ENVIRONMENTAL REINFORCEMENT LEARNING:
A Real-time Learning Architecture for Primitive Behavior Refinement

TaeHoon Anthony Choi, Eunbin Augustine Yim, and Keith L. Doty
Machine Intelligence Laboratory
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611
E-mail: tae@mil.ufl.edu, yim@mil.ufl.edu, and doty@mil.ufl.edu
URL: http://www.mil.ufl.edu/

ABSTRACT

This paper presents *Environmental Reinforcement Learning (ERL)*: a real-time learning architecture for refining primitive behaviors through repetitive execution of those behaviors in highly structured environments. Environment plays the key role in *ERL* not only by providing a suitable reinforcement to the agent, but also by forcing the agent to execute the behavior repetitively. Strength of *ERL* lies in the fact that complex factors governing primitive behaviors can be reduced or simply ignored by treating the agent's behavioral interactions with the environment as a "black box." In other words, analysis of different sources of variance in a behavior is not necessary to refine that behavior. The specially constrained environment "punishes" or "rewards" the agent for incorrect or correct execution of a behavior. Any reward or punishment for a behavior is relayed to the agent as *Environmental Reinforcement (ER)*, thereby causing the agent to adjust its behavioral parameters. An autonomous mobile agent, Mantaray, experimentally demonstrates *ERL* architecture on two primitive behaviors, (1) traversing in a straight line, and (2) turning 180 degrees.

1. INTRODUCTION

A current trend in autonomous agent learning deals with optimally sequencing certain combination of primitive behaviors which require low level motor skills to reach a goal. There have been many advances in this area [Lin, 1992; Parker, 1992; Ram and Santamaria, 1993; Ring, 1994]. However, the learning process of the primitive behaviors themselves should not be ignored or trivialized altogether. Even if a learning algorithm determines the proper behavior sequence to reach a particular goal, the crude accuracy of the primitive behaviors can substantially degrade its overall performance. Although the accuracy of certain behaviors can be refined through the use of more precise sensors and actuators, one pays the price of significantly higher financial cost for the agent.

In practice, characteristics of each agent are different; and consequently, the agent's physical performance varies greatly one from the other. Even if equivalent parts are used to assemble two seemingly identical agents, the actual performance of these two agents can vary greatly. In fact, possible sources of errors are almost limitless. For example, variance can be introduced in the radius of wheels, mounting of the motors, mounting of the wheels, motor performance, etc.

As shown in figure 1, adapted from [Yim and Choi, 1995], minute difference in wheel radius can result in significant accumulation of position error of the agent. One possible solution to this problem is to account for all the possible sources of variance. This course of action would be tedious, time consuming, and practically impossible. Furthermore, this process must be repeated for each new agent.

This paper describes a real-time architecture by which an agent can learn to correct the problems encountered in the accuracy or consistency of primitive behaviors. This architecture, called *Environmental Reinforcement Learning (ERL)*, is a learning process where the agent refines the primitive behavior through multiple iterations of the behaviors to be refined. The agent executes the behaviors in a highly structured environment from which the agent learns through *Environmental Reinforcement (ER)*. Section 2 describes the *ERL* architecture, providing an overview of *ERL* architecture, assumptions taken, environmental considerations, mutual refinement, and strategy for faster convergence. Section 3 presents experimental results and analysis of physically implemented *ERL* architecture. This section introduces the autonomous

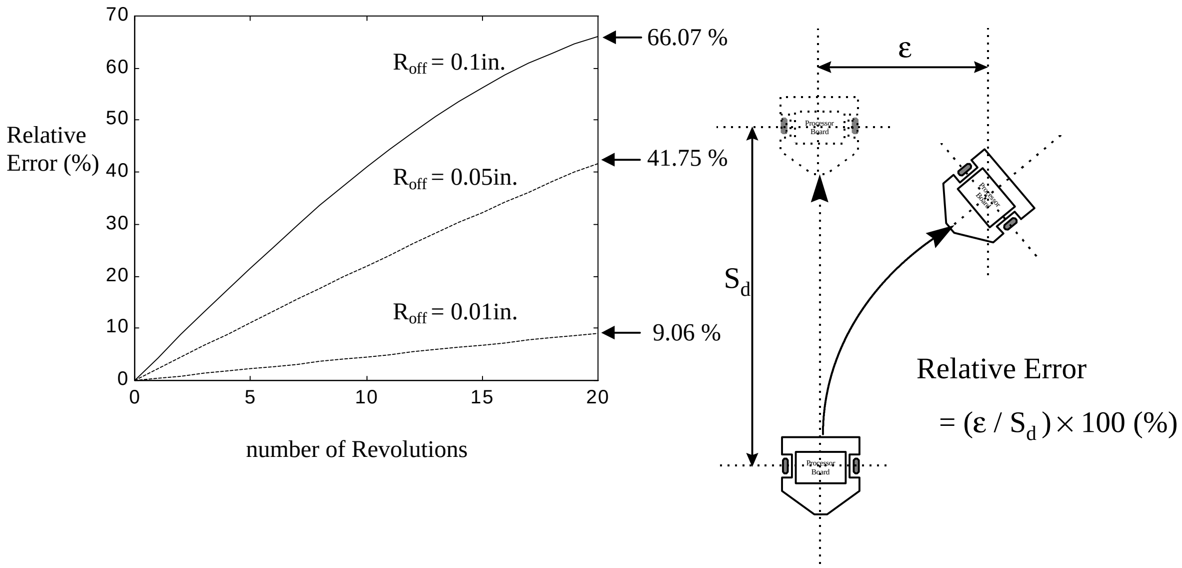


Figure 1: Relative Error vs. the Number of Revolutions, where R_{off} equals the difference in radii of the two wheels.

mobile agent, Mantaray [Choi, 1995], and two specific examples of the *ERL* process by which the primitive behaviors are refined (traverse a straight line and turn 180 degrees). Finally, section 4 offers concluding.

2. ENVIRONMENTAL REINFORCEMENT LEARNING

As shown in Figure 2, *Environmental Reinforcement Learning (ERL)* defines a learning architecture by which primitive behaviors, like low level motor skills (turning a certain number of degrees, traversing in a straight line, wall-following, etc.), are refined through repetitive execution of the same primitive behaviors in a highly structured environment. The environment forces the agent to perform a set of behaviors, where the execution of the behaviors results in *Environmental Reinforcement (ER)*. Due to the iterative nature of *ERL*, the environment should be designed with repetition in mind. This requirement can usually be satisfied by having an enclosed environment with a restrictive path for the agent. As long as the agent can return to its initial starting state, an iterative process can be guaranteed. Furthermore, environmental constraints should induce "punishment" in the agent for incorrect execution of a behavior. Any reward or punishment conveyed to the agent in this manner is *ER*. *ER* stems from sensory input triggered by environmental constraints, such as a proximity sensor indicating an approaching wall. The punishment induced by the environment causes the agent to adjust learning behavioral parameters for the behavior being punished. Consequently, the agent learns to perform the behavior "better" on the next trial.

2.1 Assumptions

Although *ERL* architecture can be adapted for higher level learning with or without increased complexity, it is best suited for primitive behaviors. *ERL* favors primitive behaviors due to the following assumptions. First, the variance of a behavior is constant (time-invariant) for a given state of the agent. Second, the variance of a behavior depends only on the practically (easily) observable states of the agent. Finally, a simple environment can be configured to force the execution of the desired behavior and *ER* can be given to the agent as feedback. These are quite restrictive assumptions, especially when dealing with complex behaviors. However, for primitive behavior these assumptions become quite reasonable. In fact, *ERL* can be applicable even for primitive behaviors which fail the first or second assumptions, by treating the agent's behavioral interactions with the environment as a "black box." In other words, analysis of different sources of variance in the behavior is not necessary to refine a behavior within acceptable error. Consequently, one may only be interested in

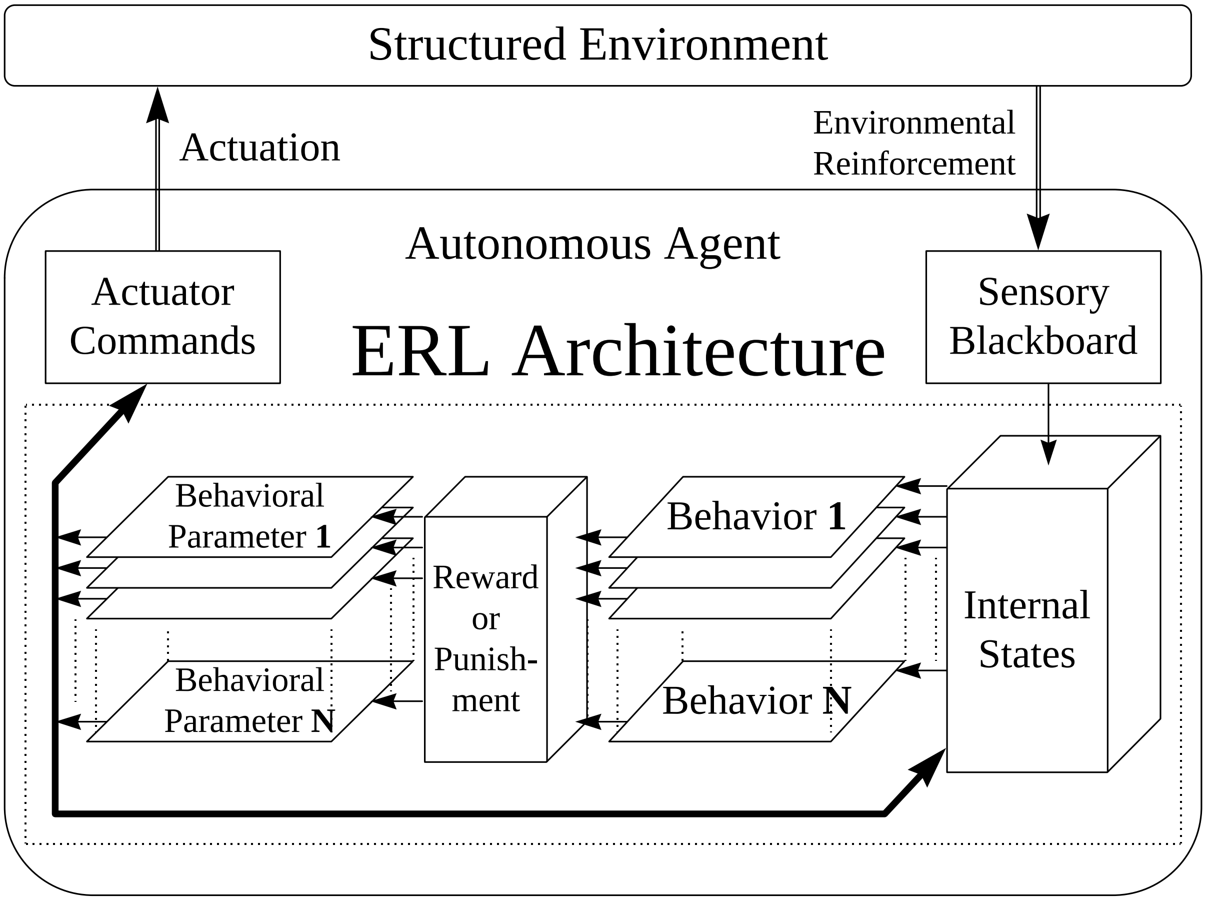


Figure 2: The *Environmental Reinforcement Learning (ERL)* Architecture

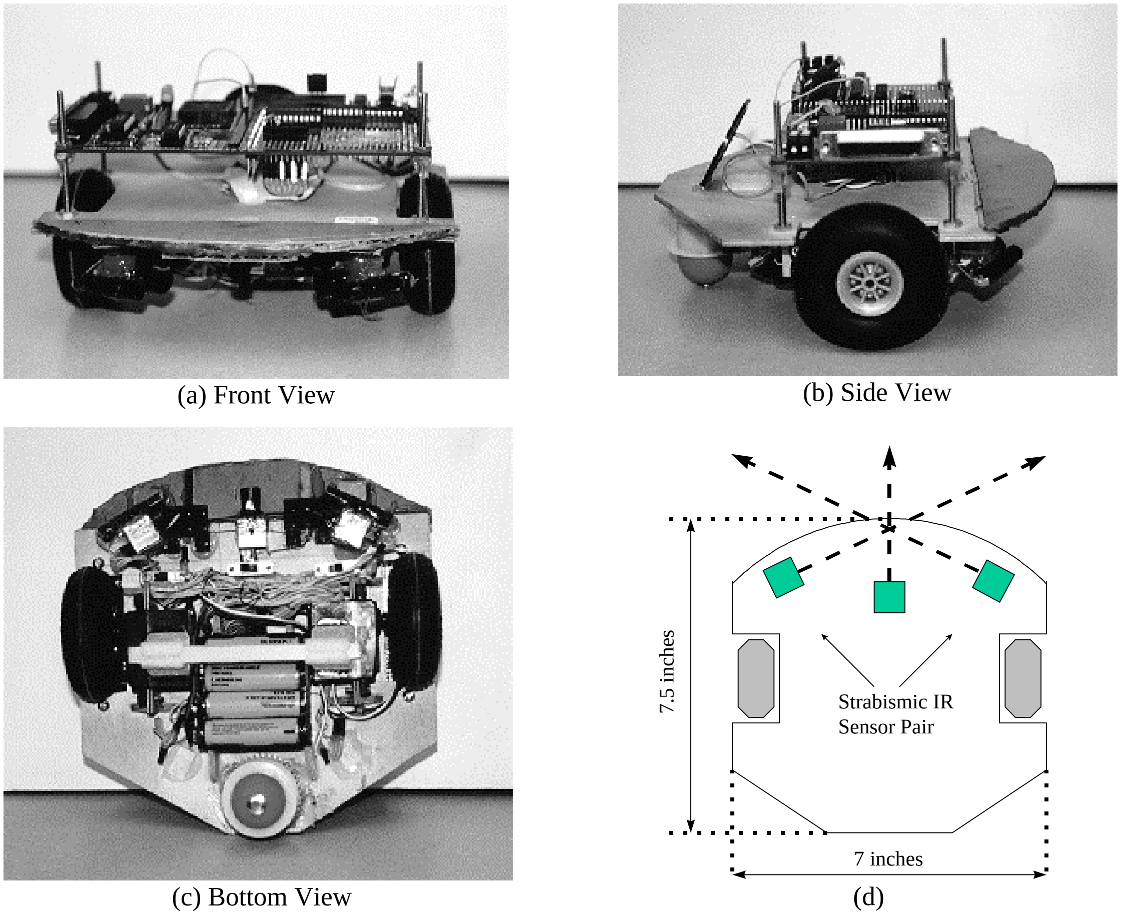
what actuation control is necessary for proper execution of a behavior without having to understand the sources of error in that behavior.

2.2 Environmental Reinforcement (ER) and Environmental Considerations

Environment plays a critical role, since the agent learns through its interaction with the environment. First, the environment must be configured in such a manner that it forces the agent to perform the desired behaviors. Second, for each forced behavior, the agent must receive *ER*, which the agent interprets as reward or punishment. *ER* allows the agent to evaluate its performance for that behavior and to adjust the learning behavioral parameters accordingly. One method of establishing *ER* is to specify restrictive paths for the behaviors to be learned. For perfect behavioral parameters and ideal hardware, the agent does not receive any negative *ER* from the environment. However, due to non-ideal variation in agent behavior, an agent receives *ER* for any deviation from the environmental constraints. Ultimately, the agent must return to the initial position for repeated runs under the same environmental conditions. This ability to find the initial position guarantees continual repetition of behaviors for learning purposes.

2.3 Mutual Refinement Process

An agent's ability to refine one primitive behavior is usually dependent on the refinement of other primitive behaviors. These dependent behaviors require mutual refinement process of several behaviors. By executing the interdependent behaviors in one series of an iteration, mutual refinement of several behaviors can be accomplished. However, Mutual Refinement Process


(a) Front View

(b) Side View

(c) Bottom View

(d)

Figure 3.1: Mantaray with Strabismic sensory array. (d) shows the Strabismic IR sensor pair.

assumes that the environmental constraints allow for the required combination of mutual behaviors. Consequently, the learning performance can be greatly enhanced by learning the interdependent behaviors concurrently.

2.4 Faster Convergence Through "Fine" and "Rough" Adjustment Trials

By starting with the "rough" adjustment trials, one can achieve faster convergence to the rough behavioral parameters. The desired characteristic of the "rough" adjustment trials is to minimize the time necessary to execute one iteration, correct by larger adjustment values, or both. Once the "rough" adjustment trials converge, "fine" adjustment trials begin. "Fine" adjustment's main goad is to fine tune the behavioral parameter accurately as possible.

3. EXPERIMENTAL RESULTS

The *Environmental Reinforcement Learning (ERL)* architecture was tested for the following two primitive behaviors: traversing in a straight line and turning a specified amount of degrees. Both behaviors were implemented concurrently in both the "rough" and "fine" adjustment trials on Mantaray.

3.1 Mantaray: Autonomous Mobile Agent

Mantaray, shown in Figure 3.1, is a seven inch by seven and a half inch, two wheeled mobile agent with a caster for support. It uses three IR sensors: one in the center and two orthogonal Strabismic sensor array [Choi, 1994]. Mantaray comes with all circuitry for the basic functions built on to the boards

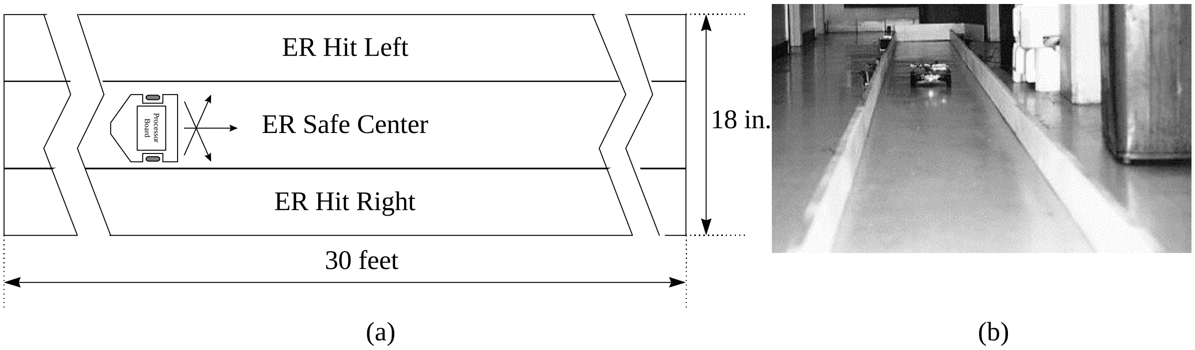


Figure 3.2: Environmental setup for the experiment. (a) *Environmental Reinforcement (ER)* for Straight Line Traversing. (b) Actual picture of the environment.

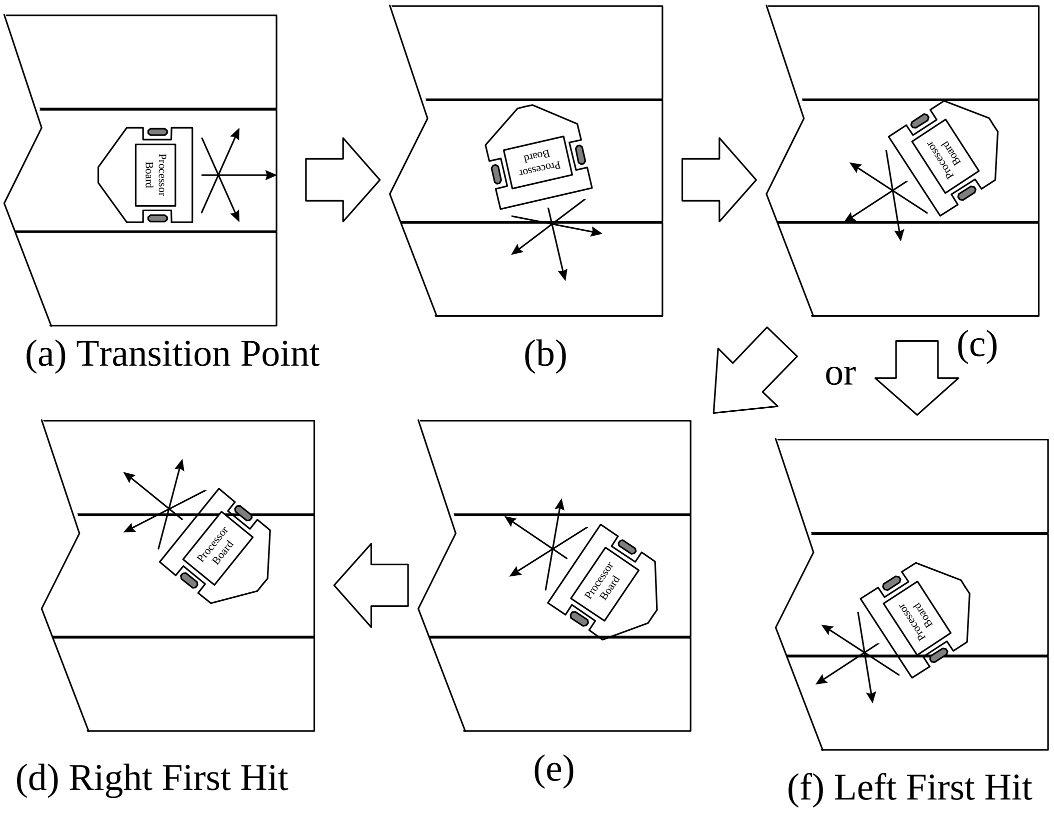
themselves. The 68HC11 8-Bit Microcontroller is clocked at 2 MHz with 32K SRAM memory. An 74HC138 Address Decoder supplies eight 8-Bit Memory Mapped I/O addresses. One of the memory mapped I/O is used to control the independent operation of the IR LED's. Using the A/D capability of the 68HC11 8-Bit Microcontroller, the Mantaray comes equipped with eight A/D ports (E0 - E7). These A/D ports can be used to collect sensory inputs from sensors with voltage level outputs. The UDN2993B Dual H-Bridge Motor Driver can drive two DC motors, usually used to drive the two wheels of the robot platform. It must be pointed out that Mantaray was not made for this experiment. Mantaray was constructed as a prototype for mass production (total of eight). However, several modifications were made to the original body and circuitry. First, the rolling caster was replaced by a stationary sliding caster. The rolling caster was replace due to the unpredictable error it introduced. Second, a voltage divider was added in parallel to the battery voltage with an analog tap; thus, allowing the agent to sense the battery state.

3.2 Learning to traverse a straight line

Due to numerous sources of errors as mentioned in the introduction, Mantaray had a natural tendency to arc left with the same speed (Pulse Width Modulation) applied to both motors. The tendency to stray from the straight path was controlled by the speed coefficient. The speed coefficient represents a percentage by which the faster motor is slowed to match other motor's speed. The environmental constraints applied to Mantaray was a straight corridor, shown in figure 3.2, with width of 18 inches and length of 30 feet for the "fine" adjustment trials and length of 10 feet for "rough" adjustment trials. The two ends of the corridor were blocked off. The Strabismic sensors were used to keep Mantaray in the center of the corridor, and since the corridor was straight, Mantaray was forced to traverse in a straight path. *ER* occurred when Mantaray veered off its center path and violated the *ER* zone to the left or to the right. As Mantaray traverses the full length of the corridor, it counts the number of *ER* from the left and number of *ER* from the right. If Mantaray has more *ER*'s on one side than the other with a difference of two or greater, the speed coefficient was adjusted to equalize the speed of both sides. The stop condition occurred when the front sensor detected the end of the corridor.

3.3 Learning to turn 180 degrees

The same straight corridor was also used to learn the turning coefficient. Turning coefficient represents the absolute time to turn 180 degrees. When Mantaray reaches the end of the corridor after a straight behavior, it ends up parallel to the corridor facing the enclosed end of the corridor, as in figure 3.3 (a). This forces a 180 degree turn by Mantaray in order to face the open end of the corridor. Initial turning reference is obtained by measuring the actual time it takes Mantaray to turn until the front sensor detects open space. Since motor speed is dependent not only on the duty cycle of the pulse width modulation, but also on the battery voltage applied to the to the motors, the learned turning coefficient is a function of the battery voltage measured during the turn. Once Mantaray obtains the turning reference, it turns for an amount of time specified by the turning coefficient, as shown in figure 3.3 (b), and stops in a position shown in figure 3.3 (c) or (e). Finally, Mantaray goes forward until an increase (decrease in distance to the wall) in one of the Strabismic sensor is detected, as shown in figure 3.3 (d) or (f). This detection then becomes the *ER* for this behavior.



(a) Transition Point (b) (c)

(d) Right First Hit (e) (f) Left First Hit

Figure 3.3: *Environmental Reinforcement (ER)* for Turn 180 degrees.

Depending on whether *ER* occurred from the left or the right, the turning coefficient is decreased or increase.

3.4 Speed and Turning Offset (adjustment values)

Initially, speed offsets are set to 1%; while, turning offsets are set to 50 milliseconds. The initial values of the offsets are not tweaked values. Except for the obvious extremes like zero, the offset can take on just about any value. If the offsets start as small numbers, it would take the agent longer to reach the correct value but would settle quickly. On the other hand, if the offsets start as large numbers, it would take the agent less time to reach the correct value but due to oscillation, would settle slowly. The values of each offsets do not change as long as the “trend” does not change (i.e., consistently increasing or decreasing). However, as soon as the “trend” changes, the offset is reduced to half of its previous value. This strategy allows for fast convergence in the beginning when the coefficients are far from their true values. On the other hand, as the coefficients near their true values, large offsets cause over adjustments which in turn causes “trend” changes. Consequently, adjustments become finer and finer as the offsets decrease. To keep the adjustments meaningful, lower limits of 0.01% and 1 millisecond were set for the speed and turning offsets, respectively.

3.5 Mutual Refinement

As the speed coefficient approaches its correct value, the initial condition for the turning behavior becomes more stable. Furthermore, being able to go straight ensures that the *ER* for the turning coefficient does not occur due to incorrect arcing of Mantaray. Similarly, as the turning coefficient approaches its optimal value, the initial condition for the turning behavior becomes more stable.

3.6 Experimental Results and Analysis

The state of the agent and the learning coefficients were recorded into the agent’s on board memory after completion of every trial. The following data were collected at the end of every trial:

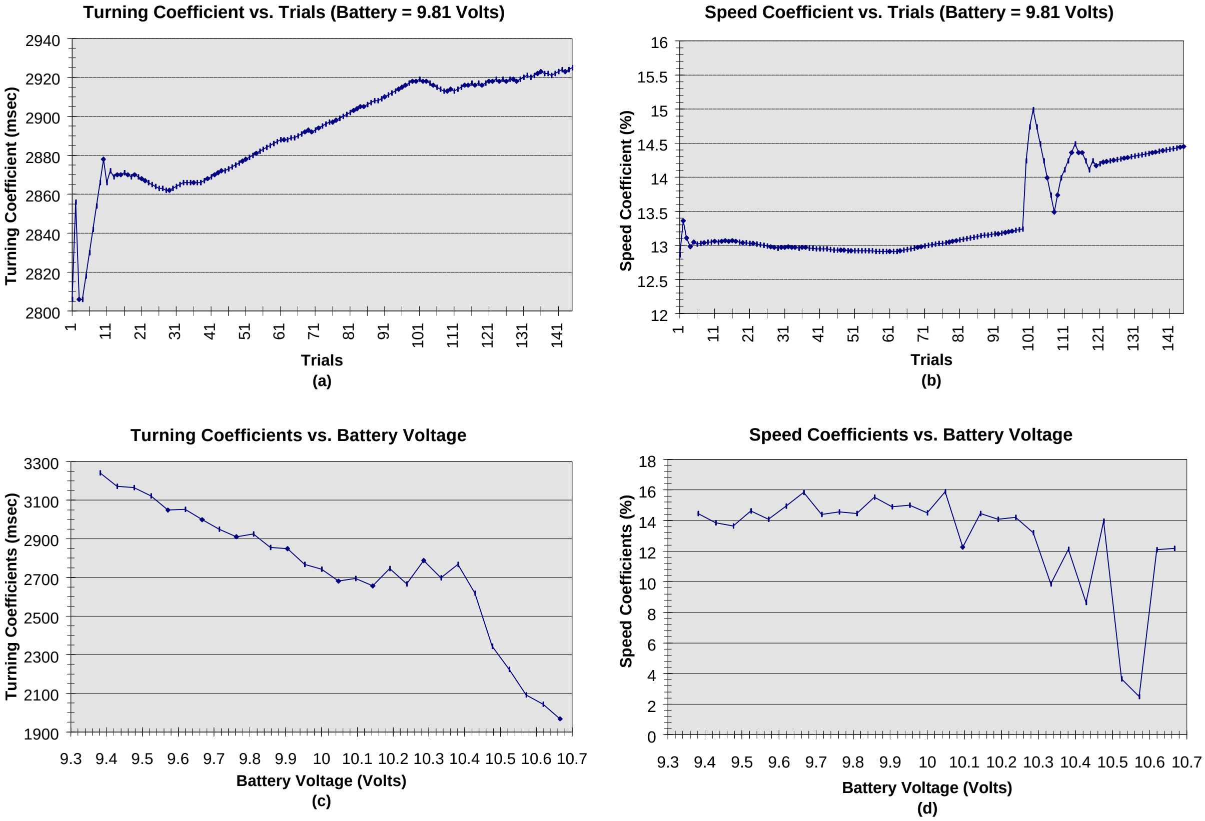


Figure 3.6: (a) and (b): Turning and Speed coefficients vs. Trials are plotted for a single battery voltage level of 9.81 Volts. Trials one to 99 represent "rough" trials. The rest represent "fine" trials (c) and (d) represent the accumulated Turning and Speed coefficients for each battery level.

Current system time (seconds).

Current battery voltage (analog sensor reading in the range of 225 to 197, which translate into 10.71V to 9.38V).

Current Speed coefficient (percent) used to decrease the faster motor to equalize the speed of both motors.

Current Turning coefficient (milliseconds) used to turn for a set amount of time for 180 degree turn.

Update **Speed coefficient** as a function of battery voltage.

Update **Turning coefficient** as a function of battery voltage.

Due to the Mutual Refinement dependencies, the "rough" trials were used for the first 99 trials. Consequently, the coefficients were updated three times as fast. However, "rough" trials using a short corridor causes problems for the Speed coefficients, because the agent does not have enough corridor length to execute the straight behavior, especially as the turning becomes more accurate. Due to the poor result of Speed Coefficient during the "rough" trials, the Speed offsets were reset to their initial values at the beginning of the "fine" trials. This transition also points out the Mutual Refinement process. As the Speed Coefficient over adjusts, the Turning Coefficient is improperly affected, as seen in the hump which occurred in the Turning Coefficient after the 99th trial. But as the Speed Coefficient settles down, so does the Turning Coefficient.

In figure 3.6 (c) and (d), inconsistent fluctuations in the coefficients above battery voltage of 10V is due to the small number of trial runs for these battery voltages. Eight NiCad batteries should produce 9.6V when fully charged, but the batteries actually charge up to 10.7V. However, the batteries can not hold this charge for very long; thus, resulting in small number of trial runs at the voltages higher than 10V. Keeping this in mind, we concentrate in the interval between 9.38V and 10.0V.

The performance of the Speed Coefficient was disappointing. It raised more questions than answers. Even when the motor's pulse width modulation was directly controlled through interrupts without any other over head by using assembly code and the Speed Coefficient was manually tweaked, the

Mantaray's straight path improved, but was still unpredictable. This meant that the performance of the motors varied as a function of some factor which was poorly accounted for. This variance was great enough that although *ERL* improved the behavior, consistency was not achieved.

On the other hand, good results were obtained for the Turning Coefficients. The 180 degree turns were executed with precision where the error was less than five degree. Linear regression analysis on the data points less than or equal to 10 Volts resulted in standard error of 19.56 milliseconds. This translates into standard error of 1.2 degrees for an 180 degree turn.

4. CONCLUSION

In this paper, two primitive behaviors were developed using the *ERL* architecture. This architecture proposes a novel idea, where, the environment forces an agent to alter its behavior in a desired manner. Through *ER* the agent learns to modify and refine its behavior. Furthermore, complexities were reduced by treating the agent's behavioral interactions with the environment as a "black box."

Refinement of two primitive behaviors, traversing in a straight line and turning 180 degrees, was attempted. The first behavior showed significant improvements when compared to the original performance but lacked consistency. The 180 degrees turning behavior, however, showed the flexibility and simplicity of the *ERL* architecture.

Possible future work in the area of *ERL* includes turning in subdivisions of 180 degrees (i.e., 90, 45, 30 degrees). Also, by adjusting the Speed Coefficient to keep the motor at constant absolute speed, the motor speed can be independent of changes in the battery voltage.

As primitive behaviors are refined through *ERL*, the problems of object identification and map building can be simplified. Wonder what an agent can do if it could measure angles and distances with negligible or small error? These two problems are definite candidates applications of the behaviors refined through *ERL*.

REFERENCES

[Choi, 1994] TaeHoon A. Choi. Earthcruiser. *Machine Intelligence Laboratory Technical Report* **MIL121294TAC**, University of Florida, 1994

[Choi, 1995] TaeHoon A. Choi. Mantaray. *Machine Intelligence Laboratory Technical Report* **MIL051595TAC**, University of Florida, 1995

[Lin, 1992] Long-Ji Lin. Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching. In *Machine Learning* **8**, pages 293-321, Kluwer Academic Publishers, 1992

[Parker, 1992] Lynne E. Parker. Adaptive Action Selection for Cooperative Agent Teams. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 442-450, MIT Press, 1992

[Ram and Santamaria, 1993] Ashwin Ram and Juan C. Santamaria. Multistrategy Learning in Reactive Control Systems for Autonomous Robotic Navigation. In *Informatica* **17 (4)**, pages 347-369, 1993.

[Ring, 1994] Mark B. Ring. Continual Learning in Reinforcement Environments. *Ph.D. Thesis*, University of Texas, 1994

[Yim and Choi, 1995] Eunbin A. Yim and TaeHoon A. Choi. Movement Analysis of a Dual Wheel-based Autonomous Mobile Agent. *Machine Intelligence Laboratory Technical Report* **MIL090395EAY**, University of Florida, 1995