>,\text{Value5})$, $(\text{Red}, <,\text{Value6})$, $(\text{White}, y<,\text{Value7})$, $(\text{White}, x<,\text{Value8})$, $(\text{White}, >,\text{Value9})$. The refined rules are as follows:

RuleF2.1.1.1.1: $(\text{White},\%,0) \wedge (\text{Red},<,\text{Value1}) \wedge (\text{Red}, y<,\text{Value4}) \rightarrow \text{FORWARD} \rightarrow (\text{White},y>) \vee \neg (\text{Red},>)$

RuleF2.1.2.1.2: $(\text{White},\%,0) \wedge \neg ((\text{Red},<,\text{Value1}) \wedge (\text{Red}, y<,\text{Value4})) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White},y>) \wedge (\text{Red},>)$

RuleF2.1.1.2.1: $(\text{White},\%,0) \wedge (\text{Red},<,\text{Value1}) \wedge (\text{Red}, x>,\text{Value5}) \rightarrow \text{FORWARD} \rightarrow (\text{White},y>) \vee \neg (\text{Red},>)$

RuleF2.1.2.2.2: $(\text{White},\%,0) \wedge \neg ((\text{Red},<,\text{Value1}) \wedge (\text{Red}, x>,\text{Value5})) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White},y>) \wedge (\text{Red},>)$

RuleF2.1.1.3.1: $(\text{White},\%,0) \wedge (\text{Red},<,\text{Value1}) \wedge (\text{Red}, <,\text{Value6}) \rightarrow \text{FORWARD} \rightarrow (\text{White},y>) \vee \neg (\text{Red},>)$

RuleF2.1.2.3.2: $(\text{White},\%,0) \wedge \neg ((\text{Red},<,\text{Value1}) \wedge (\text{Red}, <,\text{Value6})) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White},y>) \wedge (\text{Red},>)$

RuleF2.1.1.4.1: $(\text{White},\%,0) \wedge (\text{Red},<,\text{Value1}) \wedge (\text{White}, y<,\text{Value7}) \rightarrow \text{FORWARD} \rightarrow (\text{White},y>) \vee \neg (\text{Red},>)$

RuleF2.1.2.4.2: $(\text{White},\%,0) \wedge \neg ((\text{Red},<,\text{Value1}) \wedge (\text{Red}, y<,\text{Value7})) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White},y>) \wedge (\text{Red},>)$

RuleF2.1.1.5.1: $(\text{White},\%,0) \wedge (\text{Red},<,\text{Value1}) \wedge (\text{White}, x<,\text{Value8}) \rightarrow \text{FORWARD} \rightarrow (\text{White},y>) \vee \neg (\text{Red},>)$

RuleF2.1.2.5.2: $(\text{White},\%,0) \wedge \neg ((\text{Red},<,\text{Value1}) \wedge (\text{Red}, x<,\text{Value8})) \rightarrow \text{FORWARD} \rightarrow \neg (\text{White},y>) \wedge (\text{Red},>)$

RuleF2.1.1.6.1: $(\text{White},\%,0) \wedge (\text{Red},<,\text{Value1}) \wedge (\text{White}, >,\text{Value9}) \rightarrow \text{FORWARD} \rightarrow (\text{White},y>) \vee \neg (\text{Red},>)$

RuleF2.1.2.6.2: $(\text{White}, \%, 0) \wedge \neg ((\text{Red}, <, \text{Value1}) \wedge (\text{Red}, >, \text{Value9})) \rightarrow \text{FORWARD} \rightarrow \neg(\text{White}, y >) \wedge (\text{Red}, >)$

One of the important advantages of learning complementary rules is that even though one rule may be specialized and limited for application, its complementary rule is always general enough for further surprises and learning. For example, RuleF3.1.1 is specialized, while RuleF3.1.2 is general enough to predict changes that are either less than or equal to the original value of the feature while the other feature remains unseen.

4.3 Planning with Abstracted Rules

The purpose of learning is to solve problems. Here, the prediction rules are learned so that they can be reasoned and planned by the planner to select actions for solving any given goals in the environment. In general, the prediction rules can be viewed as the "operators" [2] in a planning system, so that any planner can be used here as long as they are compatible with the representation of the prediction rules. The job of a planner is to find a sequence of rules/actions that if they are executed they can carry the robot from the current state to a goal state. In this paper, we implemented a planner based on the standard goal-regression with a greedy search. A goal state is defined as a scene with a set of desired features.

To plan effectively, we first extract the feature transition from a pair of complementary rules. We call the extracted transition an *abstracted rule* of the original pair of complementary rules. The basic idea is that for a given pair of complementary rules, if one feature is mentioned *only* in the condition and another feature is mentioned *only* in the predictions, then there must exist a transition from the first feature to the second feature that is achievable via the action in the rules. In the representation used in questions (5)-(6) in Section 4.2, the first feature is the stem condition C_0 and the second feature is the surprised prediction P_X . So an abstracted rule is defined as:

An Abstracted Rule: $C_0 \rightarrow \text{Action} \rightarrow P_X$ (11)

For example, the abstracted rule of the complementary RuleF3.1.1 and RuleF3.1.2 at the end of Section 4.2.3 is simply:

RuleF3.1.X: $(\text{Red}, \%, 0) \rightarrow \text{FORWARD} \rightarrow (\text{White}, \sim)$

Rule abstraction is a powerful tool to infer feature transitions that must be achieved by actions. To minimize the execution time of the planner, rule abstraction is performed immediately after rule splitting. Abstract rules are marked such that rule selection will not use them as predictions, as they are only required for planning.

The robot can be assigned a target or goal scene either prior to, or during, or after the learning process. Typically a goal scene is assigned by placing the robot in a particular location and prompting it to record the desired features. Runtime goal assignment permits the robot to change goals dynamically. In fact, a plan can be considered a set of sub goals that leads to the goal scene. When planning is invoked, the next action from the plan subsumes any random action that the robot had previously decided to explore.

The abstracted rules are used by the planner to establish the high links between the goal state and the current state, and the details are filled in based on the complete world model learned so far.

Planning with abstracted rules faces the same challenges that are faced by any general planning system. In particular, the result of our planner is not always optimal in terms of the number of actions to achieve a goal. When there are multiple goals to be achieved with arbitrary orders, our greedy search method may cause oscillations in selecting which goal to achieve first. This situation can be detected but it consumes several actions while planning.

4.4 Rule Forgetting

Adaptation requires both learning new and useful knowledge and forgetting obsolete and incorrect knowledge. A prediction rule may be obsolete because the robot's sensors or actions are changed or damaged, or the environment poses new tasks or situations. To deal with unexpected situations, a learning robot should be prepared to consider all possible hypotheses of the combinations and relationships of its sensors and actions during its adaptation. Naturally, not all hypotheses are correct and those that are wrong should not always reside in the memory.

Rule forgetting is a technique derived to deal with incorrect or useless prediction rules. It is achieved by marking certain rules "inappropriate" so that they are used or refined. They are still kept in memory as a "reminder" so that

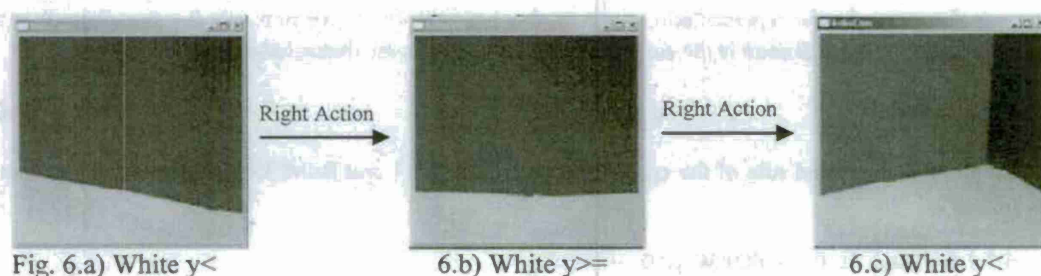
they will not be recreated again in the future. Specifically, when new rules are being constructed the system checks each new rule against valid rules as well as the rejected rules to ensure uniqueness prior to adding it to the world model.

A prediction rule is determined to be inappropriate in two ways: they cause contradictions, or they consistently and consecutively cause surprises. A contradiction occurs immediately after rule splitting and means that neither of the complementary rules can describe the surprised consequence correctly. This can be caused by two reasons: either the rule was created with a wrong C_0 in the first place (i.e. the feature C_0 was irrelevant to the current action), or the analysis of the surprise just returned an incorrect cause C_X , along with an incorrect prediction P_X . Either way, these two rules are self-contradicting and have no future in learning.

The second way to reject rules is to keep track of the progress of a pair of complementary rules. Each time a surprise occurs and the complementary rules are refined, it should imply that on the subsequent selection of either rule, the robot should produce accurate predictions. However, if these rules fail consistently or consecutively (i.e. a rule fails followed immediately by a failure of its complementary rule), then it can be inferred that the feature relation captured in the rule is inappropriate for the given context and that the rules should be rejected.

When complementary rules are rejected, it is important to reject any abstract rules that were spawned by them, because the relationship might no longer be valid.

To illustrate the procedure of rule rejection, consider an example where the robot executes a right-turn action in Figure 6.a and the action results in Figure 6.b. A pair of complementary rules RuleF5.1.1 and RuleF5.1.2 is created below.



RuleF5.1.1: $(\text{White}, \%, 0) \wedge (\text{Red}, <, \text{Value5}) \rightarrow \text{RIGHT} \rightarrow (\text{White}, y<) \vee \neg (\text{Red}, >)$
 RuleF5.1.2: $(\text{White}, \%, 0) \wedge \neg (\text{Red}, <, \text{Value5}) \rightarrow \text{RIGHT} \rightarrow \neg (\text{White}, y<) \wedge (\text{Red}, >)$

A subsequent right-turn action in Figure 6.b results in Figure 6.c. This causes RuleF5.1.1 to fail as the vertical displacement of White (floor) remained the same ($y \geq$) instead of decreasing ($y <$). On the next right-turn action the complementary rule RuleF5.1.2 is selected, which states that the vertical displacement of the floor should remain the same. Yet, this rule also fails, resulting in consecutive failures of the complementary rules. Hence, RuleF5.1.1 and RuleF5.1.2 are rejected.

4.5 Feature Relevance

Feature relevance is critical for a number of capabilities of an autonomous robot. First, it is critical for learning new knowledge. While learning how to navigate in the environment, a robot must couple the relevant features with the relevant actions. This could be situation or time dependent. For example when turning, the robot should use location information rather than size information to avoid ambiguity.

Second, feature relevance is critical for adaptation. Due to unexpected changes in the environment, certain features that were previously relevant to certain actions may become irrelevant to those actions. A learning robot must detect such cases quickly so that it can adapt to the new situations. In our experiments, we deliberately switch the meaning of actions (e.g. backward to forward and vice versa) or invert the robot's camera (similar to human wearing inverted vision goggles [46]) during the learning process.

Third, feature relevance is critical for recovering from the damaged or failed sensors and actions. If a sensor is returning data that has no correlation with a particular action, then the robot should be able to conclude that this sensor is irrelevant to that action. If a sensor returns random or constant data, then the robot should detect that the sensor may be damaged or unreliable for learning new knowledge. In our experiments, we simulate the failures of

sensors by deliberately introducing random or constant sensors to the robot.

Fourth, feature relevance is critical for the scalability of a learning algorithm. In real-world situations, the perceived information by a robot can often be overwhelming and not all the perceived features are relevant to the tasks in hand. The robot must ignore the irrelevant and focus on the relevant or else its learning process will quickly become saturated. In the learning environment in this paper, the number of perceivable features can increase rapidly as new sensors and actions may be dynamically introduced during runtime. Each sensor can have multiple modalities, meaning that the data it returns can be processed to extract one or more features for analysis. For example, a camera returns information regarding color, size, and location, while a range finder returns a distance measure.

Our approach to feature relevance is as follows. The relevance of features are recorded in a table in which each row uniquely identifies a feature representing the modality of a sensor, an action and an indicator corresponding to its relevance during learning. The relevance flag is updated depending on the following four scenarios:

1. If a feature does not change its value for any action, then despite that the feature may remain active it will not be used in rule creation, splitting or refinement, for that action. E.g. a sensor reading the constant temperature of the room has no relevance to the robot's motion.
2. For a given action, if all rules that contain this feature as its first condition C_0 have been rejected, then the feature is flagged as irrelevant to that action. For example, selecting a sensor that returns random values will eventually cause all rules that use it to fail and be rejected.
3. For a given action, when many rules containing the same feature consecutively fail then the feature will be flagged as irrelevant. For example, the size of the floor will be deemed as irrelevant to the turning action because it results in consecutive failures as it causes ambiguities when turning at the corner of walls.
4. When none of the active features for a given action register any change (this could be caused by an unexpected change in the robot's hardware), then reactivate those flagged features that indicated change for future evaluation. A reactivated feature will be considered from the next learning cycle onwards. We call this reactivation as "forced relevance" which represents enlarging the attention of the robot. For example, when the camera is unexpectedly rotated by 180° , the total initial confusion will flag many features as irrelevant. At this point the robot recognizes the lack of progress and re-evaluates its sensors and actuators in an attempt to improve the quality of its world model.

Feature relevance works during rule creation, rule splitting and refinement. SBL performs a lookup on the table and ignores any features that are considered irrelevant during the analysis of surprises. When a rule is newly created, a copy of it (called trace rule) will be recorded to ensure that after the original rule is split, refined and rejected, it would not be recreated to repeat the same learning process. As we can see, the combination of rule rejection, trace rules, and focus of attention on relevant features help to increase the speed of learning while reducing the wastage of resources. However, if a forced relevance is invoked, SBL will purge the trace rules, rejected rules and abstract rules that refer to the toggled feature to facilitate relearning.

4.6 Goal Association and Transfer

With the mechanisms described so far, SBL is able to construct a suitable model of the environment through the execution of a sequence of random actions over a period of time even in the presence of faulty hardware. During learning, if the robot is shown a "goal scene" and asked to navigate to the goal scene, it will attempt to generate a plan to reach the goal scene. In this case, the robot knows what to look for because the goal scene is given.

When a goal is not directly observable, then the feedback is not incremental and immediate but only available when the robot is at the goal location. For example, suppose the goal is a submerged platform in a body of water and the robot must swim to look for it. Then, it cannot know what to look for until it is standing on the platform. To find such "hidden" goals, the best strategy seems to explore and learn the environment as much as possible while searching for the goal. Once the goal is reached, the robot should remember the goal location by associating it with as many visual clues as possible so that it knows what to look for when it visits the goal again.

A mechanism for making this association is implemented in our learning robot to test the ability to learning to reach a hidden goal, as well as transferring knowledge if the location of the goal is secretly moved. The strategy here is to record the associations as a set of "goal scenes" and maintain them as a dynamic list of goal scenes. A new goal scene is added to the list only if a similar scene is not currently in the list. In addition to recording the goal scenes, two statistics corresponding to the number of successes and failures for each scene are noted. The number of

successes is incremented each time a similar goal scene is encountered. Note that the robot is allowed to continue randomly moving in the environment and learning even after it has successfully located the hidden area. This provides the robot with an opportunity to record multiple goal scenes with varying levels of success.

At any subsequent time after the initial discovery, if the robot is prompted to track back to the hidden goal, it will invoke the planner by selecting the best goal scene from the list and follow the sequence of actions required to make that scene appear. Some scenes are unambiguous meaning that the features captured in the goal scene are unique. For example, a goal scene comprised of a corner where two colored walls and the floor intersect is unique within the environment (e.g. a scene such as Figure 6.c); hence, it can be located by matching the presence of features while considering the size of each feature. In contrast, most goal scenes are ambiguous due to the lack of information, such as a scene with just one wall (e.g. a scene such as Figure 4.c), which can be encountered anywhere along a line parallel to that wall. When there is ambiguity in a goal scene, the robot could successfully reach it but not receive the feedback that the goal has been reached. In this case, the number of failures attached to that particular goal scene is incremented. It is important to realize that the planner might not be able to find a valid sequence of actions for some goal scenes as there might be insufficient rules in the world model at that time (especially because SBL has no a priori knowledge hinting that actions are reversible), in which case the robot resorts to selecting random actions forcing scene changes until a valid plan can be generated.

The ratio of successes to failures for a goal scene indicates the probability of finding the hidden platform using that goal scene. Intuitively, the strategy of the planner is to select the best goal scene by picking goal scene with the highest probability and following the planned set of actions to reach the goal. When the goal scene is reached, if the hidden area is not found, then the probability of that scene is lowered as the failures increase and the next most probable scene will be selected and tracked. Maintaining the dynamic list of goals and its statistics while planning and learning makes SBL robust and adaptive. If the planner tracks every goal scene in the dynamic goal list and does not come across the hidden goal, SBL concludes that the hidden goal has moved and proceeds to execute random actions. This automatically facilitates scenarios where the hidden platform is randomly relocated after it has been discovered during experimentation.

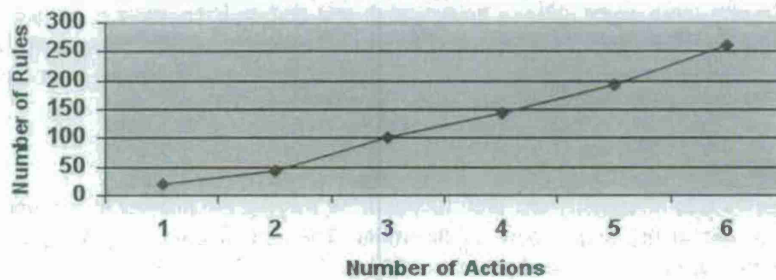
5. Experimental Results

5.1 Scalability

Autonomous robots are required to operate in many different environments equipped with a variety of sensors and actuators. The success of any learning algorithm depends on its ability to cope with the complexity of the task, which grows with the size of the environment, the number of sensors, and the number of actuators or actions. In our experiments, we mainly tested how the complexity would vary with the number of sensors and actions. We measured the performance by the time taken to accomplish a particular task and the amount of resources consumed during that time (or the number of executed actions). The amount of resources consumed is measured as the size of the world model (or the number of active prediction rules). Overall, the scalability of SBL was measured by observing the change in the number of rules against the number of executed actions for a particular task as the number of sensors and actuators or actions were gradually increased.

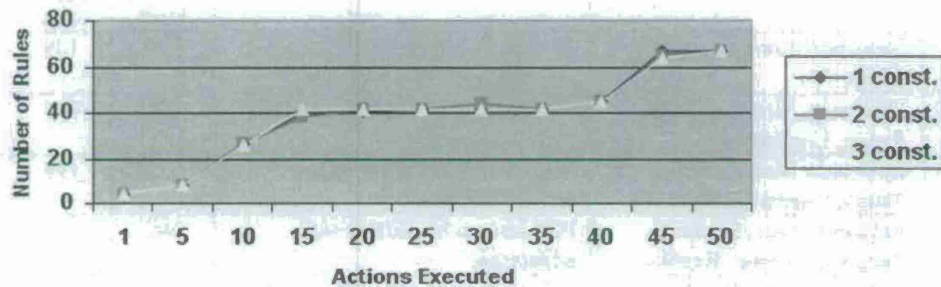
Three experiments were conducted to test the scalability of SBL. In the first experiment scalability was tested by gradually increasing the number of actions available to the robot and recording the number of rules executed until the model of the environment could no longer be improved i.e. there were no consecutive surprises over a large period of time. The second experiment consisted of increasing the number of constant valued sensors serving as features to SBL while it executed a fixed sequence of actions i.e. A sequence of 50 actions was repeated. The final experiment was conducted by increasing the number of random sensors provided to SBL while it performed the same sequence of actions defined in the previous experiment.

Fig 7. Scalability in actions



As seen in Figure 7, SBL scales linearly as the number of actions is increased. In particular they were increased by adding the actions forward, backward, left, right, large left and large right in that sequence.

Fig 8. Scalability in constant sensors



Irrelevant sensors are added as two types: sensors return only constant or random values. The response curves against adding the constant sensors shown in Figure 8 are almost identical throughout the three experiments in which the 1st experiment had one constant sensor, the 2nd experiment had two constant sensors and the final experiment had three constant sensors. Note that as the 50 executed actions were the same throughout each experiment, the slight change in the number of rules was caused by noisy sensing and actuation of the other sensors onboard the robot. So in general, the results indicate that SBL can scale to an arbitrary number of constant or failed sensors.

Fig 9. Scalability in random sensors

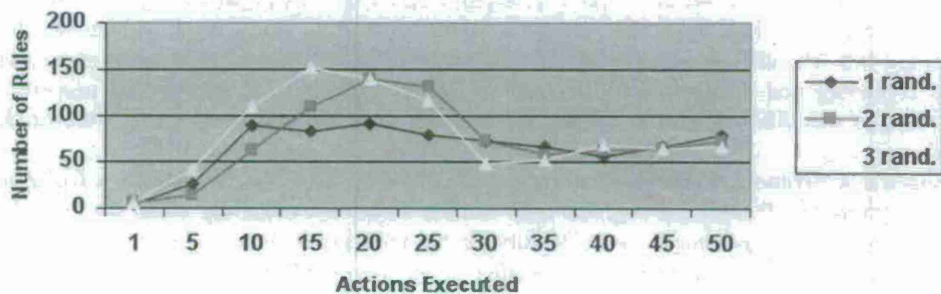


Figure 9 shows the three experiments where the number of random sensors was increased by one each time. In each experiment the response describes how SBL is reacting to the random data. Notice that the random sensors have no correlation even amongst the three experiments. This is visible by the fact that the curves have peaks and

troughs at different times, which means that there is no way to reproduce any of the curves on subsequent runs with the same number of sensors. Nevertheless, a trend can be observed in each experiment. The number of rules grows exponentially with the number of random sensors during the first few actions. However, as more actions are taken contradictions in the predictions force many rules to be forgotten and feature irrelevance is invoked to ignore the random sensors during subsequent learning. This is visible from the relatively linear response of the tail of each curve. So in general, SBL can scale to a reasonable number of random or irrelevant sensors over a period of time, but it will experience an explosion of rules until the sensors are deemed irrelevant.

5.2 Fault Tolerance

Five experiments were devised to thoroughly test fault tolerance by varying the number of sensors, altering them at runtime and toggling two out of the four actions of the robot. The experiments were designed in increasing complexity, such that the first experiment tested the minimal configuration of hardware required to learn the environment for successful navigation to a goal scene, the second and third experiments tested the scalability of learning to an arbitrary number of sensors and adaptation to irrelevant data, the fourth experiment tested sensor failure and subsequent recovery, while the fifth experiment tested actuator failure and recovery.

Table 1: Details of SBL experiments

#	Features / Sensors Available	Description	Rules		Avg. Actions
			Max.	Avg.	
1	Existence, Size, Location, Range	Minimal configuration to test the original capabilities of SBL	420	148	88
2	Existence, Size, Location, Range, Constant	Add a constant sensor to test feature relevance	426	152	92
3	Existence, Size, Location, Range, Constant, Random	Add a random sensor to test feature relevance	512	155	103
4	Existence, Size, Location, Range, Constant, Random	Flip camera by 180° to test adaptation	600	144	136
5	Existence, Size, Location, Range, Constant, Random	Switch the left-turn action with the right-turn action	740	139	152

Table 1 displays the experiment number, the features used for the corresponding experiment, a brief description of the objective, followed by the maximum and average number of rules learned and the average number of actions taken during the learning process. In this implementation, rule rejection flags and ignores rules instead of deleting them. Therefore, the maximum number of rules indicates the total number of rules explored since the commencement of the experiment. In contrast, the average number of rules only considers the rules that are active during learning because they represent the current world model. Each experiment was conducted several times by varying the starting location & orientation of the robot and terminated only after the robot demonstrated successful tracking of several randomly assigned targets.

The objective of the first experiment was to establish a base case for comparison against subsequent experiments. The robot is given only the correct and relevant sensors and actions. The average number of rules learned in this experiment is 148. The difference between the maximum number of rules 420 and the average number is evidence that rule forgetting and focus-of-attention through feature relevance is functioning properly. An interesting observation was that SBL learned to ignore the floor and vertical displacement for turning due to numerous surprises.

In the second experiment, a sensor reporting a constant value was added to simulate a faulty sensor. The robot managed to ignore this feature, proving that such failures would not adversely affect the learning or the learning time. This satisfies the expectations for scalability. In the third experiment, a completely irrelevant sensor which produced random numbers that have no correlation to the environment or the actions was added. Initially, the algorithm constructed several rules using this sensor and as soon as a contradiction occurred or after several consecutive failures the feature was flagged as irrelevant and was ignored during subsequent learning. This demonstrated adaptation to sensor change.

During the fourth experiment, the camera was deliberately rotated 180° after the robot had learned a good model of the environment. This immediately caused certain rules, such as the ones associated with feature location to cause

contradictions. These features were flagged as irrelevant (causing the average number of rules to dip slightly as the attached rules disappeared), but were subsequently toggled by forced relevance because the robot was not making any progress. The swapping of the left and right turn actions in the fifth experiment proved that SBL can cope with actuation errors gracefully.

5.3 Transfer Learned Knowledge across Goals

In neuroscience, the water maze [47] is a behavioral procedure designed by Richard G. Morris to test spatial memory. Typically, the water maze task consisted of a rat or mouse placed in a small pool of opaque water, which contained an escape platform hidden a few millimeters below the surface of the water. Some visual cues such as colored shapes were placed around the pool in plain sight of the animal such that it could learn to move to the location of the platform from any subsequent release location. Experiments proved that the time taken to reach the platform or latency on subsequent releases decreased, indicating that the animal had successfully learned the environment. This experiment is well suited to test learning and adaptation in certain robotic learning algorithms such as Reinforcement Learning and SBL.

A dry version of the water maze environment was used to test the learning and adaptation of a modular robot running the surprise-based learning algorithm. The robot was released from different starting locations within a boxed environment consisting of four uniquely colored walls and a floor. A hidden platform was marked such that the robot would detect it only if it had moved into the designated area. Several experiments were carried out by varying the starting location of the robot as well as the location of the hidden platform, such that the time taken to reach the platform could be recorded and compared against other robotic learning algorithms.

The focus of this experiment was to evaluate the feasibility of using SBL to solve the Morris water maze procedure. This is an interesting problem as biologic entities like small animals have demonstrated their ability to learn in such situations, making it a good benchmark test for a learning algorithm that attempts to reproduce similar intelligence.

The hidden platform was marked as a small rectangle on the overhead camera unknown to the robot, and whenever the robot partially or fully covered the view of the floor in this area, SBL would be notified that it is within the hidden area. This hidden platform was randomly moved to another location within the box during certain experiments.

Experiments showed that SBL can successfully solve the water maze. When the robot was first released from any arbitrary starting location, SBL randomly explored and located the hidden platform. The robot captured several goal scenes while inside the hidden area and stored it in a dynamic list with some success and failure statistics. On subsequent attempts when the robot was released from any starting location, it was able to plan its way to the hidden platform by cycling through the goal scenes based on the probability computed from previous successes. At some point in time as the hidden platform was randomly relocated, SBL visited each goal scene in its dynamic list while continuously updating the statistics for each scene and added new goal scenes if it came across any. This ensured that if it stumbled across the new location it would reinforce the information pertaining to it. Yet, there were times when all the scenes in the list were visited and the hidden platform was not located. Then SBL concluded that the platform had been relocated and switched to random actions. In comparison to most model-less learning algorithms such as reinforcement learning, SBL demonstrated faster tracking and far less training especially when the hidden platform was relocated.

5.3.1 Learning to Reach a Hidden Goal

In this scenario the location of the hidden platform remained unchanged throughout each experiment. A minimum of 9 trials were made during each experiment. Each trial was started by placing the robot at particular starting location within the box and terminated when the robot reached the hidden platform. The number of actions executed during each trial was recorded. To improve the goal probabilities, the robot was allowed 10 random actions after reaching the hidden area so that it could learn new goal scenes and reinforce its existing goal knowledge. After three trials the starting point was randomly moved to a new location within the box. The results of the 3 experiments are presented in Figure 10. As one can see, once the robot reaches the hidden goal, the subsequent visits to the goal takes much less number of actions and after four trials the robot learned to go to the hidden goal directly.

Fig 10. Fixed hidden platform, new starting point at trial 1, 4 & 7

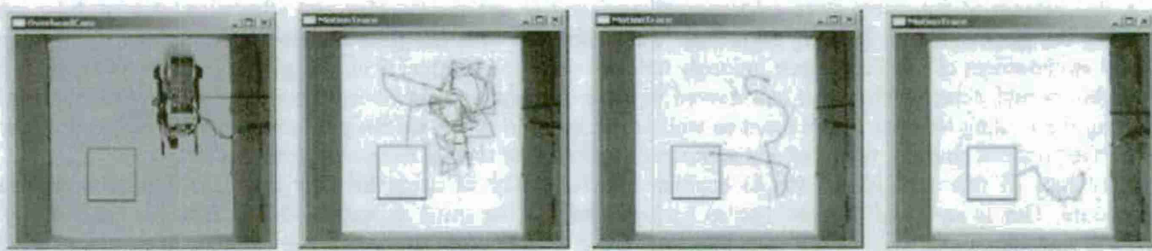
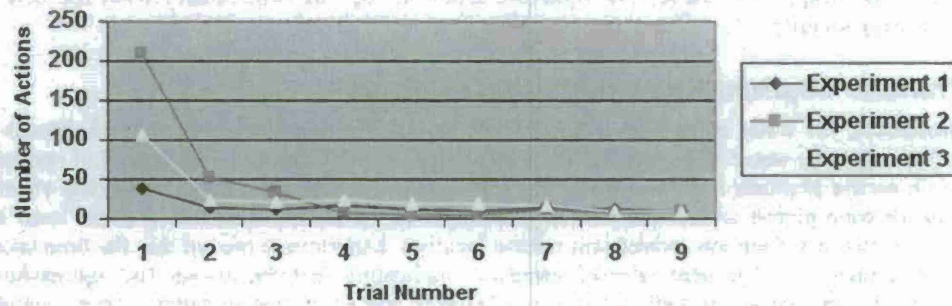


Fig. 11: a) Initial b) Trial 1 c) Trial 2 d) Trial 7

Some of the data from the above experiment is presented in Figure 11 as an example of the data analyzed to generate the graph in Figure 10. Figure 11a shows the starting location of the robot as well as the location of the hidden platform. Figure 11b is a trace of the path explored to find the hidden area from the initial release i.e. trial 1. Notice that the robot moves in arcs when it is turning and moves in straight lines for the forward and backward actions, but at times it is subjected to slippage due to the change in friction of the floor and also when it impacts walls. Figure 11c is a trace of the path taken when tracking from the starting position to the hidden platform during trial 2. As is visible, the robot had learned the location of the hidden platform and was able to navigate to it using the world model. Figure 11d details the results of trial 7, in which the robot was placed at a new starting location. SBL tracks the hidden area very quickly due to a good world model and good goal scenes that are available at this point in time.

These results verify that SBL is able to learn the water maze and unlike most model-less learning techniques it is able to navigate from any starting location and does not require a large number of training episodes. To be precise, SBL does not differentiate between training episodes and testing episodes, instead it interleaves learning and testing such that it is continuously improves its world model as well as goal scenes without any explicit resets. The diminishing number of actions during each subsequent trial is a clear indicator of this.

5.3.2 Transfer to Reach a New Hidden Goal

In this scenario the location of the hidden platform was changed during the course of each experiment. Once again a minimum of 9 trials were made during experiment. Each trial was started by placing the robot at the same starting location within the box and terminated when the robot reached the hidden platform. The number of actions executed during each trial was recorded. To improve the goal probabilities, the robot was allowed 10 random actions after reaching the hidden area so that it could learn new goal scenes and reinforce its existing goal knowledge. After three trials the hidden platform was randomly moved to a new location. The results of the three experiments are presented in Figure 12.

Fig 12. The platform is moved to a new location at trial 1, 4 & 7

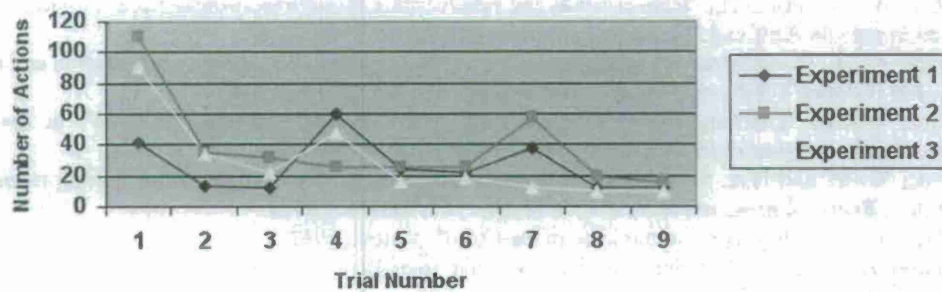


Fig. 13: a) Initial b) Trial 1 c) Trial 2 d) Trial 7

Some of the data from the above experiment is presented in Figure 13 as an example of the data analyzed to generate the graph in Figure 12. Figure 13a shows the starting location of the robot as well as the location of the hidden platform. Figure 13b is a trace of the path explored to find the hidden area during trial 1. The robot found the hidden platform after performing 42 random actions. Figure 13c is a trace of the path taken when tracking from the starting position to the hidden platform during trial 2. Once again the robot is able to track as it has a sufficient world model. Figure 13d details the results of trial 7, in which the robot was placed at the same starting location and the hidden platform was randomly relocated. Since the goal location was found during trial 1, then found again after the random relocation at trial 4, SBL has several goal scenes. It attempted to track to the last known location and was switching between the different goal-scenes when it happened to come across the new hidden location. This was not the case during trial 4 as SBL was able to conclude that the goal-scene had moved by testing each associated goal-scene in its dynamic list. Notice that when the hidden goal is moved, the robot finds it again using a less number of actions compared to the first time when it had no knowledge about the environment. We conclude that the knowledge learned from previous trials are clearly useful in identifying and relocating the new goal. It does not need to completely start from the scratch because the learned model can be transferred to the new tasks.

6. Conclusion

This report presents the surprise-based learning algorithm that enables a robot to learn from an environment, adapt to unexpected changes in sensors and actions, and solve the Morris water maze procedure autonomously, with no prior knowledge about the environment, its sensors, or the consequences of its actions. SBL proceeds by learning and relearning a model of the environment using prediction rules. Each prediction rule describes the observation of the environment prior to the execution of an action, and forecasts or predicts the observation of the environment after the execution of this action. The robot learns by investigating "surprises", which are inconsistencies between the predictions and the observed outcome. Therefore, each time a surprise occurs, the robot would attempt to identify the cause of the surprise and update its world model accordingly. This facilitates adaptation to changes in the environment. Rule rejection or the ability to detect and discard inappropriate rules, permits SBL to generate many prediction rules and then quickly prune them in an effort to converge the world model to a good representation of the environment. Similarly, feature relevance permits SBL to learn the sensor-actuator coupling autonomously. Hence, SBL is able to adapt to both changes in the environment and changes in the robot's sensors and actuators, making it truly adaptive and fault tolerant.

7. References

- [1] Shen, W.-M., Simon H., "Rule creation and rule learning through environmental exploration", Eleventh Joint Conference on Artificial Intelligence, Morgan Kaufmann. 1989.
- [2] Shen, W.-M., "Complementary discrimination learning: a duality between generalization and discrimination", *Eighth National Conference on Artificial Intelligence*, MIT Press. 1990
- [3] Shen, W.-M., "Learning finite automata using local distinguishing experiments", in the *Proceeding of International Joint Conference on Artificial Intelligence*, 1993.
- [4] Shen, W.-M. and H.A. Simon, "Fitness requirements for scientific theories containing recursive theoretical terms", *British Journal of Philosophy of Science*, 44, 641-652, 1993.
- [5] Piaget J., "The Origins of Intelligence in the Child", Norton, 1952.
- [6] Simon H., Lea G., "Problem solving and rule induction: A unified view", Knowledge and Cognition, Hillsdale, NJ, 1974.
- [7] Peirce C.S., "How to make our ideas clear", in *Popular Science Monthly*, 12: 286-302, 1878.
- [8] Shen W.-M., "Autonomous Learning From The Environment", New York, W.H. Freeman and Company, 1994.
- [9] Shen, W.-M., "The process of discovery", in *Foundations of Science*, 1(2), 1995.
- [10] Shen W.-M., "Discovery as Autonomous Learning from the Environment," *Machine Learning Journal*, vol. 12, Aug. 1993, pp. 143-165.
- [11] Shen, W.-M., "Discovering regularities from large knowledge bases", *International Journal of Intelligent Systems*, 7(7), 623-636, 1992.
- [12] Ranasinghe N., Shen W.-M., "The Surprise-Based Learning Algorithm", USC ISI internal publication, April 2008, ISI-TR-651.
- [13] Ranasinghe N., Shen W.-M., "Surprise-Based Learning for Fault Tolerant Robotics", Robotics and Automation, ICRA 2009, May 2009. (*Pending Acceptance*)
- [14] Angluin, D. "Learning regular sets from queries and counter-examples", *Information and Computation*, 75(2), 1987.
- [15] Rivest, R., Schapire R., "Inference of finite automata using homing sequences", *Information and Computation*, 1993.
- [16] Murphy, K. (2004). Hidden Markov Model (hmm) toolbox for matlab.
<http://www.ai.mit.edu/~murphyk/Software/HMM/hmm.html>
- [17] Cassandra, A. (1999). Tony's POMDP file repository page.
<http://www.cs.brown.edu/research/ai/pomdp/examples/index.html>
- [18] Wolfe, B., M.R. James, S. Singh, "Learning Predictive State Representations in Dynamic systems without reset", in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.
- [19] Sutton, R.S. and B. Tanner, "Temporal-difference networks", in *Advances in neural information processing systems*, 17:1377-1384, 2005.
- [20] Hurang X., Weng J., "Novelty and reinforcement learning in the value system of developmental robots" 2nd Intl. Workshop on Epigenetic Robotics, 2002.
- [21] Gibson, J., "The Ecological Approach to Visual Perception", Houghton Mifflin. 1979.
- [22] Drescher, G., "Made-Up Minds: A Constructivist Approach to Artificial Intelligence", MIT Press, 1991.
- [23] Cohen, P.R., M.S. Atkin, T. Oates, C.R.Beal, "Neo: Learning conceptual knowledge by sensorimotor interaction with an environment", in *Proceedings of International Conference on Intelligent Agents*, 1997.
- [24] Koslowski, B., J. Bruner, "Learning to use a lever", *Child Development*, 43:790-799, 1972.
- [25] Nolfi S., Floreano D., "Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines", MIT Press, Cambridge, MA, 2000.
- [26] Stout A., Konidaris G., Barto G., "Intrinsically Motivated Reinforcement Learning: A Promising Framework For Developmental Robot Learning", AAAI Spring Symposium on Developmental Robotics, 2005.
- [27] Shen W.-M., "Learning from the Environment Based on Actions and Percepts" Ph.D. dissertation, Dept. Computer Science., Carnegie Mellon University., Pittsburgh, PA, 1989.
- [28] Lipson H., Bongard J., "An Exploration-Estimation Algorithm for Synthesis and Analysis of Engineering Systems Using Minimal Physical Testing", ASME Design Engineering Technical Conferences, Salt Lake City, UT, 2004.
- [29] Oudeyer P., Kaplan F., Hafner V., "Intrinsic Motivation Systems for Autonomous Mental Development", *IEEE Transactions on Evolutionary Computation*, Vol. 11, 2007.

- [30] Schembri M., Mirolli M., Baldassarre G., "Evolving internal reinforcers for an intrinsically motivated reinforcement-learning robot", IEEE International Conference on Development and Learning, 2007.
- [31] Pierce D., Kuipers B., "Map learning with uninterpreted sensors and effectors", Artificial Intelligence, 92: 169-229, 1997.
- [32] Stronger D., Stone P., "Towards Autonomous Sensor and Actuator Model Induction on a Mobile Robot", Connection Science, 18(2), June 2006, pp. 97-119.
- [33] Horvitz E., Johnson A., Sarin R., Liao L., "Prediction, Expectation, and Surprise: Methods, Designs, and Study of a Deployed Traffic Forecasting Service", Twenty-First Conference on Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 2005.
- [34] Itti L., Baldi P., "A Surprising Theory of Attention", IEEE Workshop on Applied Imagery and Pattern Recognition, Oct 2004.
- [35] Thrun, S., "Robotic Mapping: A Survey", Exploring Artificial Intelligence in the New Millenium, Morgan Kaufmann, 2002.
- [36] Hamilton D., Visinsky M., Bennett J., Cavallaro J., Walker I., "Fault tolerant algorithms and architectures for robotics", Electrotechnical Conference, Apr 1994, 3: 1034-1036.
- [37] Ferrell C., "Failure recognition and fault tolerance of an autonomous robot", Adaptive Behavior, 1994, 2: 375-398.
- [38] Bongard J., Zykov V., Lipson H., "Resilient machines through continuous self-modeling", Science, Nov. 2006. 314: 1118-1121.
- [39] Krichmar J., Nitz D., Gally J., Edelman G., "Characterizing functional hippocampal pathways in a brain-based device as it solves a spatial memory task", Proc. National Academy of Science USA, February 2005, 102 (6): pp. 2111-2116.
- [40] Busch M., Skubic M., Keller J., Stone E. "A Robot in a Water Maze: Learning a Spatial Memory Task", Robotics and Automation, ICRA 2007, April 2007, pp. 1727-1732.
- [41] Stone E., Skubic M., Keller J., "Adaptive Temporal Difference Learning of Spatial Memory in the Water Maze Task", Development and Learning, ICDL '08, August 2008, pp. 85-90.
- [42] Salemi B., Moll M., Shen W., "SUPERBOT: A Deployable, Multi-Functional, and Modular Self-Reconfigurable Robotic System", Intelligent Robots and Systems, IROS 2006, October 2006, pp.43-52.
- [43] Comaniciu D., Meer P., "Mean Shift: A Robust Approach toward Feature Space Analysis", Pattern Analysis and Machine Intelligence, May 2002, 24: pp. 603-619.
- [44] Littman, M.L., Sutton, R.S., Singh, S. "Predictive representations of state. In *Advances in neural information processing systems*", 14, 1555-1561. 2002.
- [45] Ranasinghe N., Shen W-M, "Surprise-Based Learning for Developmental Robotics", Learning and Adaptive Behaviors for Robotic Systems, LAB-RS '08, August 2008, pp. 65-70.
- [46] Linden D., Kallenbach U., Heinecke A., Singer W., Goebel R., "The myth of upright vision. A psychophysical and functional imaging study of adaptation to inverting spectacles", Perception, 1999, (28):469 - 481.
- [47] Morris R, "Developments of a water-maze procedure for studying spatial learning in the rat". Journal of Neuroscience Methods, May 1984, 11 (1): pp. 47-60.

An Accumulative List of People Involved in the Effort

Dr. Wei-Min Shen (PI, Professor)

Mr. Nadeesha Ranasinghe (PhD student), whose thesis is on the objective of the project: Surprise-Based Learning.

Dr. Behnam Salemi (Computer Science Researcher).

Michael Rubenstein (PhD student)

Jacob Everist (PhD student)

Feili Hou (PhD student)

C.H. Chiu (PhD student)

Publications Stemming from the Effort

1. Nadeesha Ranasinghe and Wei-Min Shen. Surprise-based developmental learning and experimental results on robots. *International Conference on Developmental Learning*, Shanghai, China, June 2009.

2. Feili Hou, Nadeesha Ranasinghe, Behnam Salemi, and Wei-Min Shen. Wheeled Locomotion for Payload Carrying with Modular Robot. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Nice, France, September 2008.

3. Nadeesha Ranasinghe and Wei-Min Shen. The Surprise-Based Learning Algorithm. *Technical Report ISI-TR-651*, USC Information Sciences Institute, 2008.

4. Nadeesha Ranasinghe and Wei-Min Shen. Surprise-Based Learning for Developmental Robotics. *ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems*, Edinburgh, Scotland, August 2008.

5. Feili Hou, Nadeesha Ranasinghe, Behnam Salemi, and Wei-Min Shen. Remotely-Controlled Autonomous TricycleBot Locomotion via SuperBot. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Workshop on Self-Reconfigurable Robots, Systems & Applications*, San Diego, CA, November 2007.

6. Nadeesha Ranasinghe, Jacob Everist, and Wei-Min Shen. Modular Robot Climbers. InProc. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, Workshop on Self-Reconfigurable Robots, Systems & Applications*, San Diego, CA, November 2007.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to the Department of Defense, Executive Service Directorate (0704-0188). Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE (DD-MM-YYYY) 02-28-2009		2. REPORT TYPE Final Report		3. DATES COVERED (From - To) 02/2006-2/2009	
4. TITLE AND SUBTITLE SURPRISE: Surprise-Based Learning				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER FA9550-06-1-0336	
				5c. PROGRAM ELEMENT NUMBER	
				5d. PROJECT NUMBER	
6. AUTHOR(S) Dr. Wei-Min Shen				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Southern California Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90292				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research 875 North Randolph Street, Suite 325, Room 3112, Arlington, Va., 22203-1768				10. SPONSOR/MONITOR'S ACRONYM(S) AFOSR	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL DSK-VA-TR-2012-0086	
12. DISTRIBUTION/AVAILABILITY STATEMENT Unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Developed a learning algorithm that enable autonomous systems/robots to deal with and adapt to unexpected situations. This "Surprise-Based Learning" (SBL) technique engages an autonomous system in a life-long cyclic learning process consisting of "prediction, action, observation, analysis (of surprise) and adaptation". In particular, the robot always predicts the consequences of its actions, detects the surprises whenever there is a significant discrepancy between the prediction and the observed reality, analyzes the surprises for its causes and uses the critical knowledge extracted from analysis to adapt itself to the unexpected situations. SBL was successfully demonstrated on a physical modular robot which learned to navigate to desired goal-scenes with no prior knowledge about the environment, its sensors or the consequences of its actions. The scalability of SBL in the number of sensors and actions was evaluated to be reasonable for a physical robot, while adaptation to unexpected situations such as hardware failure and goal transfer was very successful.					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:		17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	
a. REPORT	b. ABSTRACT	c. THIS PAGE		19a. NAME OF RESPONSIBLE PERSON Dr. Wei-Min Shen	
				19b. TELEPHONE NUMBER (include area code)	

10

rd Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18
Adobe Professional 7.0

20120918070