# Reinforcement Learning in Biologically-Inspired Collective Robotics: A Rough Set Approach

**by**

**Christopher Henry**

**A Thesis**
**submitted to the Faculty of Graduate Studies,**
**in Partial Fulfilment of the Requirements for the degree of**

**Master of Science**
**in**
**Electrical and Computer Engineering**

© by Christopher Henry, January 2006

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba R3T 5V6 Canada

# Reinforcement Learning in Biologically-Inspired Collective Robotics: A Rough Set Approach

**by**
**Christopher Henry**

**A Thesis**
**submitted to the Faculty of Graduate Studies,**
**in Partial Fulfilment of the Requirements for the degree of**

**Master of Science**
**in**
**Electrical and Computer Engineering**

© by Christopher Henry, January 2006

Permission has been granted to the Library of the University of Manitoba to lend or sell copies of this thesis to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and University Microfilms to publish an abstract of this thesis.
The author reserves other publication rights, and neither the thesis nor extensive abstracts from it may be printed or otherwise reproduced without the author's permission.

**Abstract**

This thesis presents a rough set approach to reinforcement learning. This is made possible by considering behaviour patterns of learning agents in the context of approximation spaces. Rough set theory introduced by Zdzisław Pawlak in the early 1980s provides a ground for deriving pattern-based rewards within approximation spaces. Learning can be considered episodic. The framework provided by an approximation space makes it possible to derive pattern-based reference rewards at the end of each episode. Reference rewards provide a standard for reinforcement comparison as well as the actor-critic method of reinforcement learning. In addition, approximation spaces provide a basis for deriving episodic weights that provide a basis for a new form of off-policy Monte Carlo learning control method.

A number of conventional and pattern-based reinforcement learning methods are investigated in this thesis. In addition, this thesis introduces two learning environments used to compare the algorithms. The first is is a Monocular Vision System used to track a moving target. The second is an artificial ecosystem testbed that makes it possible to study swarm behaviour by collections of biologically-inspired bots (tiny robotic devices designed to crawl on the ground, up and down power towers, and along sky wires stretching between power towers). The simulated ecosystem has an ethological basis inspired by the work of Niko Tinbergen, who introduced in the 1960s methods of observing and explaining the behaviour of biological organisms that carry over into the study of the behaviour of interacting robotic devices that cooperate to survive and to carry out highly specialized tasks.

Agent behaviour during each episode is recorded in a decision table called an $ethogram$, which records features such as states, proximate causes, responses (actions), action preferences, rewards and decisions (actions chosen and actions rejected). At all times an agent follows a policy that maps perceived states of the environment to actions. The goal of the learning algorithms is to find an optimal policy in a non-stationary environment. The results of the learning experiments with seven forms of reinforcement learning are given. The contribution of this thesis is a comprehensive introduction to a pattern-based evaluation of behaviour during reinforcement learning using approximation spaces.

**Keywords**: Approximation space, ecosystem, ethology, reinforcement learning, rough sets, swarm, target tracking

# Acknowledgements

Many thanks the to following people for their role to in the completion of this thesis.

- My adviser, James Peters, for his counsel and advice, great discussions, valued feedback, and steadfast confidence. Thank you for being a great role model and helping me to create a thesis that I am proud of. A smile goes a long way.
- Dan Lockery and Maciej Borkowski for their helpful discussions and work on various parts of the physical ecosystem called Alice II that parallels the research with the ecosystem testbed used to obtain swarm behaviour samples reported in this thesis.
- Wes Mueller and Manitoba Hydro for their generous support.
- My parents and family for continuing to be part of my strong foundation and their constant support.
- Lastly, my fiancée Tanya Kowalski for her love and never ending encouragement.

This research has been supported by Natural Sciences and Engineering Research Council of Canada (NSERC) grant 185986 and grant T247 from Manitoba Hydro.

# Contents

# List of Tables

# List of Figures

# 1 Introduction

In reinforcement learning, the choice of an action is based on estimates of the value of a state and/or the value of an action in the current state using some form of an update rule (see, *e.g.*, [8, 17, 56, 69]). An agent learns the best action to take in each state by maximizing a reward signal obtained from the environment. Two different forms of reinforcement comparison and the actor-critic method as well as three forms of the off-policy Monte Carlo learning control method are investigated in this thesis, namely, the conventional approach and the approximation space approach to reinforcement learning in real-time by an agent. Furthermore, the two instances of an agent investigated in this thesis are a collection of cooperating bots that learn by evaluating their actions, and a single agent learning to track a moving target. First, incremental reinforcement comparison (IRC) with a very simple update rule is considered. Then a new form of IRC is considered where reference rewards derived from what is known as an approximation space. Briefly, an approximation space is a framework for measuring the closeness of clusters (called blocks) of equivalent objects to a set representing a standard. The basic idea for an approximation space comes from [37], and is amplified in [40]. The motivation for considering approximation spaces as an aid to reinforcement learning comes from the fact that it becomes possible to derive pattern-based rewards (see, *e.g.*, [44]).

The conventional actor-critic method is also considered in this thesis. A critic evaluates whether things have gotten better or worse than expected as a result of an action-selection in the previous state. A temporal difference (TD) error term $\delta$ is computed by the critic to evaluate an action previously selected. An estimated action preference in the current state is then determined using $\delta$. Agent actions are generated using the Gibbs softmax method [69]. In the study of swarm behaviour of multiagent systems such as systems of cooperating bots, it is helpful to consider ethological methods (see, *e.g.*, [70]), where each proximate cause (stimulus) usually has more than one possible response. Swarm actions with lower TD error tend to be favoured. A second form of actor-critic method is defined in the context of an approximation space (see, *e.g.*, [37, 38, 39, 40, 47, 48, 59, 60, 65]), and which is an extension of recent work with reinforcement comparison (see, *e.g.*, [40, 43, 44]). This form of actor-critic method utilizes what is known as a reference reward, which is pattern-based and action-specific. Each action has its own reference reward which is computed within an approximation space that makes it possible to measure the closeness of action-based blocks of equivalent behaviours to a standard.

A basic assumption in the study of real world reinforcement learning problems is that the state transition probabilities are non-stationary and a perfect model of the environment can not be obtained. This is particularly true during the investigation of the biologically-inspired, artificial ecosystems containing swarms of cooperating bots. Let $Pr(X = x)$

denote the probability that $X$ equals $x$. It is assumed that the return $R$ (cumulative discounted future rewards) for a sequence of actions is a discrete random variable, and the probability $Pr(R = r)$ is not known. In effect, if the episodic behaviour of a swarm yields a sequence of returns $R_1, \ldots, R_n$, the value of the expectation $E[R] = \sum_{j=1}^{n} x_j \Pr(R_j = x_j)$ is not known. Monte Carlo methods (see, *e.g.*, [29, 52, 53, 72, 73]) offer an approach to estimating the expected value of $R$. In general, a Monte Carlo method relies on the use of pseudo-random methods to construct estimates of unknown quantities such as the values of action or state functions. In a Monte Carlo method calculation, random numbers defined by real phenomena are generated, and then the resulting calculation is a direct simulation (*i.e.*, imitation) of the phenomena [29]. Agents in a non-stationary ecosystem agents learn from experience, which is reflected in sample sequences of states, actions and rewards that result from episodic interaction with an environment that is constantly changing. Learning in such an environment entails continual exploration. That is, agents learn during an episode by exploring actions that may not have not have been particularly promising (low rewards) in the past. Monte Carlo estimation is used in controlling behaviour. Only the off-policy learning control method and its rough coverage counterpart are considered in this thesis. The contribution of this thesis is a comprehensive introduction to a pattern-based evaluation of behaviour during reinforcement learning using approximation spaces.

This thesis is organized as follows. Basic ideas and notation from rough set theory is given in Sect. 2. The basic idea of an approximation space is presented in Sect. 3. Two examples of special forms of approximation spaces used in the reinforcement learning algorithms presented in this thesis are given in Sect. 3.1 and Sect. 3.2. Sect. 4 gives a brief overview of reinforcement learning. Sect. 5 relates the reinforcement learning paradigm to the Monte Carlo method. Half of the experiments given in this thesis are obtained from observations of swarm behaviour in an ethology based, artificial ecosystem. Ethology is briefly introduced in Sect. 6. Ethograms (tabular representations of swarm behaviour) are presented in Sect. 7 in the context of an ecosystem testbed designed to support the study of reinforcement learning by swarms (see Appendices A & B). The remaining experimental results for this thesis are obtained from a single agent which must learn to follow a moving target by controlling the position of a camera via a mini high powered servo. Consequently, the Monocular Vision System is described in Sect. 8 and the image processing techniques used to track the moving target are presented in Sect. 9. Next, a discussion of utilizing reinforcement learning in the Monocular Vision System is presented in Sect. 10. Conventional and approximation space-based incremental reinforcement learning, Actor-Critic, and off-policy learning methods are presented in sections 11, 12, and 13. A comparison of results obtained from the three different reinforcement learning methods by testing with either the ecosystem or the monocular vision system is given in Sect. 14.

## 2 Rough Sets

This section presents some fundamental concepts in rough set theory that provide a foundation for estimating the expected value of a state or state-action pair for actions by collections of cooperating agents. The rough set approach introduced by Zdzisław Pawlak [35, 36, 37] provides a ground for concluding to what degree a set of equivalent behaviours are covered by a set of behaviours representing a standard. The term "coverage" is used relative to the extent that a given set is contained in a standard set. An overview of rough set theory and applications is given in [20, 49]. For computational reasons, a syntactic representation of knowledge is provided by rough sets in the form of data tables (see, *e.g.*, Table 1). A data table (information system IS) is represented by a pair ($U$, $A$), where $U$ is a non-empty, finite set of elements and $A$ is a non-empty, finite set of attributes (features), where for every $a \in A$ there is a function $a : U \longrightarrow V_a$ in which $V_a$ is the value set of $a$ (*i.e.* $V_a = a(x)$ where $x \in U$). For each $B \subseteq A$, there is associated an equivalence relation $Ind_{IS}(B)$, called the relation of indiscernibility with respect to the attribute set $B$, such that

$$Ind_{IS}(B) = \left\{(x, x') \in U^2 | \forall a \in B, a(x) = a(x')\right\} \tag{1}$$

where $U^2$ is the Cartesian square of the set $U$. Let $U/Ind_{IS}(B)$ denote a partition of $U$ determined by $B$ (*i.e.*, $U/Ind_{IS}(B)$ denotes a family of subsets created from the relation $Ind_{IS}(B)$), and let ***B(x)*** denote a set of $B$-indiscernible elements containing $x$. ***B(x)*** is called a block, which is a member of the family of subsets created from the partition $U/Ind_{IS}(B)$. For example, Fig. 1 contains an information system $(U, A)$ where the elements of $U$ are represented by coloured circles, and the feature set $A$ contains only one attribute, namely, the element's colour. Furthermore, the indiscernibility relation, $Ind_{IS}(B)$ is used to partition the set $U$ into blocks of elements containing equivalent value sets (*i.e.* each block contains elements whose feature values are the same). Lastly, note that for this case the set $B = A$.

Next, for any $X \subseteq U$, the sample $X$ can be approximated from information contained in $B$ by constructing a B-lower and B-upper approximation denoted by $B_*X$ and $B^*X$, respectively, where

$$B_*X = \cup \{B(x) | B(x) \subseteq X\} \tag{2}$$

and

$$B^*X = \cup \{B(x) | B(x) \cap X \neq \emptyset\} \tag{3}$$

The B-lower approximation $B_*X$ is a collection of blocks of sample elements that can be classified with full certainty as members of $X$ using the knowledge represented by attributes in B. By contrast, the B-upper approximation $B^*X$ is a collection of blocks of sample elements representing both certain and possibly uncertain knowledge about $X$.

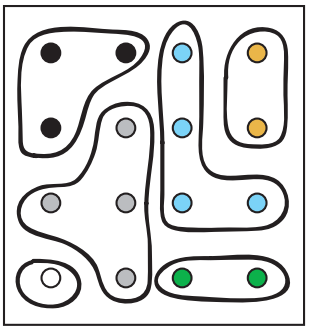


Figure 1: Blocks of $B$-indiscernible elements

| $x_i$ | $s$ | $PC$ | $a$ | $p(s,a)$ | $r$ | $d$ |
|---|---|---|---|---|---|---|
| $x0$ | 1 | 3 | 4 | 0.010 | 0.010 | 0 |
| $x1$ | 1 | 3 | 5 | 0.010 | 0.010 | 1 |
| $x2$ | 1 | 3 | 4 | 0.010 | 0.010 | 0 |
| $x3$ | 1 | 3 | 5 | 0.020 | 0.011 | 1 |
| $x4$ | 1 | 3 | 4 | 0.010 | 0.010 | 0 |
| $x5$ | 1 | 3 | 5 | 0.031 | 0.012 | 1 |
| $x6$ | 1 | 3 | 4 | 0.010 | 0.010 | 0 |
| $x7$ | 1 | 3 | 5 | 0.043 | 0.013 | 1 |
| $x8$ | 1 | 3 | 4 | 0.010 | 0.010 | 1 |
| $x9$ | 1 | 3 | 5 | 0.056 | 0.014 | 0 |

Table 1: Sample Information System

These concepts can be observed in Fig. 2 in which the lower and upper approximations of a sample $X$ are traced with thick contours. Finally, whenever $B_*X$ is a proper subset of $B^*X$, *i.e.*, $B_*X \subset B^*X$, the sample $X$ has been classified imperfectly, and is considered a rough set.

**Definition 1 Rough Set [35]** *Let $X \subseteq U$(universe) and $B \subseteq A$(feature set). Whenever $B_*X$ is a proper subset of $B^*X$, i.e., $B_*X \subset B^*X$, the sample $X$ has been classified imperfectly, and is considered a rough set.*

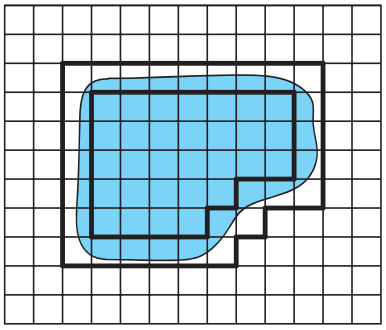


Figure 2: The Lower and Upper Approximations to a Sample $X$ [49]

# 3 Approximation Spaces

This section gives a brief introduction to approximation spaces. The basic model for an approximation space was introduced by Pawlak in 1981 [34], elaborated in [33, 36], generalized in [59, 65], and applied in a number of ways (see, *e.g.*, [9, 39, 40, 61]). An approximation space serves as a formal counterpart of perception or observation [33], and provides a framework for approximate reasoning about vague concepts.

A very detailed introduction to approximation spaces considered in the context of rough sets is presented in [49]. The classical definition of an approximation space given by Zdzisław Pawlak in [34, 36] is represented as a pair ($U$, $Ind$), where the indiscernibility relation $Ind$ is defined on a universe of objects $U$ (see, *e.g.*, [57]). As a result, any subset $X$ of $U$ has an approximate characterization in an approximation space. A generalized approximation space was introduced by Skowron and Stepaniuk in [59, 65]. Let $\mathcal{P}(U)$ denote the powerset of $U$. A *generalized approximation space* is a system $GAS$ = ( $U$, $N$, $\nu$ ) where

- $U$ is a non-empty set of objects, and $\mathcal{P}(U)$ is the powerset of $U$,
- $N : U \rightarrow \mathcal{P}(U)$ is a neighbourhood function,
- $\nu$ : $\mathcal{P}$(U) x $\mathcal{P}$(U) $\rightarrow [0, 1]$ is an overlap function.

A set $X \subseteq U$ is said to be in a GAS if, and only if $X$ is the union of some values of the neighbourhood function. In effect, the neighbourhood function $N$ defines for every object $x$ a set of similarly defined objects [58]. That is, $N$ defines a neighbourhood of every sample element $x$ belonging to the universe $U$ (see, *e.g.*, [47]). Generally, $N$ can be created by placing constraints on the value sets of attributes (see, *e.g.*, [31]) as in Eq. 4.

$$y \in N(x) \Leftrightarrow \max_{a}\{dist_a(a(x), a(y))\} \leq \epsilon. \tag{4}$$

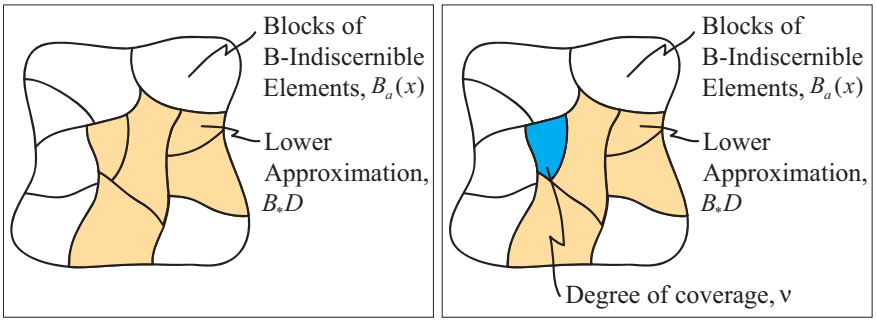
where $dist_a$ is a metric on the value set of $a$ and $\epsilon$ represents a threshold [31]. Specifically, any information system $IS = (U, A)$ defines for any $B \subseteq A$ a parameterized approximation space $AS_B = (U, N_B, \nu)$, where $N_B = B(x)$, a B-indiscernibility class in the partition of $U$ [58]. The rough inclusion function $\nu$ computes the degree of overlap between two subsets of U. The overlap function $\nu$ is commonly defined as standard rough inclusion (SRI) $\nu$ : $\mathcal{P}$(U) x $\mathcal{P}$(U) $\rightarrow [0, 1]$ as defined in Eq. 5.

$$\nu_{SRI}(X, Y) = \begin{cases} \frac{|X \cap Y|}{|X|}, & \text{if} \quad \text{X} \neq \emptyset, \\ 1 \quad , & \text{if} \quad \text{X} = \emptyset. \end{cases} \tag{5}$$

for any $X$, $Y \subseteq U$, where it is understood that the first term is the smaller of the two sets. The result is that $\nu_{SRI}(X, Y)$ represents the proportion of $X$ that is "included" in $Y$. However, for the purposes of this thesis it is desirable to be able to determine how well $Y$ "covers" $X$, where $Y$ represents a standard for evaluating sets of similar behaviours. This can be accomplished through the use of standard rough coverage (SRC) $\nu_{SRC}$ which can be defined as in Eq. 6 (also, see Fig. 3) where again, it is understood that the first term $X$ is the smaller of the two sets.

$$\nu_{SRC}\,(X, Y) = \begin{cases} \frac{|X \cap Y|}{|Y|}, & \text{if Y} \neq \emptyset, \\ 1 \quad , & \text{if Y} = \emptyset. \end{cases} \tag{6}$$

In other words, $\nu_{SRC}\,(X, Y)$ returns the degree that $Y$ covers $X$. In the case where




(a) Lower Approximation (b) Rough Coverage

Figure 3: Rough Coverage in a Lower Approximation Space

$X = Y$, then $\nu_{SRC}\,(X, Y)$ = 1. The minimum coverage value $\nu_{SRC}(X, Y)$ = 0 is obtained when $X \cap Y = \emptyset$ (*i.e.*, $X$ and $Y$ have no elements in common).

## 3.1 Example: Lower (Upper) Approximation Space

Let $AS_L$ = ( $U$, $N$, $\nu$ ) denote an approximation space defined in the context of the decision table $(U, A, d)$,

- $U$ is a non-empty set of objects. Assume $D \subset U$.
- $N : U \rightarrow \mathcal{P}(U)$, where $N(x) = B(x)$ for $B \subseteq A$.
- $\nu$ : $\mathcal{P}$(U) x $\mathcal{P}$(U) $\rightarrow [0, 1]$.

where $U$ is a non-empty set of behaviours, $A$ is a set of behaviour features, and $d$ is a distinguished attribute representing a decision. Let $D = \{x \in U : d(x) = 1\}$, where $d(x) = 1$ specifies that behaviour $x$ has been accepted. Recall that the lower approximation of a set $D$ is a collection of sample elements that can be classified with full certainty as members of $D$ using the knowledge represented by attributes in $B$ [37]. Let $v_B(B_{ac}(x), B_*D)$ denote *lower rough coverage* defined in Eq. 7

$$\nu_B\left(B_{ac}(x), B_*D\right) = \frac{|B_{ac}(x) \cap B_*D|}{|B_*D|} \tag{7}$$

where $\nu = \nu_B$ and $\nu_B(X, Y) = \nu_{SRC}(X, B_*Y)$ for $X, Y \subseteq U$. The value $\nu_B(X, Y)$ determines the extent with which $B_*Y$ covers $X$. $AS_L$ is called a lower approximation space. Similarly, an upper approximation space $AS_U$ can be defined by specializing the neighbourhood function $N$ and rough coverage $\nu_B$ in terms of the upper approximation of a set. Note, it is now important to differentiate between two different notations used throughout this thesis. The value of $ac$ used with respect to rough sets and approximation spaces is different from the value of $a$ mentioned in regard to the state value functions mentioned in Sect. 4. The term $ac$ represents the values obtained from discretizing the values of $a$ recorded during the episode (*i.e.*, the value $ac$ represents the interval to which $a$ belongs). The method of discretization is mentioned in Sect. 11.1.

## 3.2 Example: Approximation Space for a Swarmbot

Let $AS_L = (U_{beh}, N_B, \nu_B)$ denote an approximation space defined in the context of a decision system $IS = (U_{beh}, A, d)$. Assume that $N_B : U_{beh} \rightarrow \mathcal{P}(U_{beh})$ is used to compute $B_*D$ as in Sect. 3.1 where $N(x) = B(x)$ for $B \subseteq A$. Further, let $B_*D$ represent a standard for swarmbot behaviours, and let $B_{ac}(x)$ be a block in the partition of $U_{beh}$ containing $x$ relative to action $ac$ (*i.e.*, $B_{ac}(x)$ contains behaviours for a particular action $ac$ that are equivalent to $x$). The block $B_{ac}(x)$ is defined in Eq. 8.

$$B_{ac}(x) = \{y \in U_{beh} : xIND(B)y\} \tag{8}$$

This relation can be used to measure the extent that $B_*D$ covers $B_{ac}(x)$ as in Eq. 9.

$$\nu_B(B_{ac}(x), B_*D) = \begin{cases} \frac{|B_{ac}(x) \cap B_*D|}{|B_*D|}, & \text{if } B_*D \neq \emptyset, \\ 1 \quad , & \text{if } B_*D = \emptyset. \end{cases} \tag{9}$$

What follows is a simple example of how to set up a lower approximation space relative to a decision system. The calculations are performed on the feature values shown in Table 1.

$B = \{s_i, PC_i, a_i, p(s,a)_i, r_i\}$
$D = \{x \in U : d(x) = 1\} = \{x1, x3, x5, x7, x8\}$
$B_{ac}(x) = \{y \in U_{beh} : xIND(B \cup \{a\})y\}$, where
$B_{ac=4}(x0) = \{x0, x2, x4, x6, x8\}$
$B_{ac=5}(x1) = \{x1\}$
$B_{ac=5}(x3) = \{x3\}$
$B_{ac=5}(x5) = \{x5\}$
$B_{ac=5}(x7) = \{x7\}$
$B_{ac=5}(x9) = \{x9\}$
$B_*D = \cup\{B_{ac}(x) | B_{ac}(x) \subseteq D\} = \{x1, x3, x5, x7\}$,
$\nu_B(B_{ac=4}(x0), B_*D) = 0$
$\nu_B(B_{ac=5}(x1), B_*D) = 0.25$
$\nu_B(B_{ac=5}(x3), B_*D) = 0.25$
$\nu_B(B_{ac=5}(x5), B_*D) = 0.25$
$\nu_B(B_{ac=5}(x7), B_*D) = 0.25$
$\nu_B(B_{ac=5}(x9), B_*D) = 0$

$B_*D$ represents certain knowledge about the behaviours in $D$. For this reason, $B_*D$ provides a useful behaviour standard or behaviour norm in gaining knowledge about the proximity of behaviours to what is considered normal. The term $normal$ applied to a set of behaviours denotes forms of behaviour that have been accepted. The introduction of some form of behaviour standard makes it possible to measure the extent that the standard covers blocks of B-similar action-specific behaviours. The framework provided by an approximation space makes it possible to derive pattern-based rewards, which are used by swarms that learn to choose actions in response to perceived states of its environment (see, *e.g.*, Fig. 4).

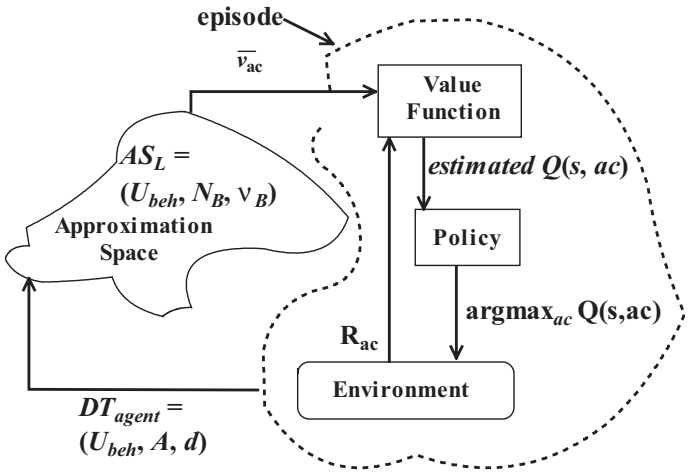



Figure 4: Episode Framework

# 4 Reinforcement Learning

There has been extensive research in reinforcement learning (see, *e.g.*, [1, 2, 5, 6, 7, 8, 14, 15, 16, 17, 18, 19, 22, 24, 26, 28, 27, 54, 55, 56, 62, 63, 64, 66, 67, 68, 69, 71, 74, 75, 77, 78]), and more recently in the context of rough sets and approximation spaces (see, *e.g.*, [40, 43, 44, 45]). Reinforcement learning itself is considered a problem formulation, not a solution technique [28]. The basic problem that provides a setting for reinforcement learning is formulated by [54]: a system is required to interact with its environment to achieve a particular task or goal, and based on the feedback about the current state of the environment, what action should the system perform next? Reinforcement learning itself is the act of learning the correct action to take in a specific situation based on feedback obtained from the environment [69]. In the context of this thesis, the feedback is a numerical reward assigned to actions taken by an agent. Specifically, reinforcement learning can be divided into *off-line* and *on-line* learning. Off-line learning is similar to the idea of a student learning by instruction from a teacher. In effect, the agent is taught what it needs to know before venturing into the environment in which it is to operate. In contrast, on-line learning resembles an infant learning to walk. Learning occurs in real-time in which the agent is exploring its environment and constantly adding to its experience in order to make better decisions in the future. Learning techniques are typically applied to stationary or non-stationary models of the environment. In stationary models all the state transition probabilities are fixed, whereas, in non-stationary models they change over time. This thesis investigates on-line non-stationary models with off-line learning and stationary models being outside the scope of this thesis. Lastly, a central theme to all

of the algorithms discussed in this thesis is the idea of an update rule. The update rule is extensively discussed in [69], as well as, [5, 15, 66, 78]. At its heart the update rule is a form of incremental average used by the agent to record experience gained from the environment. Formally, the update takes the form given in Eq. 10 (see *e.g.*, [69]).

$$\textit{New Estimate} \leftarrow \textit{Old Estimate} + \textit{Step Size}[\textit{Target} - \textit{Old Estimate}] \tag{10}$$

where *Step Size* is used to control the rate of learning. The terms $agent$, $state$, $action$, $reward$, $policy$, and *value of a state* are fundamental in RL (see, *e.g.*, [51, 54, 69]). An $agent$ is some form of a system that that has sensors that enable it to perceive (*i.e.*, sense and evaluate its environment), actuators, and the ability to perform actions that modify its environment. An *intelligent agent* is one that recognizes patterns and learns over time to choose beneficial actions. The sensors of an agent are used to describe the state of the environment. A $state\ s \in S$ is a unique interpretation of the environment from which an action $a \in A(s)$ is selected. An $action\ a$ by an agent causes a change in the environment. Each action is assigned a $reward\ r$ with a numerical value that represents the desirability of the action performed. Whenever an agent receives a reward for an action performed, it is viewed as a reinforcement signal [51]. Similarly, goals are achieved by the agent through maximizing the rewards it receives. The most desirable action to be performed by an agent is the action that is most likely to give the highest reward. Finding this action is called the *credit assignment* problem [54].

**Definition 2 Agent [69]** *A system that has sensors that enable it to perceive, actuators, and the ability to perform actions that modify its environment.*
**Definition 3 Reward [69]** *A value assigned to a state or action which represents its desirability.*
**Definition 4 Reinforcement [69]** *A stimulus which increases/decreases the likelihood of an action being selected in the future.*

Lastly this thesis compares three conventional RL algorithms (see *e.g.*, [8, 51, 69]) to similar versions incorporating approximation spaces. The first method *Incremental Reinforcement Comparison*, directly implements Eq. 10 to create a standard which is used as the basis for comparison. The second method is *Actor-Critic*, which is an extension of reinforcement comparison in that an update rule is still used. However, this method uses an actor to make action selections, and a critic evaluates the actor's performance and updates the policy (see, *e.g.*, [8, 51, 69]). The last method discussed in this thesis is the *Monte Carlo off-policy*, where off-policy means that the behaviour policy is separate from the estimation policy (see, *e.g.*, [8, 50, 69]). Whereas in on-policy algorithms the behaviour and estimation policies are one and the same. Furthermore, Monte Carlo algorithms are episodic in which all the learning occurs at the end of an episode.

# 5 Reinforcement Learning and the Monte Carlo Method

Let $S$, $s$, $A(s)$, $\pi$ denote set of possible states, particular state, and set of possible actions for a state $s$, and policy used in selecting an action to perform, respectively. Moreover, each $a \in A(s)$ is associated with a state $s$ (see, *e.g.*, [51]). In general, a policy can be characterized as a decision-making rule (see, e.g., [69]). Two types of policies are commonly used in reinforcement learning (see, e.g., [4, 69]. A *deterministic policy* is a mapping $\pi : S \rightarrow A$. By contrast, a *stochastic policy* is a mapping $\pi : S \times A \rightarrow [0, 1]$, where $\pi(s, a)$ equals the probability that action $a$ will be selected in state $s$. The state-value function $V^{\pi}(s)$ denotes the value of a state $s$ obtained by following policy $\pi$. The action-value function $Q^{\pi}(s, a)$ denotes the expected discounted future reward starting in state $s$, selecting action $a$, and continuing to follow policy $\pi$ [28, 50]. The basic idea in reinforcement learning is to find a good policy, where a good policy maximizes the value functions. So, in a sense, the selection of a policy defines a behaviour. Throughout this thesis $V(s)$ and $Q(s, a)$ will be used to denote estimates of the value functions. Let $r_i \in \Re$ denote a numerical reward resulting from action $a_i$ in state $s_i$. A behaviour is defined by a finite state sequence that occurs during an episode that ends at time $T$, and is represented by Eq. 11.

$$s_0, a_0, r_1, s_1, a_1, r_2, \ \ldots, s_i, a_i, r_{i+1}, \ \ldots, s_{T-1}, a_{T-1}, r_T, s_T \tag{11}$$

The return $R_m$ (*i.e.*, cumulative future discounted rewards) on a sequence of actions is defined by Eq.12.

$$R_m = r_1 + \gamma r_2 + \gamma^2 r_3 + ... + \gamma^{T-1} r_T = \sum_{m=1}^{T} \gamma^{m-1} r_m. \tag{12}$$

where $\gamma \in [0, 1]$ is called a discount rate, and $R_m$ is a discrete random variable. The basic idea is to choose actions during an episode that ends in a terminal state at time $T$ so that the expected discounted return $E^{\pi}[R_m]$ following policy $\pi$ improves. To do this, it is necessary to estimate the expected value of $R_k$.

**Definition 5 Stochastic Policy** *Let $S$ be a set of states, and $A$ a set of possible actions. Then, a stochastic policy is a mapping $\pi : S \times A \rightarrow [0, 1]$.*

**Definition 6 Return [69]** *A function of the reward sequence received in a specific state as in Eq. 12.*

## 5.1 Expectation and Monte Carlo Method

The study of the expected value of a random variable was first introduced in 1657 by C. Huygens in estimating the results of games of chance [13]. Assume that $X$ is a discrete random variable with possible values $x_1, \ldots, x_n$. Let $Pr(X = x)$ denote the probability that $X$ has the value $x$. The expected value of X (denoted $E[X]$) is identified with the mean of the distribution containing values of $X$, and is computed using Eq. 13.

$$E[X] = \sum_{k=1}^{n} x_k Pr(X = x_k). \tag{13}$$

Consider the generic problem of estimating the value of an action $Q^\pi(s, a)$, starting in state $s$ with action $a$ following policy $\pi$. Assume that there is a probability density function (pdf) $f$ such that $f(x) = Pr(R = x)$ can be computed for each value of $R$. Assume that $R$ is a continuous random variable with pdf $f$. Then the expected value of $Q^\pi(s, a)$ is given in Eq. 14.

$$Q^\pi(s, a) = E[R_m | s_t = s, a_t = a] = \int_{R_m} x f(x) dx, \tag{14}$$

where return $R_m$ is defined in Eq. 12. However, in a non-stationary environment, $R$ is a discrete random variable, and the probability $Pr(R_m = x)$ is not known. Hence it is helpful to use Monte Carlo methods to estimate the value of $R_m$. In its simplest form, the expected value $R_m$ is estimated using an empirical average, to obtain an approximate value of $Q^\pi(s, a)$ as in Eq. 15.

$$Q_m^\pi(s, a) \approx \frac{1}{m} \sum_{k=1}^{m} R_k, \tag{15}$$

where the average in Eq. 15 converges to $E\left[R_m | s_t = s, a_t = a\right]$ as $m \to \infty$ by the Strong Law of Large Numbers [52]. That is, $\frac{1}{m} \sum_{k=1}^{m} R_k$ approximates the integral $\int_{R_m} x f(x) dx$. The sample mean in Eq. 15 is called an unbiased estimator of $Q_m^\pi(s, a)$ (see, *e.g.*, [52, 53]). The Monte Carlo method was introduced by Ulam [72, 73].

## 5.2 Update Rule

As was mentioned in Sect. 4 a central theme in the reinforcement learning algorithms discussed in this thesis is that of an update rule. The motivation in working with update rules in learning algorithms, is the resulting computational simplicity as well as the more intuitive representation of the incremental character of the learning process. Using the Monte Carlo method, $Q_n^\pi(s, a)$ can be estimated using a weighted sum. Let $w_i$ denote an

importance sampling weight on $R_i$, and obtain an approximate value of $Q_n^\pi(s,a)$ using Eq. 16.

$$Q_n^\pi(s,a) \approx \frac{\sum_{i=1}^n w_i R_i}{\sum_{i=1}^n w_i}. \tag{16}$$

**Property 1** *A weighted average value function can be rewritten as an incremental update rule.*

Proof. *Consider, first, the sum of the weights as in Eq. 17.*

$$\begin{aligned} W_n &= \textstyle\sum_{i=1}^n w_i, \\ W_{n+1} &= W_n + w_{n+1} = \textstyle\sum_{i=1}^n w_i + w_{n+1}. \end{aligned} \tag{17}$$

*Then derive an incremental update rule for Eq. 16 as shown in Eq. 18 using Eq. 17.*

$$\begin{aligned} & Q_{n+1}^\pi(s,a) = \tfrac{1}{W_{n+1}} \textstyle\sum_{i=1}^{n+1} w_i R_i \\ &= \tfrac{1}{W_n + w_{n+1}} \textstyle\sum_{i=1}^{n+1} w_i R_i \\ &= \tfrac{1}{W_{n+1}} \left(w_1 R_1 + ... + w_n R_n + w_{n+1} R_{n+1}\right) \\ &= \tfrac{1}{W_{n+1}} \left(\textstyle\sum_{i=1}^{n} w_i R_i + w_{n+1} R_{n+1}\right) \\ &= \tfrac{1}{W_{n+1}} \left(w_{n+1} R_{n+1} + \textstyle\sum_{i=1}^{n} w_i R_i\right) \\ &= \tfrac{1}{W_{n+1}} \left(w_{n+1} R_{n+1} + W_n \tfrac{\sum_{i=1}^n w_i R_i}{W_n}\right) \\ &= \tfrac{1}{W_{n+1}} \left(w_{n+1} R_{n+1} + W_n Q_n^\pi(s,a)\right) \\ &= \tfrac{1}{W_{n+1}} \left(w_{n+1} R_{n+1} + W_n Q_n^\pi(s,a) + w_{n+1} Q_n^\pi(s,a) - w_{n+1} Q_n^\pi(s,a)\right) \\ &= Q_n^\pi(s,a) + \tfrac{w_{n+1}}{W_{n+1}} \left(R_{n+1} - Q_n^\pi(s,a)\right). \end{aligned} \tag{18}$$

■

The incremental update rule in Prop. 1 provides a foundation for the value functions used in the two off-policy learning algorithms introduced in this thesis.

# 6 Intelligent System Behaviour: An Ethological Perspective

A number of features of the behaviour of an agent in an intelligent system can be discovered with ethological methods. *Ethology* is the study of the behaviour and interactions of animals (see, *e.g.*, [70]). The study of biological behaviour provides a rich source for identifying features useful in modelling and designing intelligent systems, as well as identifying how behaviour contributes to survival. It has been observed that animal behaviour has patterns with a biological purpose and that these behavioural patterns have evolved. Similarly, patterns of behaviour can be observed in various forms of intelligent systems that respond to external stimuli and evolve. In the search for features of intelligent system behaviour, one might ask *Why does a system behave the way it does?* Tinbergen's four whys are helpful in answering this question by identifying important features of intelligent system behaviour which promote survival. Namely, proximate cause (stimulus), response together with the survival value of a response to a stimulus, evolution, and behaviour ontogeny (origin and development of a behaviour) [70]. Only proximate cause and action taken in response are considered in this thesis. Tinbergen's survival value of a behaviour for an animal correlates with reward that results from an action made in response to a proximate cause in an intelligent system. The assumption made here is that action preference is influenced by a reference or standard reward and survival is defined in terms of accomplishing goals. The focus in this thesis is behaviour on the swarm intelligence level (see, *e.g.*, [3]).

## 6.1 Swarm Behaviour: Proximate Causes and Responses

In the study of proximate causation the focus is on mechanisms that underly an observed behaviour, namely, contexts in which a behaviour occurs, its external (exogenous) and internal (endogenous) stimuli [21]. Specifically, it is the study of the preceding events which may induce a particular behaviour [70]. Behaviour itself is viewed as the response of an organism to a PC originating in the environment of an agent in an artificial ecosystem. In this context, a PC can either be exogenous (*e.g.*, lightning, high electromagnetic field, a moving target) or endogenous (*e.g.*, low swarm energy, mechanical failure). A PC and its response by a swarm is represented by a (state, action) pair defined in the context of reinforcement learning, where the reward for an action influences (proximally causes) the modification of the behaviour of an agent in an artificial ecosystem. A state in the ecosystems considered in this study is a representation of a swarm's perception of its environment. The notion of a PC is used in defining ecosystem states. In effect, a swarm uses an identified PC to determine its current state. Each state defines a set of available

actions used by a swarm to make an action-selection (see, *e.g.*, Tables 4, 5, 6, and 7).

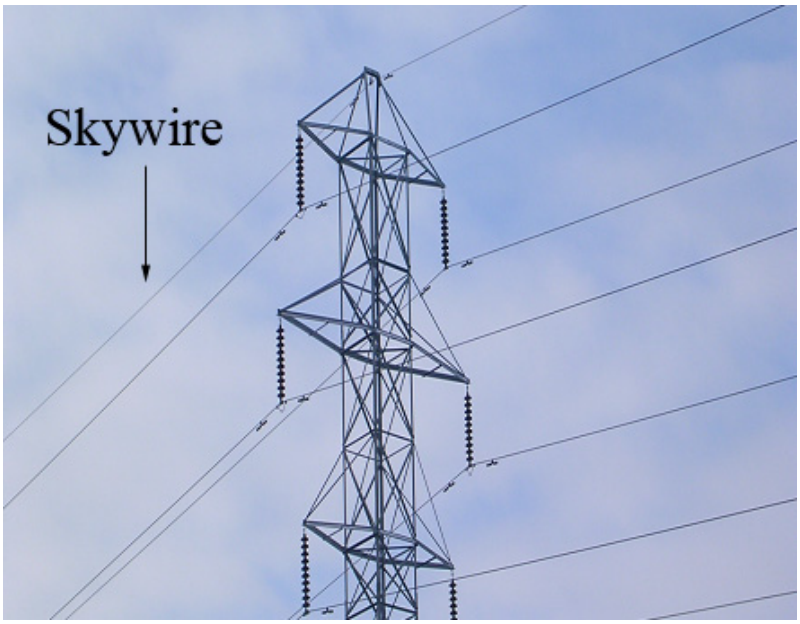
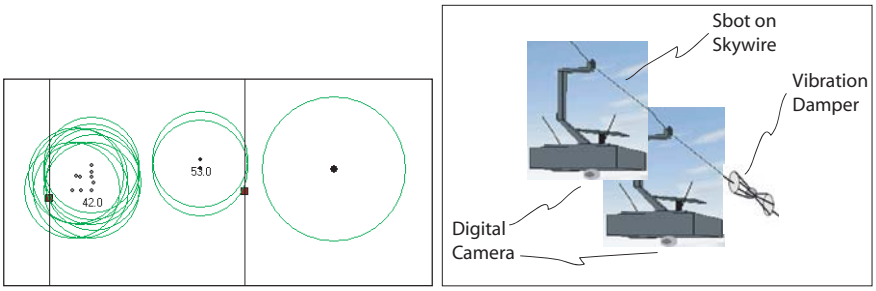



Figure 5: Hydro Tower Skywire (electrical ground line)

# 7 Ethograms Reflecting Swarmbot Behaviour

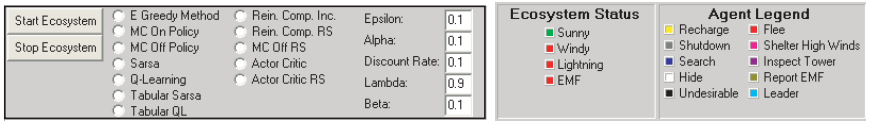
This section introduces a testbed modelled as an artificial ecosystem containing swarms of cooperating bots (sbots) in an environment containing sequences of hydro-electric towers which require inspection. The bots crawl along sky wires (electrical ground lines, see Fig. 5) strung between the apex of towers. Similarly, the bots can crawl up and down towers. These bots cooperate to inspect power system equipment (*e.g.*, insulators and towers). Two or more cooperating bots form a swarmbot. Bots are dependent on sunlight to recharge their solar cells. The ecosystem also models many bot-threatening hazards such as high winds and lightning. Fig. 6(a) is a screen shot of a testbed that automates the production of ethograms to provide a record of observed swarmbot (sbot) behaviour patterns (See Appendix A for a description of symbols used in Fig 6(a)). Fig. 7(a) shows the different learning algorithms the user can select, as well as, the different parameters that can be adjusted before starting the testbed. Similarly, Fig 7(b) shows the environmental feedback that the testbed provides to the user, as well as, a legend of swarm behaviour representing the different actions available to the bots in the ecosystem. For a detailed description of the testbed behaviour see Appendix A & B.

Episodes (random in duration in the interval [2s, 4s] with 200 ms increments), have




(a) Sbot testbed (b) Cooperating bots

Figure 6: Sample Sbot Behaviour




(a) Ecosystem Methods, parameters (b) Ecosystem Status, Agent Legend

Figure 7: Ecosystem Panels

been introduced in the ecosystem to break up learning into intervals, and make it possible for a swarm to "reflect" on what it has learned. Swarm behaviour during each episode is recorded in an decision table called an $ethogram$, which records swarm states, proximate causes, responses (actions), action preferences, state-action value estimates, rewards and decisions (actions chosen and actions rejected). The focus of the ecosystem is on swarm activity and swarm learning. At all times a swarm follows a policy that maps perceived states of the environment to actions. The goal of the learning algorithms is to find an optimal policy in a non-stationary environment.

Cooperation between bots is one of the hallmarks of swarmbot behaviour (see, *e.g.*, [3, 11]), and in multiagent systems containing independent agents that exhibit some degree of coordination (see, *e.g.*, [39]). Many of the details concerning the sbot in Fig. 6(b)[1] have been already been reported (see, *e.g.*, [42]). Fig. 6(b) provides a conceptual view of an aerial sbot called Alice II, which has a binocular vision system whenever two individual bots are coupled together. Lastly, this sample snapshot of the individual and collective behaviour of inspect bots in a line-crawling swarmbot testbed is part of a Manitoba Hydro Line-Crawling Robot research project.

[1] Patent pending.

| Vision System Component | Hardware Implementation |
|---|---|
| Camera | Logitech QuickCam Express |
| Servo Controller | UltraSound Module 5-120 |
| Servo | Hitec HS-300 Servo |

Table 2: Vision System Hardware

# 8 Monocular Vision Experiments

The monocular vision system provides an additional platform to compare the reinforcement learning algorithms. The design is a simplified conceptualization of the learning environment developed by Gaskett [8] and consists of a digital camera that is used to track a moving target. The system complements results obtained from the testbed by providing a real environment that is free of any unintentional biases that may have been included in the software simulation. Furthermore, the learning problem is smaller in scope than the ecosystem in that state and reward are based on the relative position of the moving object and that there is no swarm behaviour to consider. All actions are generated by a single reinforcement learning agent. This focused approach allows for a closer analysis between algorithms by removing much of the random nature which is inherent in the ecosystem. Although this problem is not as interesting as the ecosystem testbed it is used to support the results. The system contains only one degree of freedom (DOF) which is the horizontal rotation of the camera. The specific goal of the reinforcement learning algorithms is to horizontally track the moving target. Fig. 8 contains a conceptual diagram of the system. The moving target in these experiments is a black and white circle attached to the extended arm of a metronome (see Fig. 8 & 9(a)). The advantage of using a metronome is that it provides a baseline for testing due to its periodic nature. In addition, the target is placed between the camera, and a print of Monet's *Bras de la Seine pres Vetheuil*, to detect the target direction (see, *e.g.*, Sec. 9). This print is desirable for the vision system because it contains a large amount of contrast. The GUI for the software component of the vision system is similar to the ecosystem (see Fig. 7 & 10). The user is able to select the learning method and adjust algorithm specific parameters. Furthermore, the image window located on the bottom left of Fig. 10 is the direct feed from the digital camera, and the image on the right is the applied image filter (again see Sec. 9). Next, Fig. 9 depicts the vision system from the rear, top, and side views. Lastly, Table 2 lists the main hardware components of the vision system.

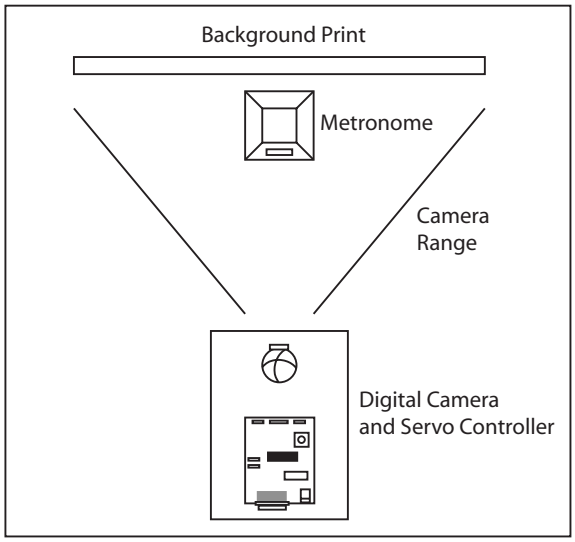



Figure 8: Vision System

# 9 Image Processing

This section introduces the image processing used to detect a moving object and determine its direction. The main objective of the vision system is to support the results of the ecosystem testbed. Consequently, the vision system employs elementary image processing techniques. An extensive discussion of image processing techniques and references is presented in [10]. An RGB color image is represented by a $M$ x $N$ matrix of color pixels, where each pixel is a triplet corresponding to the Red, Green, and Blue components of an RGB image at a specific location [10]. The RGB color model employed uses 24 bits of information per pixel which corresponds to 8 bits each for red, green, and blue. As a result there are 256 ($2^8$) values (intensity levels) for each color. Traditionally, the image origin is located at the top left corner and pixels are counted from left to right and top to bottom.

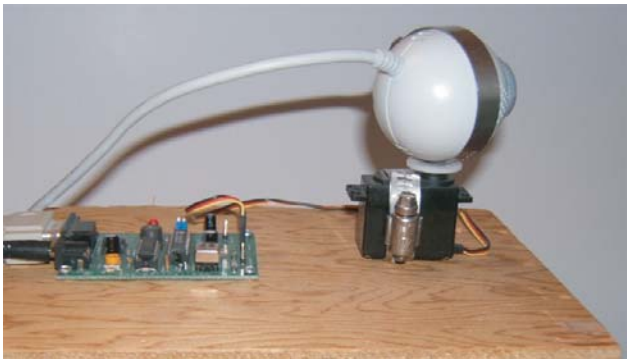
Next, movement can be detected if the background remains stationary by observing the difference between two successive frames captured from the digital camera (see, *e.g.*, Fig. 11(a) & 11(b)) [10]. Specifically, Eq. 19 is used to calculate the average difference between pixel intensities where the subscripts $a$ and $b$ refer to the first and second frames respectively.

$$j = (|R_a - R_b| + |G_a - G_b| + |B_a - B_b|)/3 \tag{19}$$

Ideally, any value of $j$ greater than 0 represents a moving object between the two frames due to the assumption that there is no movement in the background. However, in practise

(a) Rear View

(b) Top View



(c) Side View

Figure 9: Vision System

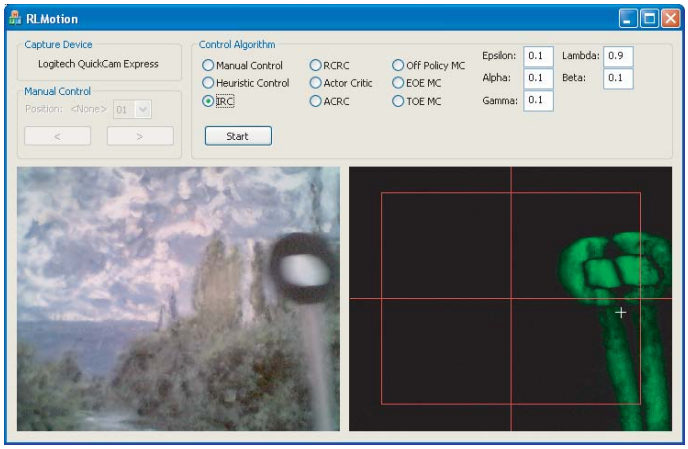
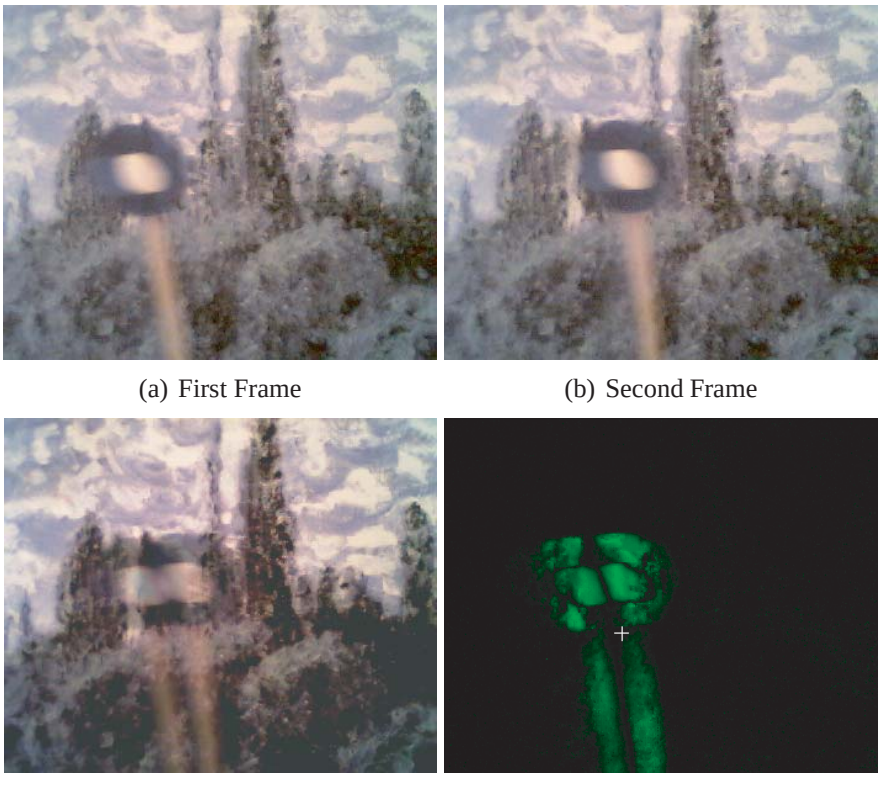



Figure 10: Vision System GUI



(a) First Frame (b) Second Frame

(c) Frames Overlapped (d) Difference of Frames

Figure 11: Image Filter

noise is introduced between successive frames due to the light sensors of the camera and analog to digital conversions. Therefore, a noise threshold is used to select only those values of $j$ which represent movement. Plotting all valid values of j produces the image shown in Fig. 11(d) by assigning $j$ to the green intensity, and making both red and blue equal to 0.

Identifying the relative position and direction of a moving object builds on the steps used to detect movement. The centroid is used to find the geometric centre of the moving object based on the intensity values of $j$ using Eq. 20 where $N$ is the number of $j$ values that are not considered noise.

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i j_i}{\sum_{i=1}^{N} j_i}, \quad \bar{y} = \frac{\sum_{i=1}^{N} y_i j_i}{\sum_{i=1}^{N} j_i} \tag{20}$$

The centroid is denoted by the small white "plus" symbol shown in Fig. 11(d). Next, the direction of the object is determined by using three consecutive frames, $a, b$ and $c$ where the centroid of the movement is calculated between frames $a$ & $b$ and frames $b$ & $c$. The case where $\bar{x}_{ab} < \bar{x}_{bc}$ represents movement in the positive direction along the $x$ axis, otherwise the opposite is true. This solution assumes that the camera is not moving, consequently, the camera control algorithm only detects object direction once the camera servo has stopped. Detecting camera movement is determined by the number of $j$ values which are larger than the noise threshold. A large percentage of $j$ values can only occur when the camera is moving due to the assumption that the background is not moving and the size of the target is constant. Therefore, another threshold on the percentage of $j$ values is used to dictate whether the control algorithm tests for object movement direction.

# 10 Tracking Problem

The task of assigning states and defining reward functions strongly affects performance, varies greatly from application to application and is more of an art than a science [69]. For example the ecosystem contains only 18 states (see Table 4) whereas the vision system has 350. The number of states for the tracking problem is a function of the pixel width of the digital camera which has a resolution 352 x 288 pixels. Specifically, the reinforcement learning state is based on the horizontal coordinate of the centroid. The main intuition being that the degree of the camera's response should be symmetric with respect to centre of the camera's field of view (which will be denoted as "the centre" for the rest of this section). For instance, consider the case when the centroid is +/- 50 pixels from the centre (see Fig 12) and is moving away from the centre in both cases. These two situations are considered the same state because in the ideal case the degree with which the camera reacts should be the same (yet opposite direction). The result is that the vision system learns the degree with which the camera should move in a specific state and the direction is based strictly on the movement of the target. Note, the opposite case of Fig. 12 is considered a separate state (*i.e.* the case when the target on either side is moving toward the centre). The algorithm used in the vision system to define the current state is located in Alg. 1. Similarly, the method used to calculate the reward is also a function of the distance of the centroid from the centre and is given in Eq. 21

$$\text{reward} = 1 - \frac{d}{W/2} \tag{21}$$

where $d$ is the distance of the centroid to the centre and $W$ is the pixel width of the digital camera (*i.e.* 352). The result of Eq. 21 is a normalized reward which equals one when the centroid of motion is located at the centre, and equals zero when the centroid is at the outside edge of the field of view. Lastly, reinforcement learning only occurs when motion has been detected, *i.e.*, the system waits for a moving target to present itself.

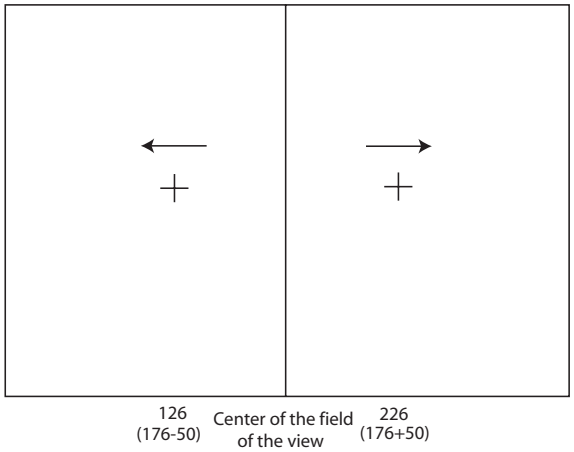



Figure 12: Vision System State Identification

**Algorithm 1**: Vision System State Definition

**Input** : Centroid x coordinate $\bar{x}$, Image Width $W$
**Output**: Vision System State $s$
**if** $\bar{x} \geq W/2$ **then**
  **if** *direction = Right* **then**
    $s = \bar{x} - W/2$
  **else**
    $s = \bar{x} - 1$
  **end**
**else**
  **if** *direction = Right* **then**
    $s = W - \bar{x} - 1$
  **else**
    $s = W/2 - \bar{x}$
  **end**
**end**

# 11 Incremental Reinforcement Comparison

The main idea behind reinforcement learning is that actions followed by positive rewards should be reinforced, and actions followed by negative rewards should be discouraged. The question is how to adapt this idea to machine learning. One solution is to use a standard or reference level, called a *reference reward*, to gauge the value of an action [69]. It is common to use average action rewards as a basis for estimating action values. Using the incremental reinforcement comparison method (IRC), a reference reward (denoted $\bar{r}$) is equated with an average of previously received rewards. Then a reward for an action is interpreted as large if it is higher than $\bar{r}$, and small if it is lower than $\bar{r}$. Let $a$, $r$, $\overline{r}$ , denote action, reward and reference reward, and let $p(s,a)$ denote an action-preference in state $s$. Preference and reference reward are computed in Eq. 22 and 23, respectively.

$$p(s,a) = p(s,a) + \beta(r - \bar{r}) \tag{22}$$

$$\overline{r} = \overline{r} + \alpha(r - \overline{r}) \tag{23}$$

where $\alpha$, and $\beta$ are positive fixed step-size parameters in the interval $(0,1]$ which influence the rate of learning [69]. After each time step, the preference $p(s,a)$ is incremented by the difference (error) between the reward $r$, and the reference reward $\bar{r}$ (a form of incremental comparison [69]). The reference reward, $\overline{r}$, is an average formed incrementally from rewards obtained in all states. Preferences are used to determine action selection probabilities according to the softmax function shown in Eq. 24.

$$\pi(s,a) = \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}} \tag{24}$$

**Definition 7 Reference Reward [69]** *A standard or reference used for judging the desirability of an action.*

Alg.2 gives the steps used to implement incremental reinforcement comparison. The infinite iteration prescribed by outer loop of Alg. 2 reflects the fact that reinforcement comparison continues forever within the learning environment. Each reward $r$ is determined by the effect the action has on the environment (see, *e.g.*, Table 6 or Eq. 21). Similarly, the next state $s'$ is determined by new conditions in the learning environment environment. At the end of the inner loop, the learning agent repeatedly enters into a new state $s$ until $s$ is terminal. Lastly, episodes are not required for this algorithm to work, however, they are introduced here in order to provide a method of comparison between incremental and rough coverage reinforcement comparison.

**Algorithm 2**: Incremental Reinforcement Comparison

**Input** : States $s \in S$, Actions $a \in A(s)$, Initialized $\alpha$, $\beta$.
**Output**: Policy $\pi(s, a)$ //where $\pi(s, a)$ is a policy in state $s$ that controls the selection of a particular action in state $s$.

**for** $(all\ s \in S, a \in A(s))$ **do**
    $p(s, a) \longleftarrow 0$;
    $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;
**end**
**while** *True* **do**
    Initialize $s$;
    **for** $(t = 0; t < T_m; t = t + 1)$ **do**
        Choose $a$ from $s$ using $\pi(s, a)$;
        Take action $a$, observe $r, s'$;
        $p(s, a) \longleftarrow p(s, a) + \beta\left[r - \bar{r}\right]$;
        $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$ ;
        $\bar{r} = \bar{r} + \alpha\left[r - \bar{r}\right]$;
        $s \longleftarrow s'$;
    **end**
**end**

## 11.1 Rough Coverage Reinforcement Comparison

This section introduces what is known as rough coverage reinforcement comparison (RCRC). Recall that IRC uses an incremental average of all recently received rewards. By contrast, RCRC uses average rough coverage values in a lower approximation space. That is, during RCRC learning, the degree that a block of equivalent behaviours is covered by a set of behaviours representing a standard, is computed [43]. This intuition matches the idea of a reference reward defined in Sect. 11 as a standard or reference level used to judge the "goodness" of a preceding action. Lower rough coverage is a useful tool because it determines to what degree a block is a member of $B_*D$ of behaviours that have been accepted by a learning agent. The reference or standard used in RCRC is $\bar{r} = \overline{\nu_B(B_{ac}(x), B_*D)}$. Specifically, let $\mathcal{B}$ denote a set of blocks representing actions in a set of sample behaviours $\mathcal{S}$ in an ethogram as in Eq. 25,

$$\mathcal{B} = \{B_{ac}(x) | x \in \mathcal{S}\}\,. \tag{25}$$

Then $\overline{r}$ is defined as average rough coverage as shown in Eq. 26.

$$\bar{r} = \frac{1}{card(\mathcal{B})} \sum\nolimits_{i=1}^{card(\mathcal{B})} \nu_B\left(B_{ac}(x), B_*D\right), \tag{26}$$

where $B_{ac}(x) \in \mathcal{B}$. Computing the average lower rough coverage value for action blocks extracted from an ethogram implicitly measures to what extent action $a$ has been selected relative to state $s$. Further, discretization is performed on the ethogram feature values before partitioning sample behaviours into blocks, deriving $B_*D$ and calculation of average lower coverage values. To achieve very fast discretization, a non-optimal discretization method based on the standard deviation of a given column of feature values is used instead of the optimal discretization algorithm introduced in [30] (see also [20]). This approach to feature value discretization is non-optimal because no attempt is made to find the minimum number of intervals.

A rough coverage reinforcement comparison algorithm used in the ecosystem testbed is given in Alg. 3. Just as before, the infinite iteration prescribed by the outer loop reflects the fact that reinforcement comparison by a learning agent continues forever. Similarly, rewards, $r$, and the next state $s'$ is determined by the learning environment. At the end of the inner loop, the learning agent repeatedly enters into a new state $s$ until $s$ is terminal. Finally, Alg. 3 implements Eq. 9 to compute rough coverage values at the end of every episode. Fig. 13 shows a plot of normalized average reward values for both reinforcement comparison methods where the learning agent is the swarm within the ecosystem. Similarly, Fig. 14 is a plot of the normalized average reward values for the vision system where there is a single learning agent controlling the camera.

**Algorithm 3**: Rough Coverage Reinforcement Comparison

**Input** : States $s \in S$, Actions $a \in A(s)$, Initialized $\beta$.

**Output**: Policy $\pi(s,a)$ //where $\pi(s,a)$ is a policy in state $s$ that controls the selection of a particular action in state $s$.

**for** $(all\ s \in S, a \in A(s))$ **do**

  $p(s,a) \longleftarrow 0$;

  $\pi(s,a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;

**end**

$\bar{r} \longleftarrow 0$;

**while** *True* **do**

  Initialize $s$;

  **for** $(t = 0; t < T_m; t = t + 1)$ **do**

    Choose $a$ from $s$ using $\pi(s,a)$;

    Take action $a$, observe $r, s'$;

    $p(s,a) \longleftarrow p(s,a) + \beta\,[r - \bar{r}]$;

    $\pi(s,a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$ ;

    $s \longleftarrow s'$;

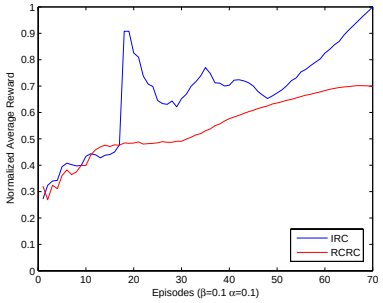
  **end**

  Extract ethogram table $IS_{swarm} = (U_{beh},\ A,\ d)$ ;

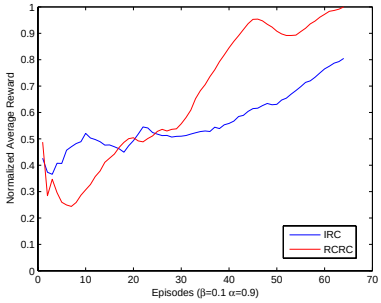
  Discretize feature values in $IS_{swarm}$ ;

  Compute $\bar{r}$ as in Eq. 26 using $IS_{swarm}$;

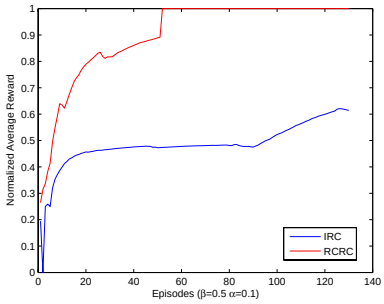
**end**



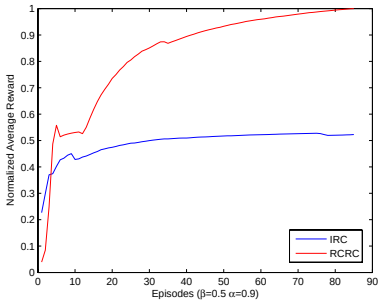

(a) $\beta = 0.1, \alpha = 0.1$



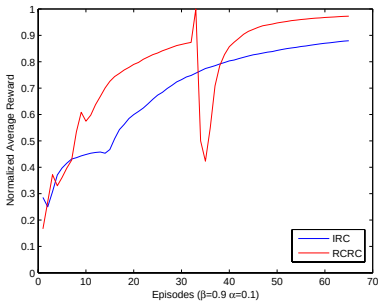

(b) $\beta = 0.1, \alpha = 0.9$



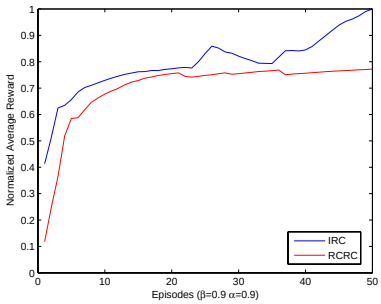

(c) $\beta = 0.5, \alpha = 0.1$




(d) $\beta = 0.5, \alpha = 0.9$

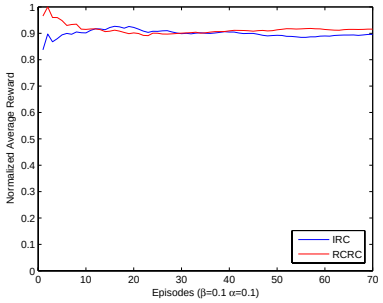



(e) $\beta = 0.9, \alpha = 0.1$

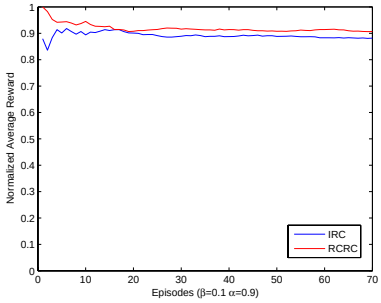



(f) $\beta = 0.9, \alpha = 0.9$

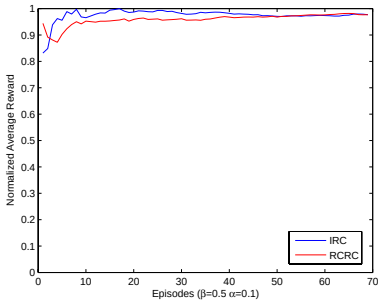
Figure 13: Ecosystem Testbed Normalized Average Reward: IRC vs RCRC



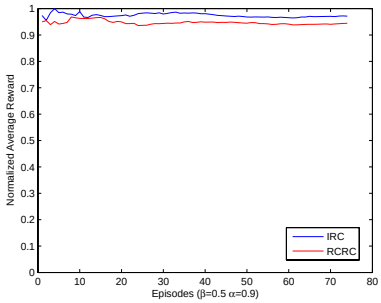

(a) $\beta = 0.1, \alpha = 0.1$



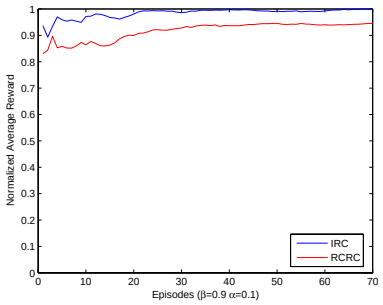

(b) $\beta = 0.1, \alpha = 0.9$



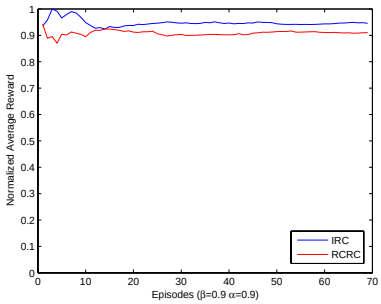

(c) $\beta = 0.5, \alpha = 0.1$




(d) $\beta = 0.5, \alpha = 0.9$




(e) $\beta = 0.9, \alpha = 0.1$




(f) $\beta = 0.9, \alpha = 0.9$

Figure 14: Vision System Normalized Average Reward: IRC vs RCRC

# 12 Actor-Critic Methods

Actor-critic (AC) methods are temporal difference (TD) learning methods with a separate memory structure to represent policy independent of the value function used (see, *e.g.* [8, 51, 69]). AC methods are an extension to the idea of reinforcement comparison (see, *e.g.*, [69]). In an AC method, the policy structure is known as the *actor*, since it is used to select actions, and the estimated value function is known as the *critic* because it criticizes the actions made by the actor. The estimated value function, $V(s)$, is an average of the rewards received while in state $s$. After each action selection, the critic evaluates the quality of the selected action using $\delta$ in Eq. 27 which represents the error (labelled the TD error) between successive estimates of the expected value of a state [54].

$$\delta = r + \gamma V(s') - V(s) \tag{27}$$

Where $\gamma$ is the discount rate used to determine the current value of future rewards [69]. The actor then uses $\delta$ as an internal reinforcement to adjust the probability of an action being selected in the future [54]. This is accomplished just as in reinforcement comparison by using the softmax relationship and preference values. The only difference is preference is now calculated in Eq. 28. Alg. 4 shows the Actor-Critic method implementation. This algorithm uses an array *Count* to calculate the value of a state $V(s)$, which is the incremental average of rewards obtained in a specific state (*i.e.* Eq. 18 is used with $w_i = 1$). In addition, this algorithm is episodic, and is executed continuously.

$$p(s,a) \leftarrow p(s,a) + \beta\delta \tag{28}$$

## 12.1 Actor-Critic Methods using Rough Coverage

This section introduces what is known as a rough-coverage actor-critic (ACRC) method. The preceding section is just one example of Actor-Critic methods [69]. In fact common variations include additional factors which vary the amount of credit assigned to selected actions. This is most commonly seen in calculating the preference, $p(s,a)$. The rough coverage form of the Actor-Critic method calculates preference values as shown in Eq. 29.

$$p(s,a) \leftarrow p(s,a) + \beta\left[\delta - \bar{r}\right] \tag{29}$$

where $\bar{r}$ is reminiscent of the idea of a reference reward used during reinforcement comparison. Recall that incremental reinforcement comparison uses an incremental average of all recently received rewards as suggested in [69]. By contrast, rough coverage reinforcement comparison (RCRC) uses average rough coverage of selected blocks in the

**Algorithm 4**: Actor-Critic

**Input** : States $s \in S$, Actions $a \in A(s)$, Initialized $\alpha$, $\gamma$.

**Output**: Policy $\pi(s, a)$ //where $\pi(s, a)$ is a policy in state $s$ that controls the selection of a particular action in state $s$.

**for** $(all\ s \in S, a \in A(s))$ **do**

  $p(s, a) \longleftarrow 0$;

  $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;

  $Count(s) \longleftarrow 0$;

**end**

**while** *True* **do**

  Initialize $s$;

  **for** $(t = 0; t < T_m; t = t + 1)$ **do**

    Choose $a$ from $s$ using $\pi(s, a)$;

    Take action $a$, observe $r, s'$;

    $Count(s) \longleftarrow Count(s) + 1$;

    $V(s) \longleftarrow V(s) + 1/Count(s)\,[r - V(s)]$;

    $\delta = r + \gamma V(s') - V(s)$;

    $p(s, a) \longleftarrow p(s, a) + \beta\delta$;

    $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$ ;

    $s \longleftarrow s'$;

  **end**

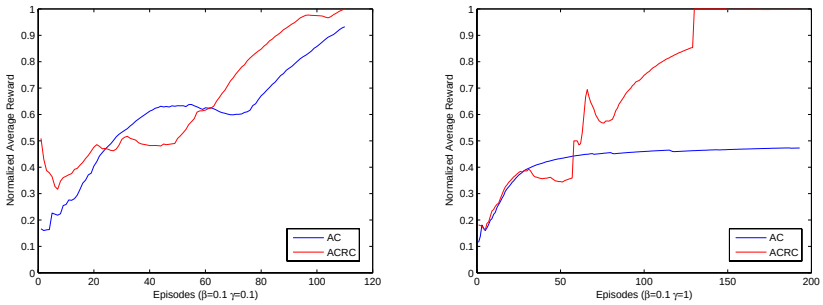
**end**

lower approximation of a set [43]. Intuitively, this means action probabilities are now governed by the coverage of an action by a set of equivalent actions which represent a standard. Rough coverage values are defined within a lower approximation space. Alg. 5 is the ACRC learning algorithm used in the ecosystem for actor-critic methods using lower rough coverage. Notice that the only difference between Algorithms 4 and 5 is the addition of the reference reward, $\bar{r}$, which is calculated using rough coverage. Fig. 15 contains plots of average reward values for both actor-critic learning methods for results obtained from the ecosystem testbed. Similarly, Fig. 16 contains the test results from the vision system experiments.

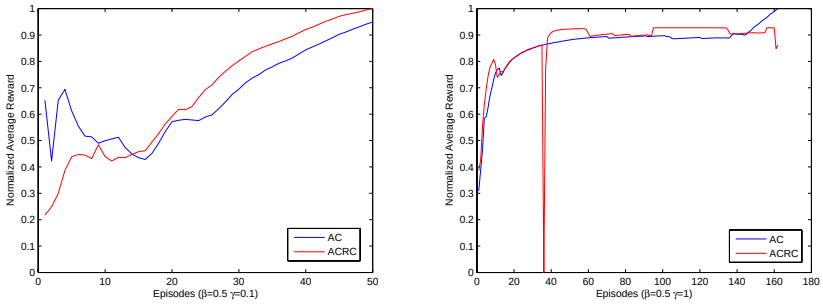
**Algorithm 5**: Rough Coverage Actor-Critic

**Input** : States $s \in S$, Actions $a \in A(s)$, Initialized $\alpha$, $\gamma$.
**Output**: Policy $\pi(s, a)$ //where $\pi(s, a)$ is a policy in state $s$ that controls the selection of a particular action in state $s$.

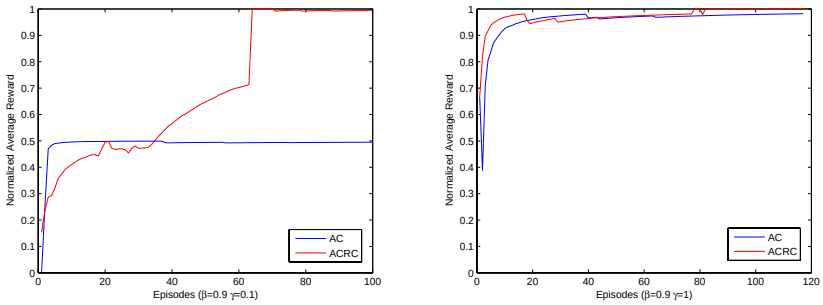
**for** $(all\ s \in S, a \in A(s))$ **do**
  $p(s, a) \longleftarrow 0$;
  $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$;
  $Count(s) \longleftarrow 0$;
**end**
**while** *True* **do**
  Initialize $s$;
  **for** $(t = 0; t < T_m; t = t + 1)$ **do**
    Choose $a$ from $s$ using $\pi(s, a)$;
    Take action $a$, observe $r, s'$;
    $Count(s) \longleftarrow Count(s) + 1$;
    $V(s) \longleftarrow V(s) + 1/Count(s)\,[r - V(s)]$;
    $\delta = r + \gamma V(s') - V(s)$;
    $p(s, a) \leftarrow p(s, a) + \beta\,[\delta - \bar{r}]$;
    $\pi(s, a) \longleftarrow \frac{e^{p(s,a)}}{\sum_{b=1}^{|A(s)|} e^{p(s,b)}}$ ;
    $s \longleftarrow s'$;
  **end**
  Extract ethogram table $IS_{swarm} = (U_{beh},\ A,\ d)$ ;
  Discretize feature values in $IS_{swarm}$ ;
  Compute $\bar{r}$ as in Eq. 26 using $IS_{swarm}$;
**end**




(a) $\beta = 0.1$,$\gamma = 0.1$ (b) $\beta = 0.1$, $\gamma = 1$

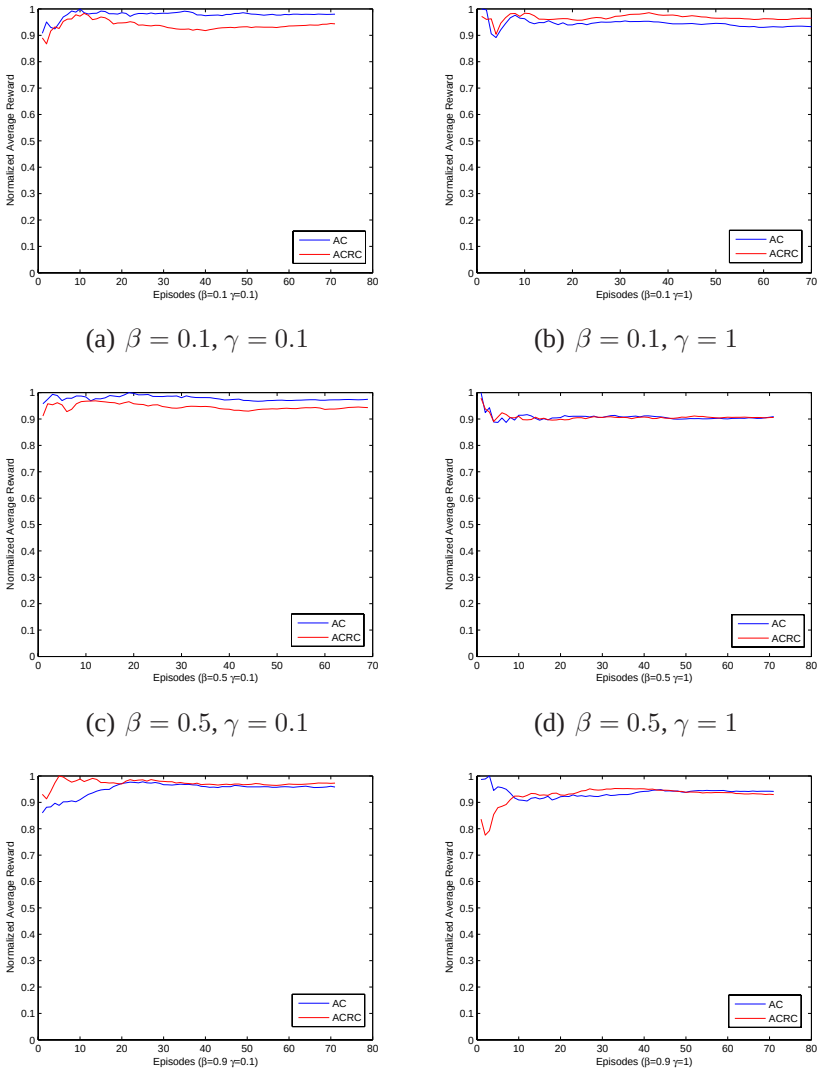



(c) $\beta = 0.5$, $\gamma = 0.1$ (d) $\beta = 0.5$, $\gamma = 1$




(e) $\beta = 0.9$, $\gamma = 0.1$ (f) $\beta = 0.9$, $\gamma = 1$

Figure 15: Ecosystem Testbed Normalized Average Reward: AC vs ACRC




(a) $\beta = 0.1, \gamma = 0.1$ (b) $\beta = 0.1, \gamma = 1$

(c) $\beta = 0.5, \gamma = 0.1$ (d) $\beta = 0.5, \gamma = 1$

(e) $\beta = 0.9, \gamma = 0.1$ (f) $\beta = 0.9, \gamma = 1$

Figure 16: Vision System Normalized Average Reward: AC vs ACRC

# 13 Monte Carlo Off-Policy Learning Control Method

A Monte Carlo control algorithm is dubbed "off policy" in the case where the policy that is being revised is not the one being used to make decisions. The two policies are $\pi'$ (called a $behaviour$ policy used to generate behaviour) and $\pi$ (called an $estimation$ policy). The advantage to this scheme is that the behaviour policy $\pi'$ can be used for exploration in sampling all possible actions while the estimation policy $\pi$ can be deterministic and can be improved after each episode [69]. In this way information about non-greedy actions can still be acquired without sacrificing the deterministic policy which would have fallen short of this task on its own. Notice that a requirement for this method to work is that the behaviour policy $\pi'$ must have a non-zero probability of returning to a state (*i.e.* $\pi'(s,a) > 0$) for all state action pairs [69]. This is guaranteed through the use of an $\epsilon$-soft policy such as Eq. 35. The basic idea is to choose actions during an episode such that the expected return $E[R]$ is maximized. To see how this is done, first consider weighted sampling of $R$-values.

**Definition 8 Behaviour Policy [69]** *A policy used to generate behaviour.*

**Definition 9 Estimation Policy [69]** *A policy which evaluates the actions selected under the behaviour policy.*

## 13.1 Weights

Traditional Monte Carlo methods differ from the weighted average in Eq. 16 only by the definition of $w_k$ given by

$$w_k = \prod_{i=n+1}^{k} \frac{p_i(s)}{p'_i(s)}. \tag{30}$$

where $p_i(s)$ and $p'_i(s)$ are the probabilities that a complete sequence occurs under the polices $\pi$ and $\pi'$ respectively. An estimate of the expected value of $R$ for $Q^\pi(s,a)$ after observing $m-n$ returns from state $s$ is then given by

$$Q(s,a) \approx \frac{1}{W} \sum_{k=n+1}^{m} R_k w_k. \tag{31}$$

where $W$ denotes the sum of the weights. The difficultly in using Eq. 30 with Eq. 31 is that probabilities $p_i(s)$ and $p'_i(s)$ are usually unknown [69]. However, here the only requirement is that their ratios which can be determined without the knowledge of the systems state transition probabilities (usually not known in Monte Carlo applications).

Moreover, the values of policies $p_i(s)$ and $p'_i(s)$ can be used after suitable definition. Assume that $p_i(s)$ is defined as in Eq. 32.

$$p_i(s) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P^{a_k}_{s_k s_{k+1}}. \tag{32}$$

where $P^{a_k}_{s_k s_{k+1}}$ is the conditional probability that a system in state $s_k$ will be in state $s_{k+1}$ after its next transition given that action $a_k$ was selected [69]. Then the following equation is obtained

$$\frac{p_i(s_t)}{p'_i(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P^{a_k}_{s_k s_{k+1}}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) P^{a_k}_{s_k s_{k+1}}} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}. \tag{33}$$

Thus the weight needed in Eq. 31, $p_i(s)/p'_i(s)$, depends only on the two policies and not at all on the environment's dynamics. Therefore, Eq. 30 can be replaced with

$$w_k = \prod_{i=n+1}^{k} \frac{\pi\left(s_i, a_i\right)}{\pi'\left(s_i, a_i\right)}. \tag{34}$$

Notice that in the case where $\pi(s, a)$ is an arbitrary $\epsilon$-soft policy, the probability of selecting an action $a$ determined by policy $\pi(s, a)$ is computed using Eq. 35.

**Definition 10** **$\epsilon$-Soft Policy [69]** *A policy which assigns the probability, $\epsilon$, to the action, $a^*$, which is the action most likely to produce the highest reward. Furthermore, the probability $1 - \epsilon$ is distributed among the remaining actions as shown in Eq. 35.*

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|}, & \text{if } \mathrm{a} = \mathrm{a}^*, \\ \frac{\epsilon}{|A(s)|}, & \text{if } \mathrm{a} \neq \mathrm{a}^*. \end{cases} \tag{35}$$

Assume that $\pi$ is a deterministic policy, where $\epsilon$ is zero, then $\pi(s, a) = 1$, where $a = a^*$, *i.e.*, the action $a$ is an optimal action $a^*$. Then a simplified form of $w_k$ is obtained as shown in Eq. 36.

**Definition 11** **Deterministic Policy [69]** *Let $\pi(s)$ denote a policy which over time selects the same action for a given state.*

$$w_k = \prod_{k=t}^{k=T_i(s)-1} \frac{1}{\pi'\left(s_k, a_k\right)}. \tag{36}$$

This line of reasoning yields the Prop. 2 and Prop. 3.

**Property 2** *A weight on a return can be defined as a function of episodic behaviour policies $\pi$ and $\pi'$ as shown in Eq. 34.*

In the case where $\pi$ is an $\epsilon$-soft policy defined relative to the greatest expected return (see Eq. 35), where $\epsilon$ is zero in Eq. 35, then the weights on returns can be formulated using only $\pi'$ as shown in Eq. 36.

**Property 3** *A weight on returns can be defined as a function of a single episodic behaviour policy $\pi'$ as shown in Eq. 36.*

## 13.2 Weighted Sampling Based On Approximation Spaces

The framework provided by an approximation space makes it possible to derive pattern-based rewards, which are used by swarms that learn to choose actions in response to perceived states of their environment (see, *e.g.*, Fig. 4). The notation $\bar{\nu}_{ac}$ denotes an average rough coverage value computed within the context of an approximation space using Eq. 9 as shown in Eq. 37.

$$\bar{\nu}_{ac} = \sum_{i=0}^{n} \nu(B_{ac}(x_i), B_* D)/n. \tag{37}$$

The decision table $DT = (U_{beh}, A, d)$ in Fig. 4 provides a record of behaviour patterns observed during different episodes in the life of system. The action value function $Q^{\pi}(s, a)$ can be approximated using the weighted average shown in Eq. 38.

$$Q^{\pi}(s, a) \approx \frac{1}{W} \sum_{k=1}^{m} R_m w_m. \tag{38}$$

where the weight defined in Eq. 36 is used in conjunction with Eq. 37 to obtain an approximation space-based weight as shown in Eq. 39.

$$w_m = \frac{\prod_{k=t_m}^{T_m-1} \frac{1}{\pi'(s_k,a_k)}}{\frac{1}{(1+\bar{\nu}_{ac})}} = (1 + \bar{\nu}_{ac}) \prod_{k=t_m}^{T_m-1} \frac{1}{\pi'(s_k, a_k)}. \tag{39}$$

In other words, in approximating the expected value of $R_m$, the samples $R_m$ can be weighted within the context of an approximation space. This leads to Prop 4.

**Property 4** *The expectation $E[R_m]$ can be estimated with approximation space-based weighted sampling.*

## 13.3 Common Off-Policy Monte Carlo Learning

This section briefly considers an off-policy Monte Carlo algorithm using weighted sampling based on Eq. 36. The conventional off-policy algorithm used in both the ecosystem and vision experiments is given in Alg. 6 (see,*e.g.*, [69]). It is assumed that $\tau$ is the latest time at which $a_t \neq \pi(s_t)$, which means that after time $\tau$ each action selected by the behaviour policy is the same action that would be selected by the estimation policy.

**Algorithm 6**: Off-Policy Monte Carlo Control Algorithm

**Input** : States $s \in S$, Actions $a \in A(s)$ //$A(s)$ is a set of actions in state $s$.
**Output**: Policies $\pi(s)$ //where $\pi(s)$ is a policy used to select action in state $s$.
**for** $(all\ s \in S, a \in A(s))$ **do**
  $\pi(s)$ is randomly chosen;
  $Q(s,a) \longleftarrow$ arbitrary; //where $Q$ is the value of an action $a$ is state $s$
  $N(s,a) \longleftarrow 0$; //numerator of $Q(s,a)$
  $D(s,a) \longleftarrow 0$; //denominator of $Q(s,a)$
**end**
**while** *True* **do**
  Select action policy $\pi'(s,a)$ to generate an episode:
  $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$;
  // $r_i$ is the reward on action $a_{i-1}$
  $\tau \longleftarrow$ the latest time at which $a_\tau \neq \pi(s_\tau)$;
  **for** *each pair* $(s,a)$ *in an episode at time* $\tau$ *or later* **do**
    $t \longleftarrow$ the time of first occurrence of $s, a$ st. $t \geq \tau$;
    $w_{ac,t} \longleftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k,a_k)}$;
    $N(s,a) \longleftarrow N(s,a) + w_{ac,t} R_t$; //$R_t$ as Eq. 12
    $D(s,a) \longleftarrow D(s,a) + w_{ac,t}$;
    $Q(s,a) \longleftarrow \frac{N(s,a)}{D(s,a)}$;
  **end**
  **for** *each* $s \in S$ **do**
    $\pi(s) \longleftarrow argmax_a Q(s,a)$;
  **end**
**end**

The approach to weighted sampling in Alg. 6 reflects the application of Prop. 3. Notice, also, that Alg. 6 can be simplified using the update rule from Prop 1. There is a disadvantage in using off-policy control and it comes in the form of how the estimation policy learns. Toward the end of an episode, the behaviour policy is modified to select

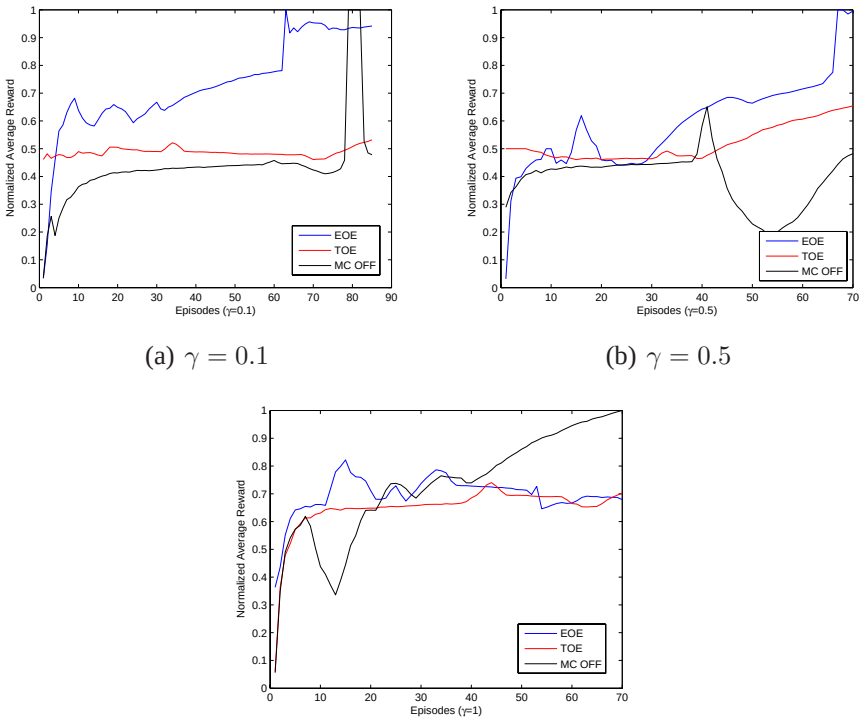
only greedy actions, at this point, the estimation policy learns from the choices made. As a result, only the end of an episode provides the information necessary for the estimation policy to be improved. This ends up slowing the learning process in comparison to the on-policy method which is able to make use of the information in the entire episode [69].

## 13.4 Off-Policy Monte Carlo Learning With Approximation Spaces

An approximation spaced based alternative to the weighted sampling method in Alg. 6 is briefly explored in this section. To obtain the new form of weighted sampling in Alg. 7, Alg. 6 has been rewritten using the update rule from Prop. 3 and the approximation space-based weighted sampling estimates of the expected value of $R_m$ from Prop. 4.

A tail-of-episode (TOE) approach to off-policy Monte Carlo control is used in Alg. 7. That is, the application of the new approach to weighted sampling only occurs after a swarm has gained some experience at the beginning of each episode. Experience at the beginning of an episode is represented by an extracted ethogram. The knowledge gained from each ethogram influences the weighted sampling (guided by average rough coverage values for each action) that governs behaviour in the tail of an episode. In some sense, Alg. 7 mimics the approach Ulam suggests in learning from experience (see, *e.g.*, [72]). For example, consider trying to estimate the success of the next solitaire hand based on previous experience with solitaire hands. In this case, the ethogram extracted at some point in time after the beginning of each episode of Alg. 7 reflects the experience of a swarm in coping with an environment. The approximation space-based TOE off-policy Monte Carlo algorithm was introduced in [25], and elaborated in [46]. Lastly, Figures 17 & 18 give results for the normalized average reward in the ecosystem and vision system respectively, Figures 19 & 18 contain results for normalized total Q values, and Figures 21 & 22 are the plots obtained by finding the RMS between the average value of $Q(s, a)$ at the end of an episode and each individual value of $Q(s, a)$.

Other forms of the approximation space-based off-policy algorithm are possible. For example, consider moving ethogram extraction to end of each episode, and use rough coverage based weighted sampling at the beginning of the next episode. This is called the end-of-episode (EOE) method that has been reported in [41]. Detailed consideration of alternative methods of approximation space-based off-policy algorithms are outside the scope of this thesis.

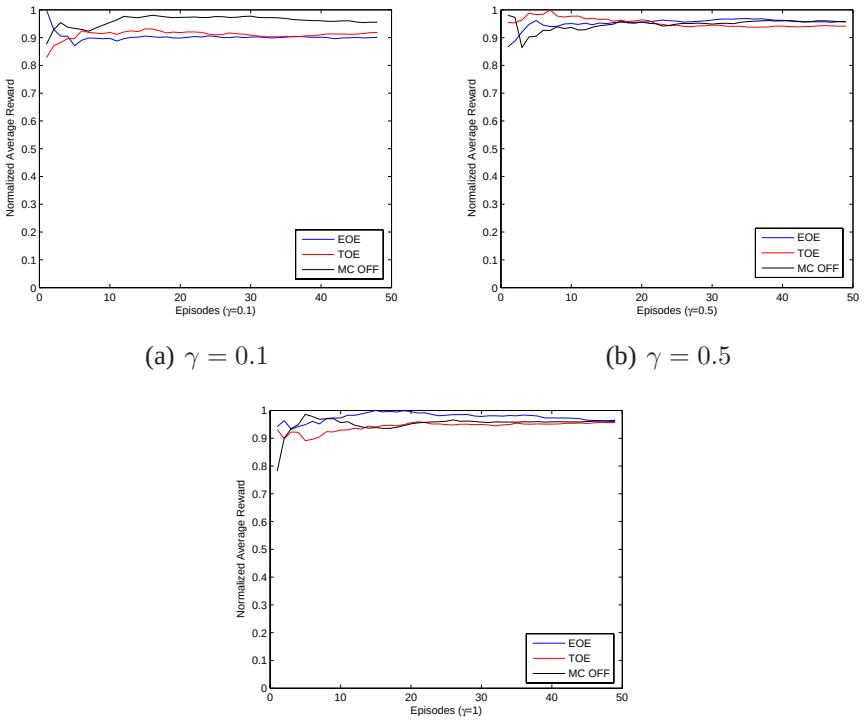



(a) $\gamma = 0.1$

(b) $\gamma = 0.5$

(c) $\gamma = 1$

Figure 17: Ecosystem Testbed Results Normalized Average Reward: Off Policy Tests

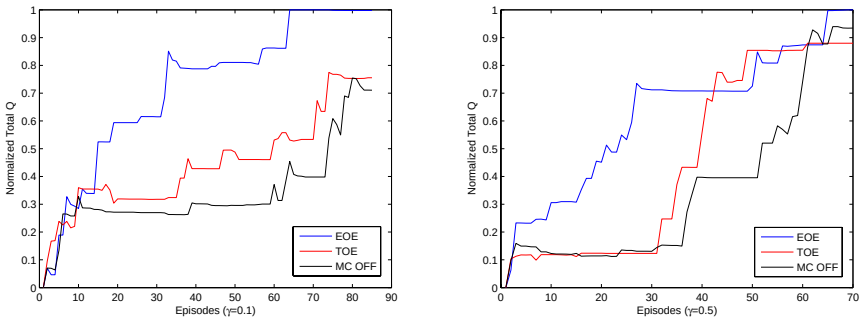



(a) $\gamma = 0.1$

(b) $\gamma = 0.5$

(c) $\gamma = 1$

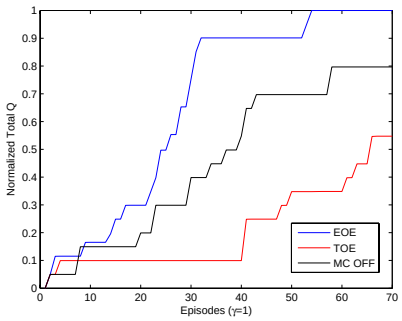
Figure 18: Vision System Results Normalized Average Reward: Off Policy Tests




(a) $\gamma = 0.1$

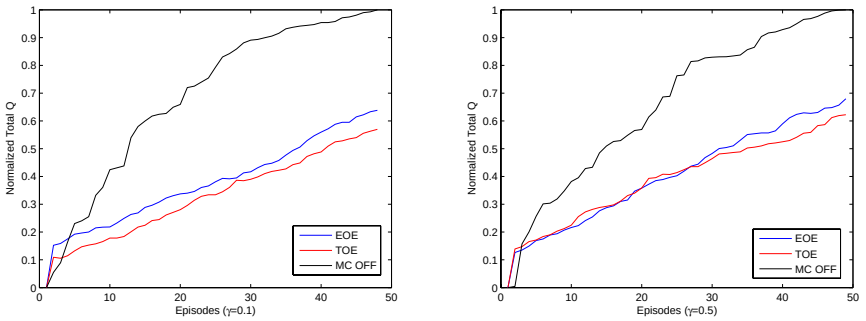



(b) $\gamma = 0.5$

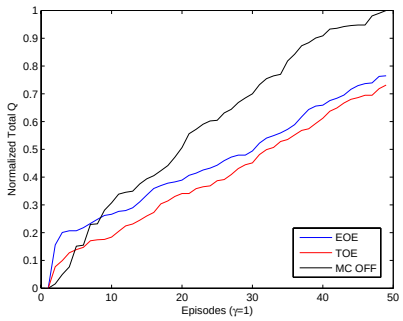



(c) $\gamma = 1$

Figure 19: Ecosystem Testbed Results Normalized Total Q: Off Policy Tests

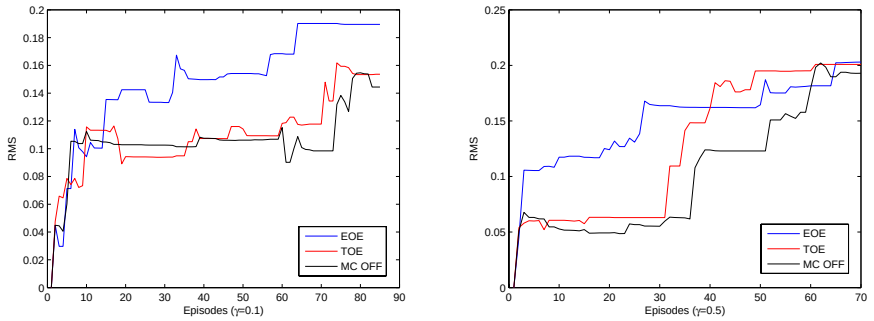



(a) $\gamma = 0.1$ (b) $\gamma = 0.5$

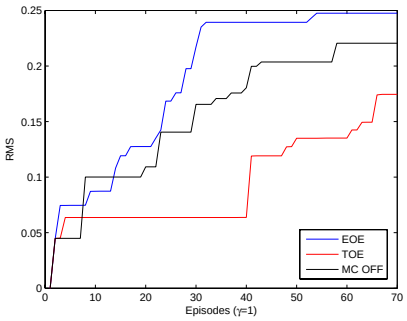



(c) $\gamma = 1$

Figure 20: Vision System Results Normalized Total Q: Off Policy Tests

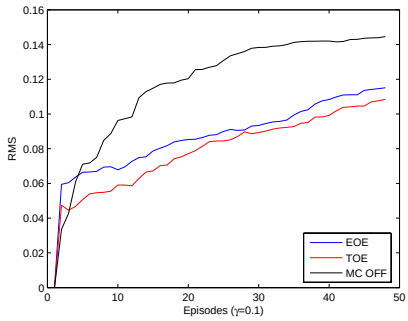



(a) $\gamma = 0.1$ (b) $\gamma = 0.5$

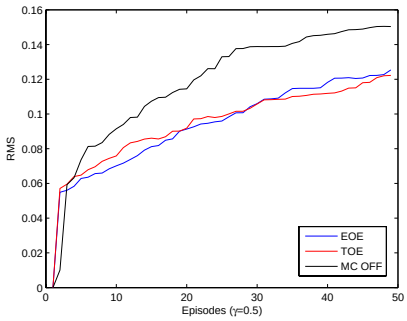



(c) $\gamma = 1$

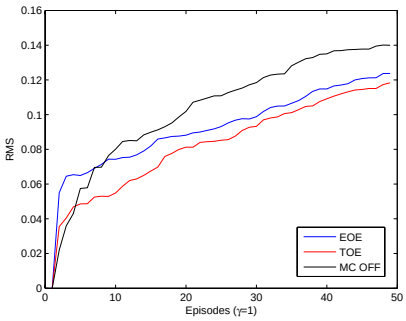
Figure 21: Ecosystem Testbed Results RMS: Off Policy Tests




(a) $\gamma = 0.1$




(b) $\gamma = 0.5$




(c) $\gamma = 1$

Figure 22: Vision System Results RMS: Off Policy Tests

**Algorithm 7**: Off Policy Monte Carlo Learning with Approximation Spaces

**Input** : States $s \in S$, Actions $a \in A(s)$
**Output**: Policy $\pi(s)$.
**for** $(all\ s \in S, a \in A(s))$ **do**
  $\pi(s)$ is randomly chosen;
  $\pi'(s,a)$ is randomly chosen;
  $R(s,a) \longleftarrow 0$;
  $C(s,a) \longleftarrow 0$; //counts the number of times action $a$ was selected in state $s$
  $Q(s,a) \longleftarrow$ arbitrary; //where $Q$ is the value of an action $a$ is state $s$
  $W(s,a) \longleftarrow 1$
**end**
**while** *True* **do**
  Use $\pi'(s,a)$ to generate an episode:
  $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$; // $r_i$ is the reward on action $a_{i-1}$
  $\tau \longleftarrow$ the first time at which $a_\tau \neq \pi(s_\tau)$;
  Extract ethogram table $DT_{swarm} = (U_{beh},\ A,\ d)$;
  Discretize feature values in $DT_{swarm}$;
  Compute $\bar{\nu}_{ac}$ as in Eq. 37 using $DT_{swarm}$;
  **for** $(t = \tau;\ t < T;\ t = t+1)$ **do**
    $C(s_t,a_t)_{n+1} \longleftarrow C(s_t,a_t)_n + 1$;
    $R(s_t,a_t)_{n+1} \longleftarrow R(s_t,a_t)_n + \frac{1}{C(s_t,a_t)_{n+1}}[r_t - R(s_t,a_t)_n]$;
    $w_{n+1} = (1+\bar{\nu}_{ac}) \prod_{k=t}^{T-1} \frac{1}{\pi'(s_k,a_k)}$;//$ac$ is the interval containing $a_t$
    $W_{n+1}(s_t,a_t) \longleftarrow W_n(s_t,a_t) + w_{n+1}$;
    $Q_{n+1}(s_t,a_t) \longleftarrow Q_n(s_t,a_t) + \frac{w_{n+1}}{W_{n+1}(s_t,a_t)}\,[R_{n+1}(s_t,a_t) - Q_n(s_t,a_t)]$;
  **end**
  **for** $(all\ s \in S, a \in A(s))$ **do**
    $C(s,a) \longleftarrow 0$;
    $R(s,a) \longleftarrow 0$;
  **end**
  **for** $(eachs \in S)$ **do**
    $\pi(s) \longleftarrow argmax_a Q(s,a)$;
  **end**
**end**

# 14 Analysis

This section considers the test results obtained from both the ecosystem testbed and the vision system. First, the plots obtained from the vision system indicated that there is not much separation between the algorithms being compared (*i.e.* the plots are quite similar). This is a result of two distinct design choices. The first consideration is that the system only learns when motion is detected. Specifically, the absence of motion is not considered a state in the environment, and there is no (negative) reward assigned for choosing an action which leads to this state. Second, when motion is detected the centroid is usually calculated to be within a central boundary of the camera's field of view. This is due to the symmetry of the target being tracked and the fact that the camera is trying to keep the target within the centre of the field of view. Consequently, the reward received by the system has a small standard deviation. One possible solution to this problem would be to vary the reward exponentially with the distance of the centroid from the centre of the field of view, and negatively reinforce (punish) any action which results in an absence of motion (*i.e.* losing the target).

Next, some of the ecosystem plots contain sections where there are large spikes or dips. These anomalies can be attributed to a number of factors. First, the maximum reward assigned in the ecosystem can only be attained in a subset of the total states. Therefore, the reward will drop if the swarm chooses a random action which causes a transition out of this subset. Also, there are times when the environment forces the swarm into a state which causes low rewards. For example, lightning causes the swarms to dismount from the skywire into the path of an adversary which reduces the net charge of the swarm. Also, the plot anomalies can also be caused by the addition or removal of a swarm member. New members can bring experience to the swarm which causes the reward to increase.

The plots in Fig. 13 & 14 indicate that the RCRC method either had a higher average reward than the incremental RC method, or was comparable. This suggests that average rough coverage is better than average reward as a standard for judging the "goodness" of a previous action. This makes sense because rough coverage is used to determine to what degree each set of equivalent behaviours are covered by a set of behaviours representing a standard. Furthermore, the plots in Fig. 15 & 16 indicate that the RAC method had a higher average reward than the AC method. This suggests that average rough coverage is better than average reward as a standard for judging the "goodness" of a previous action. This makes sense if one considers the significance of computing rough coverage to determine to what degree blocks of equivalent action-based behaviours are a part of a set of behaviours representing a standard. This is also reminiscent of findings reported elsewhere (see, *e.g.*, [40, 43, 44]).

Next, the results of the experiments with different parameter values for each of the three off-policy Monte Carlo reinforcement learning methods presented in this thesis are presented, namely, the conventional, TOE and EOE off-policy methods. A sampling of normalized reward, normalized $Q(s, a)$ values and corresponding RMS values for the TOE, EOE and conventional off-policy Monte Carlo learning are shown in Fig. 17 - 22 respectively. In each case, the TOE and EOE approaches to off-policy learning consistently do better than the conventional off-policy method. For the most part, the sample $Q(s, a)$ values using these two rough off-policy learning methods are fairly close together. The TOE off-policy methods tends to do better than the EOE off-policy method.

# 15 Conclusion

The contribution of this thesis is a comprehensive introduction to a pattern-based evaluation of behaviour during reinforcement learning using approximation spaces. Furthermore, this thesis presents the implementation of two learning environments (the ecosystem testbed, and the monocular vision system) used to compare seven different forms of reinforcement learning. Namely, two forms of Reinforcement Comparison, and Actor-Critic methods as well as three forms of Monte-Carlo off-policy control. A basic assumption made in this thesis is that learning is carried out in a non-stationary environment. In general the learning techniques presented in this thesis are ideal for problems in which little or no information is known about the environment. They allow for the discovery of better action selection choices by exploring an unknown environment while exploiting knowledge gained in the past. The results obtained by the pattern-based approach to reinforcement learning are promising. At the very least they indicate that this approach is comparable to traditional reinforcement learning techniques. However, more work is required to determine (both theoretically and empirically) which method is superior. Additionally in future work, several other forms of reinforcement learning methods will be considered. Finally, the monocular vision requires significant improvements in order to be a practical solution for tracking moving objects.

# A Swarmbot Testbed

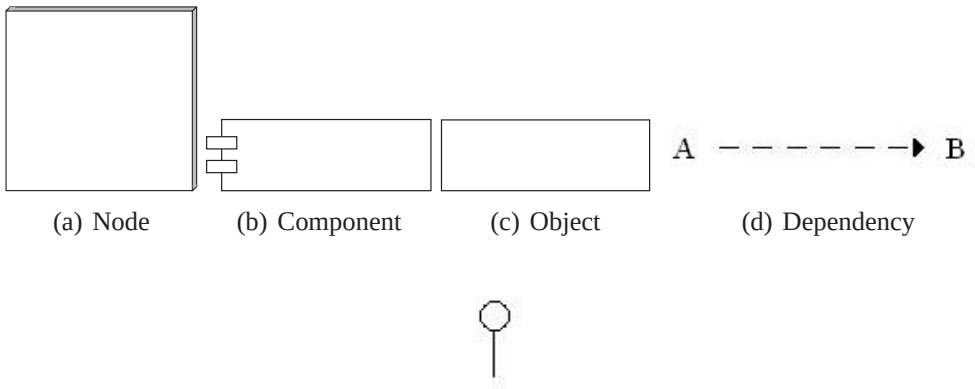
The testbed makes it possible to experiment with bots that must learn to cooperate in a number of ways (*e.g.* navigate pasts obstacles). The goal for swarms of cooperating bots within the testbed is to learn to correctly inspect the towers within the simulation while learning to avoid the many hazards. The hazards are represented by adversaries which can take the form of lighting, high wind, emf, or attacks by animals. This testbed has a computation kernel that sets up an approximation space (defined in Sect. 3) at the end of each episode. As was mentioned in Sect. 7 Fig. 6(a) is a screen shot of a testbed that automates the production of ethograms to provide a record of observed swarmbot (sbot) behaviour patterns. Swarm ethograms are appended after each time step with swarm specific data which is used at the end of an episode. Furthermore, Table 3 contains a description of the symbols that appear in Fig 6(a). The basic symbols visible during ecosystem operation are shown in Fig. 6(a), namely, bot (tiny disk), bot charge (number next to bot), swarm (cluster of tiny disks), average swarm charge (number next to cluster of tiny disks), sensor range (circle surrounding an agent or swarm), adversary (larger solid disk inside circle), power line (line) and power tower (solid square connected to power line). As was mentioned in Sect. 6 the focus of this testbed is on swarm behaviour. Moreover, the bots are based on a real world model (see Fig. 6(b)) which requires cooperation in order to achieve goals. As a result, bot behaviour is limited to searching for other bots or recharging batteries when it is not part of a swarm. A bot will search out any other bot(s) which it detects within its sensor range (green circle). Once a swarm is formed bots have access to a larger action space. Specifically, the actions available to a swarm are a function of its currently perceived state. Moreover, the bots within the swarm will elect a leader based on experience gained within the ecosystem (experience is based on the length of time the bot has "survived" within the ecosystem). It is the leader who ultimately makes the decisions for the swarms actions based on knowledge about the current state gained from all members of the swarm, as well as, conditions within the swarm (*e.g.* low energy). Lastly, new leaders are elected quite frequently within a swarm due to new members joining (with more experience), swarms dispersing due to too many members, the threat of an adversary, or the removal of a swarm member by an adversary.

A swarm determines its current state as a function of the environment variables (*e.g.* weather conditions), average swarm battery charge, and whether the swarm is inspecting hydro equipment (see, *e.g.* Table 4). It is also assumed that each swarm knows the exact state of the environment. However, this is not to say that the state transition probabilities are known, only that the sensors of sbots have small error. Tables 4 and 5 lists all the states and actions available to a agent within the ecosystem. Lists of action spaces for a specific state are given in Table 7. Table 6 contains the rewards a swarm can receive.

| Symbol | Description |
|---|---|
| Tiny Disk | Bot |
| Number next to disk | Bot charge |
| Cluster of tiny disks | Swarm |
| Number next to cluster | Swarm charge |
| Green circle | Sensor range |
| Larger solid disk | Adversary |
| Black line | Power line |
| Solid square connected to power line | Power tower |

Table 3: Testbed Symbol Descriptions

Note, that the conditions are listed in order of priority. Finally, the environment used in the testbed is non-stationary, which means that transition probabilities change over time.




(a) Node (b) Component (c) Object (d) Dependency

(e) Interface

Figure 23: UML Notation

| State $i$ | State Description |
|---|---|
| 00 | Low energy; No danger; Weak sunlight; Mounted on sky wire. |
| 01 | Low energy; No danger; Plenty of sunlight; Mounted on sky wire. |
| 02 | Low energy; No danger; Weak sunlight; Not Mounted on sky wire. |
| 03 | Low energy; No danger; Plenty of sunlight; Not Mounted on sky wire; |
| 04 | Low energy; Danger adversary; Weak sunlight; Not Mounted on sky wire; |
| 05 | Low energy; Danger adversary; Plenty of sunlight; Not Mounted on sky wire; |
| 06 | Low energy; Danger lightning; Weak sunlight; Mounted on sky wire; |
| 07 | Low energy; Danger lightning; Plenty of sunlight; Mounted on sky wire; |
| 08 | Low energy; Danger high winds; Weak sunlight; Mounted on sky wire; |
| 09 | Low energy; Danger high winds; Plenty of sunlight; Mounted on sky wire; |
| 10 | High energy; No danger Mounted on sky wire; Tower detected. |
| 11 | High energy; No danger Mounted on sky wire; No Tower detected. |
| 12 | High energy; No danger Not mounted on sky wire; No Tower detected. |
| 13 | High energy; No danger Not mounted on sky wire; Tower detected |
| 14 | High energy; Danger adversary; Not mounted on sky wire; |
| 15 | High energy; Danger lightning; Mounted on sky wire; |
| 16 | High energy; Danger high winds; Mounted on sky wire; |
| 17 | Not in a swarm |

Table 4: RL States

| Action $i$ | Action Description |
|---|---|
| 00 | Error action (only used in the code) |
| 01 | Recharge |
| 02 | Shut down |
| 03 | Search |
| 04 | Search rapid |
| 05 | Hide |
| 06 | Undesirable |
| 07 | Flee |
| 08 | Shelter high winds |
| 09 | Inspect Tower |
| 10 | Report EMF |
| 11 | Mount tower |
| 12 | Dismount tower |

Table 5: Swarm Actions

| Reward Condition | Swarm Reward |
|---|---|
| Average Swarm Charge $<$ Threshold | 0.1 |
| Action = Search | 0.5 |
| Action = Inspect Tower | 1 |
| Action = Mount Tower | 1 |
| Action = Avoid Hazards | 1 |
| Everything Else | 0 |

Table 6: RL Reward Function

| State and Action Space |
|---|
| S = 00, A(00) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 01, A(01) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 02, A(02) = {1, 2, 3, 4, 5, 6, 7} |
| S = 03, A(03) = {1, 2, 3, 4, 5, 6, 7} |
| S = 04, A(04) = {1, 2, 3, 4, 5, 6, 7} |
| S = 05, A(05) = {1, 2, 3, 4, 5, 6, 7} |
| S = 06, A(06) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 07, A(07) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 08, A(08) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 09, A(09) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 10, A(10) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 11, A(11) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 12, A(12) = {1, 2, 3, 4, 5, 6, 7} |
| S = 13, A(13) = {1, 2, 3, 4, 5, 6, 7, 11} |
| S = 14, A(14) = {1, 2, 3, 4, 5, 6, 7} |
| S = 15, A(15) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 16, A(16) = {1, 2, 3, 4, 8, 9, 10, 12} |
| S = 17, A(17) = {1, 2, 3} |

Table 7: RL State Action Spaces

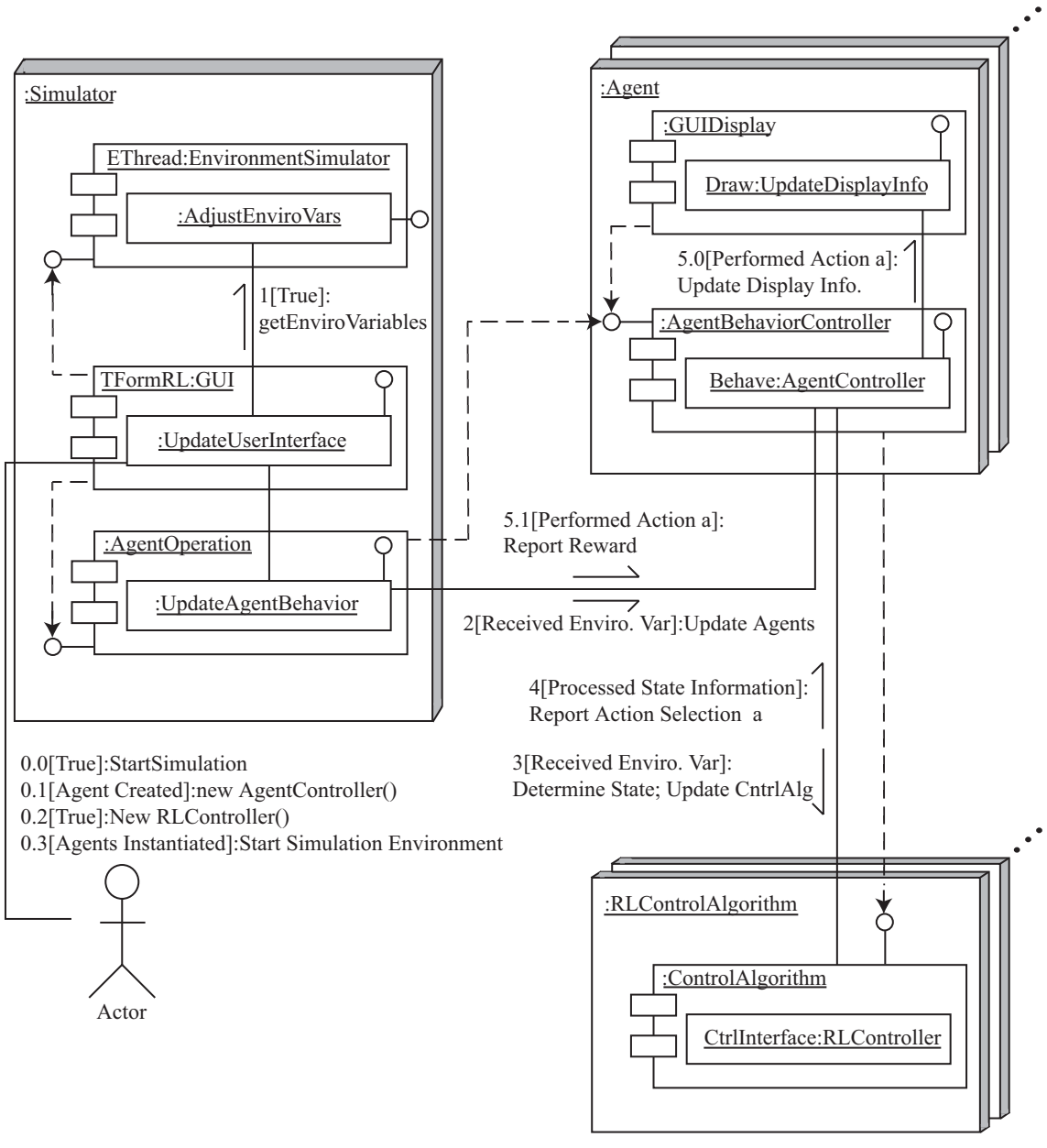



Figure 24: Static UML Model of Testbed Architecture (see, *e.g.*, [12])

# B Ecosystem Architecture

This Appendix describes the ecosystem software architecture through the use of a UML deployment diagram (see, *e.g.* [12, 32]). The main elements of a deployment diagram are given in Fig. 23 and are briefly introduced here. Nodes represent high level facets of a system which house component artifacts (see Fig. 23(a)). Components represent structure groupings of a system and are usually created once the project is nearing completion (see Fig. 23(b)). Objects are usually direct instances of system software or hardware and can be seen in Fig. 23(c). The dashed line with an arrow head in Fig. 23(d) represents a dependency and is used to designate that component A is dependent on component B. An interface represents operations which define the behaviour of an object or element (see Fig. 23(e)). Solid black lines connecting objects represent the links between objects. Messages are located next to object relationships and describe features and conditions of the deployment and usually indicate sequence. The lines with the half arrowhead represent the direction of the messages. Lastly, the following gives the convention used for naming and messaging.

$$\underline{\textit{Name:Type}}$$
$$\textit{Sequence \#[Condition]:Action}$$

A modular approach was used in order to facilitate the addition of control algorithms and to decouple the ecosystem code. Namely, each reinforcement algorithm was encapsulated in a class which implements a control interface. For instance, one requirement of the interface is for objects to report the previous state-action pair, the resultant reward, and the current state-action pair at the end of each time step. The main benefit of this approach is the ability to use the same instance of a bot regardless of the algorithm used to control it. For example, Fig. 24 is a deployment diagram of the main architectural components of the testbed including the modular control algorithm design. Notice that the control algorithm is a separate node than the agent. As a result, the agent creates (instantiates) the desired control algorithm at start up. Furthermore, the control algorithm is transparent to the agent because they all implement the control interface (*i.e.* that the agent does not need to know the type of algorithm used to control it). The dots at the top right of the nodes in Fig. 24 indicate more than one agent is associated with the simulator (remember that multiple agents form swarms). Next, note the dependencies in Fig. 24. The main responsibility of the GUI in this system is to update the environment and the spatial properties of all the objects within the simulation. Consequently, it is dependent on the thread which is responsible for the environment variables, as well as, the simulation component which is responsible for updating the agents. Lastly, Fig. 24 gives the deployment specifications for one iteration of the ecosystem testbed. Step 0 involves starting the

ecosystem simulation and steps 2 - 5.1 are repeated infinitely. They represent the basic operations of the simulation and are given in more detail below.

`Repeat forever` {*after selecting RL method, and starting ecosystem* }
  `For each time step`
    Determine environment variables (*e.g.*, lightning, wind, sunshine, emf).
    `If` end of episode reached, `then` report it to agents.
    `Update` ecosystem environment variables, and adjust swarm (s, a).
    `Adjust` agents (including swarm members) position, energy.
    `Report` swarm reward based on action performed and environment state.
    `Perform` any necessary clean up (*e.g.*, remove dormant agents).

# Notation

| | |
|---|---|
| $\delta$ | Temporal Difference (TD) Error |
| $Pr(X = x)$ | Probability that $X$ equals $x$ |
| $R$ | Return |
| $E[R]$ | Expected value of $R$ |
| $IS$ | Information System |
| $(U, A)$ | Represents an information system where $U$ is a non-empty finite set of elements and $A$ is a non-empty finite set of attributes |
| $a \in A$ | An attribute which is a member of a set of attributes for an element |
| $V_a$ | The value set of an attribute $a$ (*i.e.* $V_a = a(x)$ where $x \in U$) |
| $Ind_{IS}(B)$ | Indiscernibility relation with respect to attribute set $B$ |
| $U/Ind_{IS}(B)$ | A partition of $U$ created by the indiscernibility relation |
| $B(x)$ | Block (a set of $B$-indiscernible elements) |
| $B_*X$ | Lower approximation of the set $X$ based on the attributes in $B$ |
| $B^*X$ | Upper approximation of the set $X$ based on the attributes in $B$ |
| $(U, Ind)$ | Approximation Space |
| $GAS$ | Generalized Approximation Space |
| $(U, N, \nu)$ | A system representing a GAS where $U$ is a non-empty finite set of objects, $N$ is a neighbourhood function, and $\nu$ is an overlap function. |
| $AS_B$ | A parameterized approximation space $(U, N_B, \nu)$ where the neighbourhood function, $N_B$, is defined as a block of $B$-indiscernible elements. |
| $\mathcal{P}(U)$ | The powerset of $U$ |
| $\nu_{SRI}(X, Y)$ | Standard Rough Inclusion |
| $\nu_{SRC}(X, Y)$ | Standard Rough Coverage |
| $AS_L$ | A lower approximation space defined in the context of a decision system |
| $(U, A, d)$ | Represents a decision system which is the same as an information system with an additional distinguished attribute representing a decision. |
| $B_{ac}(x)$ | Notation used to specify a specific block containing feature value $ac$ |
| $B_*D$ | Lower approximation of the decision set $D$ |
| $\nu_B(X, Y)$ | Short hand notation for $\nu_{SRC}(X, B_*Y)$ |
| $AS_U$ | An upper approximation space defined in the context of a decision system |

| | |
|---|---|
| $D$ | Decision set containing all elements which have been accepted (*i.e.* $d(x) = 1$) |
| $s$ | State |
| $a$ | Action |
| $S$ | Set of all states |
| $A(s)$ | Set of all actions for state $s$ |
| $\pi(s, a)$ | Policy indicating the probability of taking action $a$ in state $s$ (denoted $\pi$ for short) |
| $V^\pi(s)$ | The value of a state $s$ obtained by following policy $\pi$ |
| $Q^\pi(s, a)$ | The value of taking $a$ in state $s$ while following policy $\pi$ |
| $V(s)$ | Estimate of $V^\pi(s)$ |
| $Q(s, a)$ | Estimate of $Q^\pi(s, a)$ |
| $r$ | Reward |
| $\Re$ | Set of all real numbers |
| $R_m$ | Return (cumulative future discounted reward) |
| $\gamma$ | Discount rate |
| $E^\pi[R_m]$ | Expected discounted return following policy $\pi$ |
| $j$ | Average difference between pixel intensities |
| $\bar{x}, \bar{y}$ | Centroid coordinates |
| $p(s, a)$ | Preference for action $a$ in state $s$ |
| $\bar{r}$ | Reference reward |
| $\alpha, \beta$ | Step size parameters |
| $\mathcal{B}$ | Set of all blocks consisting of elements in $U$ |
| $Count(s)$ | Number of times state $s$ was encountered during an episode |
| $w_k/w_m$ | Weight used in weighted average |
| $\pi$ | policy, a decision-making rule |
| $\pi(s)$ | Deterministic policy, a mapping from state $s$ to an action $a$ |
| $\pi(s, a)$ | Stochastic policy, a mapping from state $s$ to the probability of performing action $a$ |
| $p_i(s)$ | The probability a complete sequence occurs under a policy |
| $P^{a_k}_{s_k s_{s+1}}$ | Probability that a system in state $s_k$ will be in state $s_{k+1}$ after its next transition given that action $a_k$ was selected |
| $\bar{\nu}_{ac}$ | Average rough coverage of all the blocks containing action $ac$ (where $ac$ differs from $a$ because the decision system is discretized before setting up a lower approximation space) |
| $DT$ | Decision Table (Decision System) |
| $argmax_a$ | The value of the given argument for which the expression is maximal |

# Glossary

| | |
|---|---|
| Action | An action taken by an agent causes a change in the environment. |
| Action Value Function | The expected discounted future reward starting in state $s$, selecting action $a$, and continuing to follow policy $\pi$. |
| Actor | Policy structure used to select actions in the Actor-Critic method. |
| Actor-Critic Method | A temporal difference (TD) learning method with a separate memory structure to represent policy independent of the value function. |
| Agent | A system that has sensors that enable it to perceive, actuators, and the ability to perform actions that modify its environment. |
| Approximation Space | A universe $U$ together with the lower and upper approximations (characterized by the $Ind_{IS}$ relation). An approximation space gives an approximate description for any $X \subseteq U$. |
| Attribute | See Feature |
| Behaviour Policy | A policy used to select actions. |
| Block | A set of indiscernible elements based on the attributes contained in $B$ (where $B \subseteq A$). |
| Coverage | Used relative to the extent that a given set is contained in a standard set. |
| Critic | Structure which learns and critiques the policy used by the actor to make action selections in the Actor-Critic method. |

| | |
|---|---|
| Credit Assignment Problem | Finding the action for a specific state that will most likely give the highest reward. |
| Data Table | See Information System |
| Decision System | An information system $IS = (U, A, d)$ where $d$ is a distinguishing attribute representing a decision. |
| Deterministic Policy | A policy that assigns an action to a state. |
| Discounting | The process of determining the present value of future rewards. |
| Discretization | A process of assigning intervals to the real numbers contained in a column of an information system in order to facilitate the extraction of patterns. |
| $\epsilon$-Greedy Method | An action selection method which assigns the probability $\epsilon$ to the action with the highest value estimate and equally assigns the probability $1 - \epsilon$ to the remaining actions. |
| Estimation Policy | A policy which evaluates the actions selected under the behaviour policy. |
| Ethogram | An Ethogram is a catalogue of descriptions of separate and distinct species-typical behaviour patterns. |
| Ethology | The study of the behaviour and interactions of animals. |
| Feature | An abstraction that characterizes a part of an object of interest. |
| Feature (Behaviour Feature) | An abstraction that characterizes some aspect of a behaviour. |
| Feature (System Feature) | An abstraction that characterizes a part of a system. |

| | |
|---|---|
| Generalized Approximation Space | Is a system $GAS$ = ($U$, $N$, $\nu$) where $U$ is a non-empty set of objects, $N$ is a neighbourhood function, and $\nu$ is an overlap function. |
| Greedy Action Selection | Always selecting the action which is most likely to produce the highest reward. |
| Indiscernibility Relation | Two elements $x$ and $x'$ are indiscernible from each other if their associated feature sets are the same. |
| Information System | A pair ($U$, $A$), where $U$ is a non-empty, finite set of elements and $A$ is a non-empty, finite set of attributes |
| Intelligent Agent | An agent that recognizes patterns and learns over time to choose beneficial actions. |
| Lower Approximation | The lower approximation for a sample $X \subseteq U$ is a collection of blocks of sample elements which can be classified with full certainty as members of $X$ based on the knowledge represented by attributes in $B$. |
| Monte Carlo Method | Estimate $E[R]$ with average R-value. |
| Non-Greedy Action Selection | Selecting an action with unknown or low reward expectation in order to make better action selections in the future. |
| Non-Stationary Models | Models in which the state transition probabilities change over time. |
| Off-Line Learning | Learning which occurs before the agent is placed in the environment in which it is to act. |
| Off-Policy | A control algorithm where the policy that is being revised is not the one being used to make decisions. |

| | |
|---|---|
| On-Line Learning | Learning which occurs in real time with the agent constantly adding to its experience in order to make better action selections in the future. |
| On-Policy | A control algorithm where the behaviour and estimation policies are one and the same. |
| Pattern | A feature-value vector. |
| Policy | A decision-making rule |
| Preference | A measure of the desirability of a state-action pair used to determine a policy by the Reinforcement Comparison and Actor-Critic methods. |
| Proximate Cause | The preceding event(s) which which may induce a particular behaviour. |
| Reference Reward | Standard or reference used for judging the desirability of an action. |
| Reinforcement | A stimulus which increases/decreases the likelihood of an action being selected in the future. |
| Reinforcement Comparison Method | Methods which use a reference reward to judge the desirability of a reward. |
| Reinforcement Learning | The process of learning the correct action to take based on feedback from the environment. |
| Reward | Value assigned to a state or action which represents its desirability. |
| Reward Signal | Measured response from the environment. |

| | |
|---|---|
| Rough Coverage | A measure of the degree with which a given set $X$ is covered by a set $Y$ (where it is always assumed that $\|X\| \leq \|Y\|$). |
| Rough Inclusion | A measure of the degree with which a given set $X$ is included in a set $Y$ (where it is always assumed that $\|X\| \leq \|Y\|$). |
| Rough Set | Is an approximation of a sample $X \subseteq U$ defined in terms of the upper and lower approximation of the sample $X$. |
| Softmax Method | An action selection method which assigns probabilities based on value estimates where the action with the highest value estimate is assigned the largest probability according to the Gibbs or Boltzmann distribution. |
| State | A unique interpretation of the environment from which an action $a \in A(s)$ is selected. |
| State Value Function | The value of a state $s$ obtained by following policy $\pi$. |
| Stationary Models | Models in which the state transition probabilities are fixed. |
| Stochastic Policy | Assigns a probability of taking action $a$ in state $s$. |
| Survival Value | A measure of the degree to which an observed animal behaviour contributes to survival. |
| Swarm | A collection of cooperating organisms which can be viewed as a collective entity. |
| Swarm Intelligence | A form of artificial intelligence based on the collective behaviour of decentralized systems. |
| Swarmbot | A collection of small robots which work together as a whole. |

| | |
|---|---|
| Temporal Difference (TD) Error | The difference obtained between value estimates obtained at different times. |
| Update Rule | A form of incremental average. |
| Upper Approximation | The upper approximation for a sample $X \subseteq U$ is a collection of blocks of sample elements representing both certain and possibly uncertain knowledge about $X$. |
| Value Set | A function which defines for every element $x \in U$ a set of feature values. |

# Index

# References


[1] A.G. Barto, S. Mahadevan, Recent advances in hierarchical reinforcement learning, *Discrete Event Dynamic Systems: Theory and Applications*, **13**, 2003, 41-77.

[2] B.M. Blumberg, P.M. Todd, P. Maes, *No Bad Dogs: Ethological Lessons for Learning in Hamsterdam*, Research Report, MIT Media Lab, Massachusetts Institute of Technology, 1995.

[3] E. Bonabeau, M. Dorigo, G. Theraulaz, *Swarm Intelligence. From Natural to Artificial Systems*, (UK: Oxford University Press, 1999).

[4] P. Dayan, C.J.C.H. Watkins, Reinforcement learning, Encyclopedia of Cognitive Science. MacMillan Press, UK, 2001.

[5] P. Dayan, Reinforcement comparison. In: Touretzky D. S., Elman, J. L., Sejnowski, T. J., and Hinton, G. E. (Eds.), Proc. of the 1990 Summer School, Morgan Kaufmann, San Mateo, 1991, 45-51.

[6] T.G. Dietterich, Hierarchical reinforcement learning with the maxq value function decomposition, *Journal of Artificial Intelligence Research*, **13**, 2000, 227-303.

[7] M. Dorigo, M. Colombetti, Robot Shaping. *An Experiment in Behavior Engineering*. The MIT Press, Cambridge, MA, 1998.

[8] C. Gaskett, *Q-Learning for Robot Control*. Ph.D. Thesis, Supervisor: A. Zelinsky, Department of Systems Engineering, The Australian National University, 2002.

[9] A. Gomolinska: Rough validity, confidence, and coverage of rules in approximation spaces. In J.F. Peters, A. Skowron, D. van Albada(Eds.), Transactions on Rough Sets III *Lecture Notes in Computer Science*, **3400**, 57-81, 2005

[10] R.C. Gonzalez, R.E. Woods: Digital Image Processing, 2nd Ed. Prentice Hall, N.J., 2002.

[11] R. Gross, M. Dorigo, Cooperative transport of objects of different shapes and sizes. In M. Dorigo, M. Birattari, C. Blum, L.M. Gambardella, F. Mondada, T. Stutzle(Eds.), Ant Colony Optimization and Swarm Intelligence *Lecture Notes in Computer Science*, **3172**,106-117,2004.

[12] J. Holt, *UML for Systems Engineering watching the wheels*, The Institution of Electrical Engineers, London, UK, 2001.

[13] C. Huygens,: De Ratiociniis in Ludo Aleae (On Reasoning or Computing in Games of Chance), 1657.

[14] L.P. Kaelbling, *Learning in Embedded Systems*,The MIT Press, Cambridge, MA, 1992.

[15] L.P. Kaelbling, Hierarchical learning in stochastic domains: Preliminary results. In *Proc. of the* $10^th$ *International Conference on Machine Learning*, 1993, 167-173.

[16] L.P. Kaelbling, Acting optimally in partially observable stochastic domains. In *Proc.* $12^th$ *Nat. Conf. on Artificial Intelligence*, 1996.

[17] L.P. Kaelbling, M.L. Littman, A.W. Moore,Reinforcement learning: A survey *Journal of Artificial Intelligence Research*, **4**, 1996, 237-285.

[18] P.R. Killeen, Mathematical principles of reinforcement, *Behavioral and Brain Science*, **17**, 1994, 105-172.

[19] A.C. Kolle, *The Nature of Learning–A Study of Reinforcement Learning Methodology*, Ph.D. Thesis, supervisor: P. Johansen, University of Copenhagen, 21 Nov. 2003.

[20] J. Komorowski, Z. Pawlak, L. Polkowski, A. Skowron, Rough sets: A tutorial, in S.K. Pal, A. Skowron (Eds.) *Rough Fuzzy Hybridization. A New Trend in Decision-Making* (Singapore: Springer-Verlag Singapore Pte. Ltd., 1999), 3-98.

[21] P.N. Lehner, *Handbook of ethological methods*, Cambridge University Press, Cambridge, UK, 1979.

[22] L. Li, *Distributed Learning in Swarm Systems: A Case Study*, M.Sc. Thesis, supervisor: Y.S. Abu-Mostafa, California Institute of Technology, 2002.

[23] L. Li, V. Bulitko, R. Greiner, Batch reinforcement learning with state importance, Research Report, University of Alberta, 2004.

[24] L.J. Lin, T.M. Mitchell, *Memory Approaches to Reinforcement Learning in Non-Markovian Domains*, Research Report CMU-CS-92-138, Carnegie-Mellon University, 1992.

[25] D. Lockery: *Reinforcement Learning Methods*. Research Report CIL02.11052005, Computational Intelligence Laboratory, Department of Electrical and Computer Engineering, University of Manitoba, 11 April 2005.

[26] F. Lu, *Exploring Model-Based Methods for Reinforcement Learning*, Ph.D. Thesis, supervisor: D. Schuurmans, University of Waterloo, Waterloo, Ontario, Canada, 2003.

[27] R. Maclin, J.W. Shavlik, Creating advice-taking reinforcement learners, *Machine Learning*, **22**(1-3), 1996, 251-281.

[28] A.K. McCallum, *Reinforcement Learning with Selective Perception and Hidden State*, Ph.D. Thesis, University of Rochester, U.S.A., 1996.

[29] G.A. Mikhailov, Monte-Carlo method. In: M. Hazelwinkel(Ed.), *Encyclopedia of Mathematics*, **4**, Kluwer Aca. Pub., Dordrecht, 1995.

[30] H.S. Nguyen, *Discretization of Real Value Attributes, Boolean Reasoning Approach*, supervisor: A. Skowron, Warsaw University, 1997.

[31] S.H. Nguyen: Regularity Analysis and Its Applications in Data Mining, Doctoral Thesis, superisor: Bogdan S. Chlebus, Faculty of Mathematics, Computer Science and Mechanics, Warsaw University, July 1999.

[32] Object Management Group, UML Standard 1.5, http://www.uml.org/

[33] E. Orłowska: Semantics of Vague Concepts. Applications of Rough Sets. Institute for Computer Science, Polish Academy of Sciences Report 469, March 1982

[34] Z. Pawlak: Classification of Objects by Means of Attributes. Institute for Computer Science, Polish Academy of Sciences Report 429, March 1981

[35] Z. Pawlak: Rough Sets. Institute for Computer Science, Polish Academy of Sciences Report 431, March 1981

[36] Z. Pawlak, Rough sets, *International J. Comp. Inform. Science*, **11**, 1982, 341–356

[37] Z. Pawlak, *Rough Sets. Theoretical Reasoning about Data*, Theory and Decision Library, Series **D**: System Theory, Knowledge Engineering and Problem Solving, vol. **9**, Kluwer Academic Pub., Dordrecht, 1991.

[38] J. F. Peters, Approximation space for intelligent system design patterns. *Engineering Applications of Artificial Intelligence*, **17(4)**, 2004, 1–8.

[39] J.F. Peters, Approximation spaces for hierarchical intelligent behavioural system models. In: B.D.-Keplicz, A. Jankowski, A. Skowron, M. Szczuka (Eds.), Monitoring, Security and Rescue Techniques in Multiagent Systems, *Advances in Soft Computing*, (Heidelberg: Physica-Verlag, 2004) 13–30

[40] J.F. Peters, Rough ethology: Towards a Biologically-Inspired Study of Collective Behavior in Intelligent Systems with Approximation Spaces. *Transactions on Rough Sets*, **III**, LNCS 3400, 2005, 153-174.

[41] J.F. Peters: Approximation spaces in off-policy Monte Carlo learning. Plenary paper in T. Burczynski, W. Cholewa, W. Moczulski (Eds.), Recent Methods in Artificial Intelligence Methods, AI-METH Series, Gliwice, 2005, 139-144.

[42] J.F. Peters, T.C. Ahn, M. Borkowski, V. Degtyaryov, S. Ramanna, Line-crawling robot navigation: A rough neurocomputing approach. In: C. Zhou, D. Maravall, D. Ruan (Eds.), Autonomous Robotic Systems. *Studies in Fuzziness and Soft Computing* **116** (Heidelberg: Springer-Verlag, 2003) 141–164

[43] J.F. Peters, C. Henry, S. Ramanna, Rough Ethograms : Study of Intelligent System Behavior. In: New Trends in Intelligent Information Processing and Web Mining (IIS05), Gdańsk, Poland, June 13-16 (2005)

[44] J.F. Peters, C. Henry, S. Ramanna, Reinforcement learning with pattern-based rewards. In: Proc. Fourth Int. IASTED Conf. Computational Intelligence (CI 2005), Calgary, Alberta, Canada (4-6 July 2005) 267-272

[45] J.F. Peters, C. Henry, S. Ramanna, *Reinforcement Learning in Swarms that Learn*, Research Report, Computational Intelligence Laboratory, University of Manitoba, 2005.

[46] J.F. Peters, D. Lockery, S. Ramanna: Monte Carlo off-policy reinforcement learning: A rough set approach. In *Proc. Fifth Int. Conf. on Hybrid Intelligent Systems*, Rio de Janeiro, Brazil, 06–09 Nov. 2005, 187-192.

[47] J.F. Peters, A. Skowron, P. Synak, S. Ramanna,Rough sets and information granulation. In: Bilgic, T., Baets, D., Kaynak, O. (Eds.), Tenth Int. Fuzzy Systems Assoc. World Congress IFSA, Instanbul, Turkey, *Lecture Notes in Artificial Intelligence* **2715** (Heidelberg: Springer-Verlag, 2003) 370–377

[48] J.F. Peters, S. Ramanna, Measuring acceptance of intelligent system models. In: M. Gh. Negoita et al. (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, *Lecture Notes in Artificial Intelligence*, **3213**, Part I, 2004,764–771.

[49] L. Polkowski, *Rough Sets. Mathematical Foundations*. (Heidelberg: Springer - Verlag, 2002).

[50] D. Precup,R.S. Sutton,S. Singh, Eligibility traces for off-policy evaluation. In: *Proc. 17th Conf. on Machine Learning*(ICML 2000),Morgan Kaufmann, San Francisco, 2000, 1-8.

[51] J. Randlov, *Solving Complex Problems with Reinforcement Learning*, Ph.D. Thesis, University of Copenhagen, 2001.

[52] C.P. Robert, G. Casella: *Monte Carlo Statistical Methods*, 2nd Ed. Springer, Berlin, 2004.

[53] R.Y., Rubinstein: *Simulation and the Monte Carlo Method*. John Wiley & Sons, Toronto, 1981

[54] G.A. Rummery, *Problem Solving with Reinforcement Learning*, Ph.D. Thesis, Cambridge University, 1995.

[55] H. Sevay, *Multiagent Reactive Plan Application Learning in Dynamic Environments*, Ph.D. Thesis, supervisor: C. Tsatsoulis, University of Kansas, 2003.

[56] S.P. Singh, Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, **8**(3-4), 1992, 323-339.

[57] A. Skowron, Rough sets and vague concepts. *Fundamenta Informaticae*, **XX**, 2004, 1-15.

[58] A. Skowron, J. Stepaniuk, Modelling complex patterns by information systems. *Fundamenta Informaticae*, **XXI**, 2005, 1001-1013.

[59] A. Skowron, J. Stepaniuk, Generalized approximation spaces. In: Lin, T.Y.,Wildberger, A.M. (Eds.), *Soft Computing, Simulation Councils*, San Diego, 1995, 18–21

[60] A. Skowron, J. Stepaniuk, Information granules and approximation spaces, in *Proc. of the $7^{th}$ Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems* (IPMU98), Paris, 1998, 1354–1361

[61] Skowron, A., Swiniarski, R., Synak, P.: Approximation spaces and information granulation, Transactions on Rough Sets III, 2005, 175-189

[62] W.D. Smart, *Making Reinforcement Learning Work on Real Robots*, Ph.D. Thesis, Brown University, 2002.

[63] P. Stone, Team partitioned, opaque-transition reinforcement learning. In: M. Asada, H. Kitano (Eds.),*RoboCup-98: Robot Soccer World Cup II*,Springer-Verlag, Berlin, 1999.

[64] P. Stone, Layered Learning in Multiagent Systems. A Winning Approach to Robotic Soccer. The MIT Press, Cambridge, MA, 2000.

[65] J. Stepaniuk, Approximation spaces, reducts and representatives, in L. Polkowski and A. Skowron (Eds.), Rough Sets in Knowledge Discovery 2, *Studies in Fuzziness and Soft Computing* **19**(Heidelberg: Springer-Verlag, 1998)., 109–126

[66] R.S. Sutton, Temporal Credit Assignment in Reinforcement Learning, Ph.D. Thesis, University of Massachusetts, Amherst, MA., 1984.

[67] R.S. Sutton, Learning to predict by the methods of temporal differences. *Machine Learning*, **3**, 1988, 9-44.

[68] R.S. Sutton, Reinforcement learning architectures for animats. In: J.A. Meyer, S.W. Wilson (Eds.),*From Animals to Animats, Proc.* $1^{st}$ *Int. Conf. Simulation of Adaptive Behavior*, The MIT Press, Cambridge, MA, 1991.

[69] R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* The MIT Press, Cambridge, MA, 1998.

[70] N. Tinbergen, On aims and methods of ethology, *Zeitschrift für Tierpsychologie* **20**, 1963, 410–433

[71] T. Tyrrell, *Computational Mechanisms for Action Selection*, Ph.D. Thesis, Centre for Cognitive Science, University of Edinburgh, 1993.

[72] N. Metropolis, S. Ulam: The Monte Carlo method, *Journal of the American Statistical Association*, **44**(247), 1949, 335-341.

[73] S. Ulam: On the Monte Carlo method. In *Proc. 2nd Symposium on Largescale Digital Calculating Machinery*, 1951, 207-212

[74] C.J.C.H. Watkins, *Learning from Delayed Rewards*, Ph.D. Thesis, King's College, Cambridge, 1989.

[75] C.J.C.H. Watkins, P. Dayan, Technical note: Q-learning, *Machine Learning*, **8**, 1992, 279-292.

[76] M. Weinberg, J.S. Rosenschein, Best-response multiagent learning in non-stationary environments. In: Proc. Int. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS04), New York, N.Y., U.S.A., 19-23 July, 2004, 1-8.

[77] M. Wiering, Explorations in Efficient Reinforcement Learning, Ph.D. Thesis, University of Amsterdam, 1999.

[78] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning.,*Machine Learning*, **8**, 1992, 229–256.